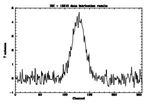
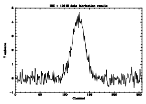


WASP design principles

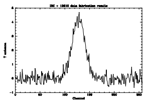
The WASP package has been designed to...

- use modern software engineering practices (e.g., CVS) to create a stable development environment and easily maintained code
- provide a uniform interface to software configuration parameters
- aid the user's management of closely-related tasks
- manage multiple backend instruments at once
- support several data output channels and formats
- make it easy to add support for new types of backends, output file formats, and telescope facilities



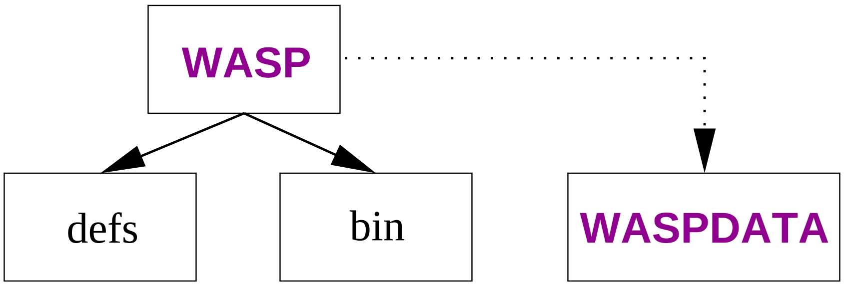
WASP software engineering

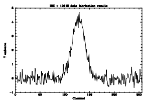
- source code management (`CVS`)
- simplified installation and configuration (`autoconf`)
- use existing libraries (`NEMO`, `cfitsio`, `gpib`)
- automated nightly builds (`testsuite` script)
- inline documentation (`doxygen`):
  HTML, man (LaTeX, RTF, ...)



WASP environment setup

- `source wasp_start` will initialize the WASP variables and add its binary directory to your PATH.
- The WASP and WASPDATA environment variables define the WASP runtime setup.
- WASP executables reside in $WASP/bin and use configuration files in $WASP/defs; output files are placed in $WASPDATA.





WASP configuration parameters

WASP maintains a keyword database to control its runtime behavior. Many parameter settings can also be overridden on the command line (`prog help=h` will display the complete list for `prog`).

Program: `waspset`
Purpose: display/set configuration parameters
Keywords: verbose mend *generic-keywords*

The `waspset` program is used to alter the values of keywords in the persistent (i.e., on-disk) database, or to display the current settings.

Examples:

- `waspset` displays the current values of the most commonly-used keywords
- `waspset verbose=t` displays the current values of *all* keywords
- `waspset exposure=10` sets the default exposure time to 10 seconds

WASP hardware verification

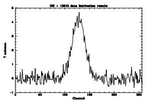
Programs are provided to enable the user to interact directly with backend devices, such as for manual configuration or verification of ready status.

Program: `serial`
Purpose: interact with a WASP2 backend
Keywords: sid timeout cmd out mode

The `serial` program initiates communication (via a serial line) with a WASP2 backend, either interactive or automated.

Examples:

- `serial` begins an interactive session with the microcontroller on sid 0 (COM1)
- `serial sid=1 cmd="v" out=vers.dat` queries the WASP2 software version used by the MC on serial port 1, writes the reply to file `vers.dat`, and exits



WASP hardware verification

Program: `gpib`
Purpose: interact with a GPIB device
Keywords: pad cmd sid nret

The `gpib` program sends a command to a GPIB device, and prints the result.

Examples:

- `gpib` lists all currently supported GPIB devices
- `gpib pad=1 cmd="*IDN?"` identifies the device at GPIB primary address 1
- `gpib pad=4 cmd="*IDN?" sid=1 nret=80` identifies the board on pad 4, connected to serial port 1 via a GPIB-serial converter

