WeightWatcher

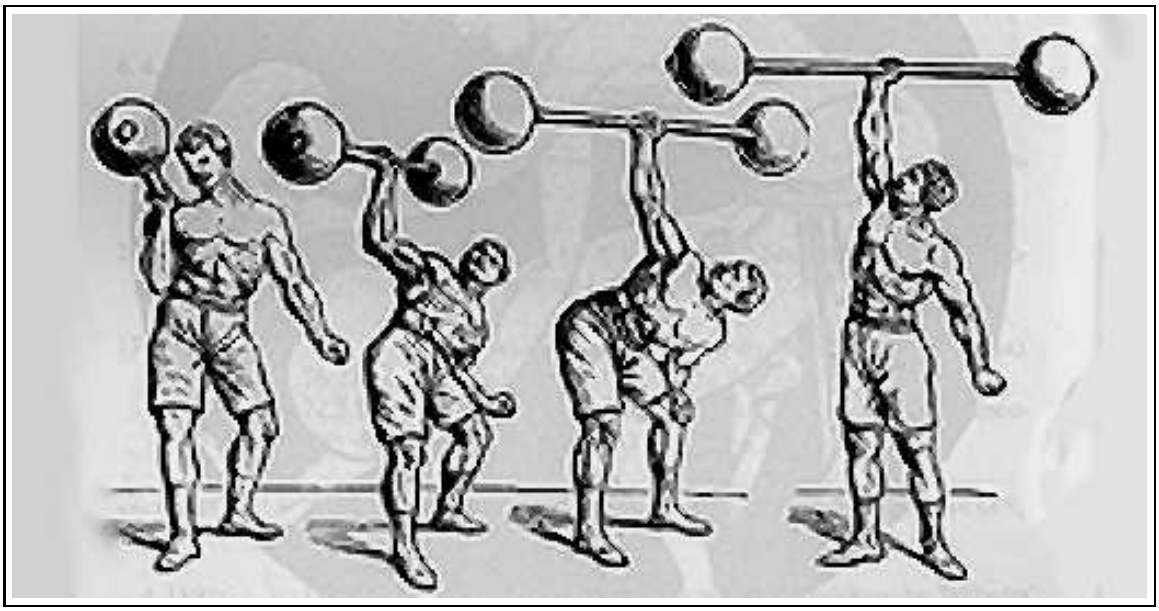
$v1.8$

User's guide

E. BERTIN

C. MARMO

Institut d'Astrophysique



Contents

1 What is WeightWatcher?

WeightWatcher (or simply WW) is a program that combines weight-maps, flag-maps, and vector data in order to produce control maps which can directly be used by astronomical image-processing packages like Drizzle[1], SExtractor or SWarp. Weight-thresholding and/or specific flag selections are applied by WW through a configuration file: this alleviates other programs from such interpretation work. WeightWatcher will mostly be useful as part of an imaging survey pipeline. Its main features are:

- Processing speed: limited by the I/O performances of the machine (typically 50 Mpixel/s on a workstation),
- Ability to work with very large images (up to, say, $10^8 \times 10^9$ pixels on a 64 bit system),
- FITS format (including Multi-Extension) is used for input and output. Output flag-map format selection is automatic (8, 16 or 32bits),
- Up to 30 weight-maps, 30 flag-maps, and thousands of polygons can be handled simultaneously.
- Automatic rasterizing of DS9 `.reg` files,
- Statistics of flagged and weighted areas,
- Metadata output in XML-VOTable format.

2 Installing the software

2.1 Software and hardware requirements

WeightWatcher has been developed on Unix machines (SUN, Compaq Tru-64 and GNU/Linux), and should compile on any POSIX-compliant system. The software is run in (ANSI) text-mode from a shell. A window system is therefore unnecessary with present versions.

Memory requirements are fairly modest in most cases, as they do not depend on the size of the output images. Count a few MB per input image.

2.2 Obtaining WeightWatcher

The easiest way to obtain Weightwatcher is to download it from an internet site. The current official anonymous FTP site is `ftp://ftp.iap.fr/pub/from_users/marmo/weightwatcher/`. There can be found the latest versions of the program as standard `.tar.gz` Unix source archives, including the documentation, and Linux binaries as RPM packages. For production, it is strongly advised to install the RPM packages if you are running Linux on a machine with x86 of x86-64 architecture and RPM-support, as they contain a strongly optimized version of the code.