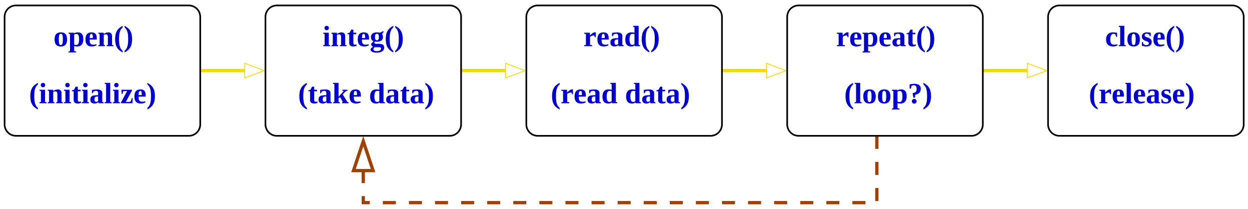
WASP calibration cycle

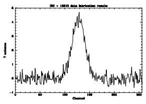
- Hardware calibration in WASP is divided into three phases: finding the zero power offsets, determining the lag to frequency mapping, and calculating the absolute temperature scale.

- Backend hardware is specified through a special set of keywords whose names begin with `be-`. For example:

  ```
  waspset betype0='WASP2 A' beaddr0=0
          beenab0=t benlfn0='none'
  ```

  defines logical backend #0 to be a WASP2 device using cabling convention 'A', connected to serial port 0, enabled (active), with non-linearity corrections turned off.





WASP zero calibration

Program: `zero`

Purpose: find zero power DAC offsets

Keywords: caztime classout diagout fitsfile master nbe

The `zero` program determines the zero-power count rate for all active backends, archives the resulting scans in `fitsfile`, and records the corresponding scan numbers in `czscan#`.

Examples:

- `zero classout=t diagout=f fitsfile=testdata` determines zero offsets for all active back-ends, enables CLASS output files, disables diagnostic output files, and uses `testdata` as the raw data archive

- `zero caztime=10 master=1 nbe=2` performs 10 s zero scans for backends #0 and #1, using the latter to control synchronization

WASP frequency calibration

Program: `cwsigs`
Purpose: Calibration task
Keywords: out freq

The `cwsigs` program is used to measure the ACF response at various frequency settings, and provide a matrix that turns frequencies into lags. Frequencies are often oversampled and binned back to a reasonable number corresponding to the number of lags the spectrometer has.

Example:

- `cwsigs sigsfile=test1.fits freq=100,4800,10` steps in 10 MHz increments from 100 to 4800 MHz and measures the ACF response of the instrument, storing the data in `test1.fits`

WASP frequency calibration

Program: `cwsigsinvert`

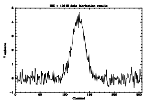
Purpose: Compute combined inversion and binning matrix

Keywords: in svdparm binning *generic-keywords*

The `cwsigsinvert` program is used to compute the inversion matrix, using a Singular Value Decomposition, for converting ACF lags into a spectrum. It can also optionally apply binning to the matrix to simplify the number of computations in the observing pipeline.

Example:

- `cwsigsinvert in=test1.fits cffile=test1m.fits binning=128,1,10,4` computes the inversion matrix from the previous example, but bins into 128 frequency channels, starting at channel 1, stepping every 4 channels, and averaging 10 channels