2.3 Installation

To install, you must first uncompress and unarchive the archive:

```
gzip -dc weightwatcher-x.x.tar.gz | tar xvf -
```

A new directory called `weightwatcher-x.x` should now appear at the current position on your disk. You should then just enter the directory and follow the instructions in the file called "`INSTALL`".

The software is also available as a precompiled RPM for Linux systems with an x86 or x86-64 architecture. The simplest way to install an RPM package is to log as root and use the following command

```
rpm -U weightwatcher-x.x-dist.arch.rpm
```

3 Overview of the software

The layout of WeightWatcher is displayed in Fig. 1. One can distinguish 5 main operations which are controlled by sets of configuration parameters: `WEIGHT_MIN` and `WEIGHT_MAX` for threshold selection; `FLAG_MASKS` and `WEIGHT_MASKS` for bit masking; `WEIGHT_OUTFLAGS`, `FLAG_OUTFLAGS` and `POLY_OUTFLAGS` for flag assignments; all value assignments are defaulted to 0.0 in the present version.

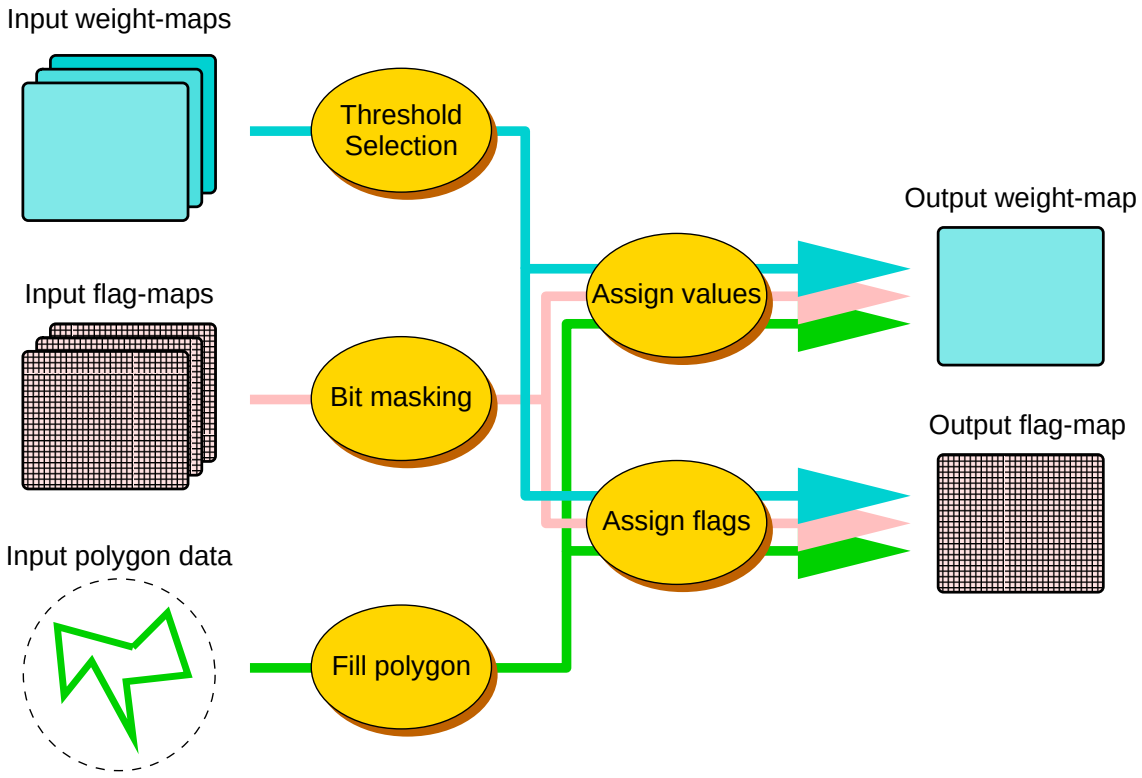


Figure 1: Global layout of WeightWatcher.

The assignment of output flags is left to the user, who should care about possibly overlapping flag-bits. Both the "Assign-values" and "Assign-flags" operations in Fig. 1 *act as the logical "OR"*. For instance, one "bad" pixel in one of the input weight-maps is enough to have the output weight-map set to zero at the same pixel location. Likewise, one flag surviving the "Bit-masking" operation is enough to trigger any bit in the output flag-map.

4 Using WeightWatcher

WeightWatcher is run from the shell with the following syntax:

% `ww -c` *configuration-file* `[ -`*Parameter1 Value1* `] [ -`*Parameter2 Value2* `]` ...

The part enclosed within brackets is optional. Any "`-`*Parameter Value*" statement in the command-line overrides the corresponding definition in the configuration-file or any default value (see below).

4.1 The Configuration file

Each time WeightWatcher is run, it looks for a configuration file. If no configuration file is specified in the command-line, it is assumed to be called "`default.ww`" and to reside in the current directory. If no configuration file is found, WW will use its own internal default configuration.

4.1.1 Creating a configuration file

WeightWatcher can generate an ASCII dump of its internal default configuration, using the "-d" option. By redirecting the standard output of WeightWatcher to a file, one creates a configuration file that can easily be modified afterward:

% `ww -d >default.ww`

A more extensive dump with less commonly used parameters can be generated by using the "`-dd`" option.

4.1.2 Format of the configuration file

The format is ASCII. There must be only one parameter set per line, following the form:

Config-parameter *Value(s)*

Extra spaces or linefeeds are ignored. Comments must begin with a "#" and end with a linefeed. Values can be of different types: strings (can be enclosed between double quotes), floats, integers, keywords or boolean (`Y/y` or `N/n`). Some parameters accept zero or several values, which must then be separated by commas. Integers can be given as decimals, in octal form (preceded by digit `0`), or in hexadecimal (preceded by `0x`). The hexadecimal format is particularly convenient for writing multiplexed bit values such as binary masks. Environment variables, written as `$HOME` or `${HOME}` are expanded.

4.1.3 Parameter list

Here is a list of all the parameters known to WeightWatcher. Please refer to §4.2 for a detailed description of their meaning.

Parameter	default	type	Description
WEIGHT_NAMES	weightin.fits	*string(s)*	Filename(s) of the input weight-map(s).
WEIGHT_MIN	0.3	*float(s)*	Weight-pixels below these thresholds will be flagged.
WEIGHT_MAX	1.30	*float(s)*	Weight-pixels above these thresholds will be flagged.
WEIGHT_OUTFLAGS	1	*integer(s)*	Flag values given to thresholded pixels.
FLAG_NAMES	flagin.fits	*string(s)*	Filename(s) of the input flag-map(s).
FLAG_WMASKS	0xff	*integer(s)*	Flag-bits which will affect the weight-map pixels.
FLAG_MASKS	0x01	*integer(s)*	Bits which will be converted as output flags.
FLAG_OUTFLAGS	2	*integer(s)*	Translation of the FLAG_MASKS bits.
POLY_NAMES	""	*string(s)*	Filename(s) of the input polygon descriptions.
POLY_OUTFLAGS		*integer(s)*	Flag values given to pixel *inside* polygons.
POLY_OUTWEIGHTS	0.0	*float(s)*	Weight factor given to pixel *inside* polygons.
POLY_INTERSECT	Y	*boolean*	Do you want to flag polygon intersections?
OUTWEIGHT_NAME	weight.fits	*string*	Filename of the output weight-map.
OUTFLAG_NAME	flag.fits	*string*	Filename of the output flag-map.
GETAREA	N	*boolean*	Compute area statistics.
GETAREA_WEIGHT	0.0	*float*	Threshold above which weight pixels are counted.
GETAREA_FLAGS	1	*integer(s)*	Bit mask for flagged pixels excluded of count statistics.
MEMORY_BUFSIZE	256	*integer*	Buffer size, in image lines.
VERBOSE_TYPE	NORMAL	*keyword*	How much WeightWatcher comments its operations:
		QUIET	– run silently,
		NORMAL	– display warnings and limited information concerning the work in progress,
		FULL	– display more complete information.
XML_NAME	ww.xml	*string*	Filename for XML output.
XSL_URL	ww.xsl	*string*	Filename for XSL style-sheet.
WRITE_XML	Y	*boolean*	Write XML file (Y/N)?
NTHREADS	1	*integer*	1 single thread, WeightWatcher is not yet parallelized.

4.2 Detailed description of the configuration parameters

4.2.1 Processing of the input weight-map(s)

WEIGHT_NAMES - Filename(s) of the weight-maps provided as input to WW. Any 2-dimensional FITS image may be used as a weight-map; integer images are automatically converted to floating-point format. If several images are requested, they must all have the same width and the same height. A null string ("") can be provided, in which case no input weight-map will be used. *Note that only the 1st weight-map will have its pixel values copied to the output weight-map.*

WEIGHT_MIN - Lower threshold(s) applied to the input weight-map(s). There must be one value per WEIGHT_NAMES filename. Pixels below these values will trigger flagging in the output flag-map

and will set to zero the corresponding output weight-map pixel.

`WEIGHT_MAX` - Same as `WEIGHT_MIN` but for upper threshold(s).

`WEIGHT_OUTFLAGS` - Flag(s) that will be OR'ed in the output flag-map in case of threshold overtaking. There must be one value per `WEIGHT_NAMES` filename.

4.2.2 Processing of the input flag-map(s)

`FLAG_NAMES` - Filename(s) of the flag-maps provided as input to WW. Any 2-dimensional FITS image with pixels coded as integers[1] (`BITPIX =` 8, 16 or 32) may be used as a weight-map. If several images are requested, they must all have the same width and the same height. A null string (`""`) can be provided, in which case no input flag-map will be used.

`FLAG_WMASKS` - Mask(s) that will be applied to the input flag-map(s). If the result is non-zero, the corresponding pixel in the output weight-map will be set to zero. There must be one mask per `FLAG_NAMES` filename.

`FLAG_MASKS` - Mask(s) that will be applied to the input flag-map(s). For each bit which passes the masking, a flag (specified by the `OUTPUT_FLAGS` parameter below) will be OR'ed with the corresponding pixel in the output flag-map. There must be one mask per `FLAG_NAMES` filename.

`FLAG_OUTFLAGS` - Flag value(s) that will be OR'ed in the output flag-map for input flag-map pixels which pass the `FLAG_MASKS` masking. There must be one flag value per `FLAG_MASKS` bit set.

4.2.3 Processing of the polygon vector files

`POLY_NAMES` - Filename(s) of the files containing descriptions of polygon shapes, provided as input to WW. Contrarily to all other files handled by WW, polygon data are in `ASCII` format. The adopted syntax is very simple and fully compatible with that of "region" data produced by the DS9[2] image visualization software tools. For single extension images, region files can be written in 'image' or 'fk5' coordinates. MEF images are compatible with the 'image' coordinate only. They are traditionally given the "`.reg`" filename extension. The rules are as follows:

- The string "`fk5`" must be present at the beginning of the file or at the beginning of each line if the polygon file is in WCS coordinates.
- Lines containing the polygon description must contain the expression "`polygon(`x_1`,`y_1`,`x_2`,`y_2`,...)`", where pairs of coordinates x_n,y_n refer to consecutive vertices of the polygon. As WW assumes that the polygon is a closed shape, it is useless to repeat the first point at the end of the list. A polygon must have at least 3 vertices. One `ASCII` line only shall be used per polygon. Overlapping polygons or folded polygon regions are combined following an

[1] According to the FITS convention, integers should be *signed* integers. The behaviour of the sign bit in bitwise operations is notoriously ambiguous. Therefore it is strongly advised to always use positive flag values and to avoid using the sign bit for storing flag information.

[2] `http://hea-www.harvard.edu/RD/ds9/`

"inclusive OR" rule if POLY_INTERSECT is set to Y (the default value) or an 'Exclusive-OR" rule otherwise.

- ASCII lines with a different content will be ignored.

When polygon information has been loaded from one or several files (each file possibly containing a very large number of polygons), WW will use them to define enclosed areas which will be flagged in the output flag-map, and/or set to zero in the output weight-map (for exclusion). A null string ("") can be provided for POLY_NAMES, in which case no polygons will be drawn.

POLY_OUTFLAGS - Flag(s) that will be OR'ed in the output flag-map for pixels "inside" polygons. There must be one value per POLY_NAMES filename.

POLY_OUTWEIGHTS - Weight factor(s) that will be applied to the output weight-map for pixels "inside" polygons. There must be one value per POLY_NAMES filename.

4.2.4 Output files

OUTWEIGHT_NAME - Filename for the weight-map produced by WW. A null string ("") can be provided if no weight-map output is desired.

OUTFLAG_NAME - Filename for the flag-map produced by WW. A null string ("") can be provided if no flag-map output is desired.

4.2.5 Optional parameters

GETAREA - Set to Y to compute effective areas for weighted and (un)flagged pixels. The effective area is written in the flag/weight map header together with the parameters GETAREA_FLAGS (for the flag map) or GETAREA_WEIGHT (for the weight map). The XML output file also contains the effective area.

GETAREA_WEIGHT - Threshold above which output weight pixels are included in the effective area computation.

GETAREA_FLAGS - Bit mask for flagged pixels included in the effective area computation. Alternatively, individual flag values (1,2,4, etc.) can be provided separated with commas.

4.2.6 Miscellaneous

WRITE_XML - If Y an XML file in VOTable format is written at the end of the processing. The default is Y.

XML_NAME - Name of the output XML file.

XSL_URL - URL of the XSLT style-sheet. The XSLT style-sheet is automatically installed with the RPM distribution in /usr/local/share/weightwatcher/ww.xsl.

4.3 Example of configuration

Let's assume that we want to produce a weight-map `weight.fits` which takes into account the gain-map of the detector if the gain is better than 0.2 (from the normalized flat-field `ff.fits`), saturated pixels (pixel values above 65000 in the raw science image `sc.fits`), cosmetic defects (flagged by bit 4 in the flag-map `cd.fits`), cosmic rays (identified by pixel values above 12.0 in the filtered image `cr.fits`) and finally some "bad" regions described as polygonal shapes in `bad.reg`. We also want to produce a flag-map `flag.fits` in which saturated pixel are flagged by bit 0, cosmetic defects by bit 1, gain drops by bit 2, cosmic rays by bit 3, and bad regions by bit 4, and we want to know the number of pixels flagged as 2 OR 1 and the number of those having weights larger than 0.0. The configuration file will be:

```
# Default configuration file for WeightWatcher 1.8.0
# EB 2006-03-01
#

#------------------------------- Weights ----------------------------------

WEIGHT_NAMES ff.fits, sc.fits, cr.fits # filenames of the input WEIGHT maps

WEIGHT_MIN      0.2,  -1e30, -1e30     # pixel values < -1e9 are unlikely!
WEIGHT_MAX      1e30, 65000,  12.0     # pixel values > 1e9 are unlikely!
WEIGHT_OUTFLAGS 0x04,  0x01,  0x08     # hexadecimal coding of bits 2, 0 and 3

#-------------------------------- Flags -----------------------------------

FLAG_NAMES      cd.fits                # filename of the input FLAG map

FLAG_WMASKS     0x10                   # hexadecimal coding of bit 4
FLAG_MASKS      0x10                   # hexadecimal coding of bit 4
FLAG_OUTFLAGS   0x02                   # hexadecimal coding of bit 1

#------------------------------- Polygons ---------------------------------

POLY_NAMES      bad.reg                # name of the file containing polygons

POLY_OUTFLAGS   0xa0                   # hexadecimal coding of bit 4
POLY_OUTWEIGHTS 0.0                    # Weight values for polygon masks
POLY_INTERSECT  Y                      # Use inclusive OR for polygon intersects
#-------------------------------- Output ----------------------------------

OUTWEIGHT_NAME  weight.fits            # output WEIGHT-map filename
OUTFLAG_NAME    flag.fits              # output FLAG-map filename

#---------------------------- Miscellaneous -------------------------------

GETAREA         Y               # Compute area for flags and weights
GETAREA_WEIGHT  0.0             # Weight inferior limit accounted in the area
GETAREA_FLAGS   0x04,0x02       # Bits which will be accounted in the area
```

Acknowledgements Thanks go to Erik Deul for proposing the WeightWatcher name.

References

[1] Fruchter A.S., Hook R.N., http://www.stsci.edu/̃fruchter/dither/dither.html

[2] Bertin E., SExtractor, User's manual, 1996-2003, IAP