WASP frequency calibration

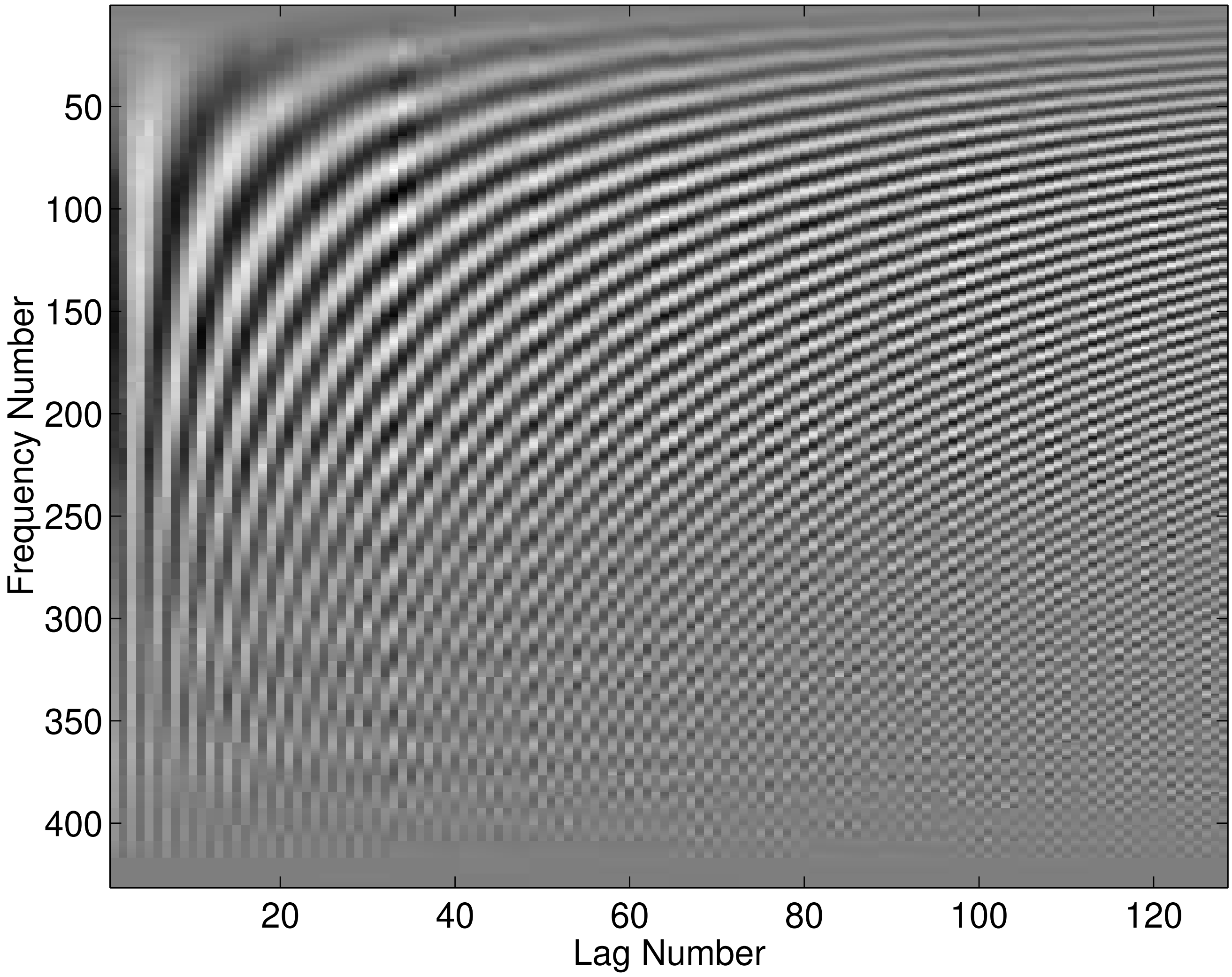
$$As = r$$

or

$$s = A^T r$$

with

$$A_{mn} = U_{mn} W_n V_{nn}^T, \qquad A_{nm}^T = V_{nn} W_n^{-1} U_{nm}^T$$



WASP temperature calibration

Program: `tcal`

Purpose: determine per-channel K/count ratio

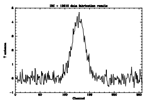
Keywords: cactime cahtime thot tcold defhot defcold

A set of zero (optional), “hot”, and “cold” scans is taken, and used to determine K/count for each frequency channel. The “hot” and “cold” loads can be any combination of hot, cold, ambient, or sky. The resulting temperature calibration vector is saved in the standard data archive and tagged with a scan number (`ctscan#`) for future reference.

Examples:

- `tcal thot=400 tcold=77` will calibrate temperatures using (physical) hot and cold loads with temperatures of 400 K and 77 K, respectively

- `tcal thot=300 tcold=-1 defcold='ambient'` will calibrate temperatures using an ambient load as the “cold”; the ambient temperature is set automatically by the facility



WASP observing

Program: `totpwr`
Purpose: perform total-power source integration
Keywords: exposure

Perform a total power observation; the resulting data is output as an unscaled spectrum. Used implicitly during the calibration cycle.

Example:

- `totpwr exposure=5` integrates directly on-source for 5 s, writing the data to the usual channels

WASP observing

Program: `nobs`

Purpose: perform chopped/nodded integration

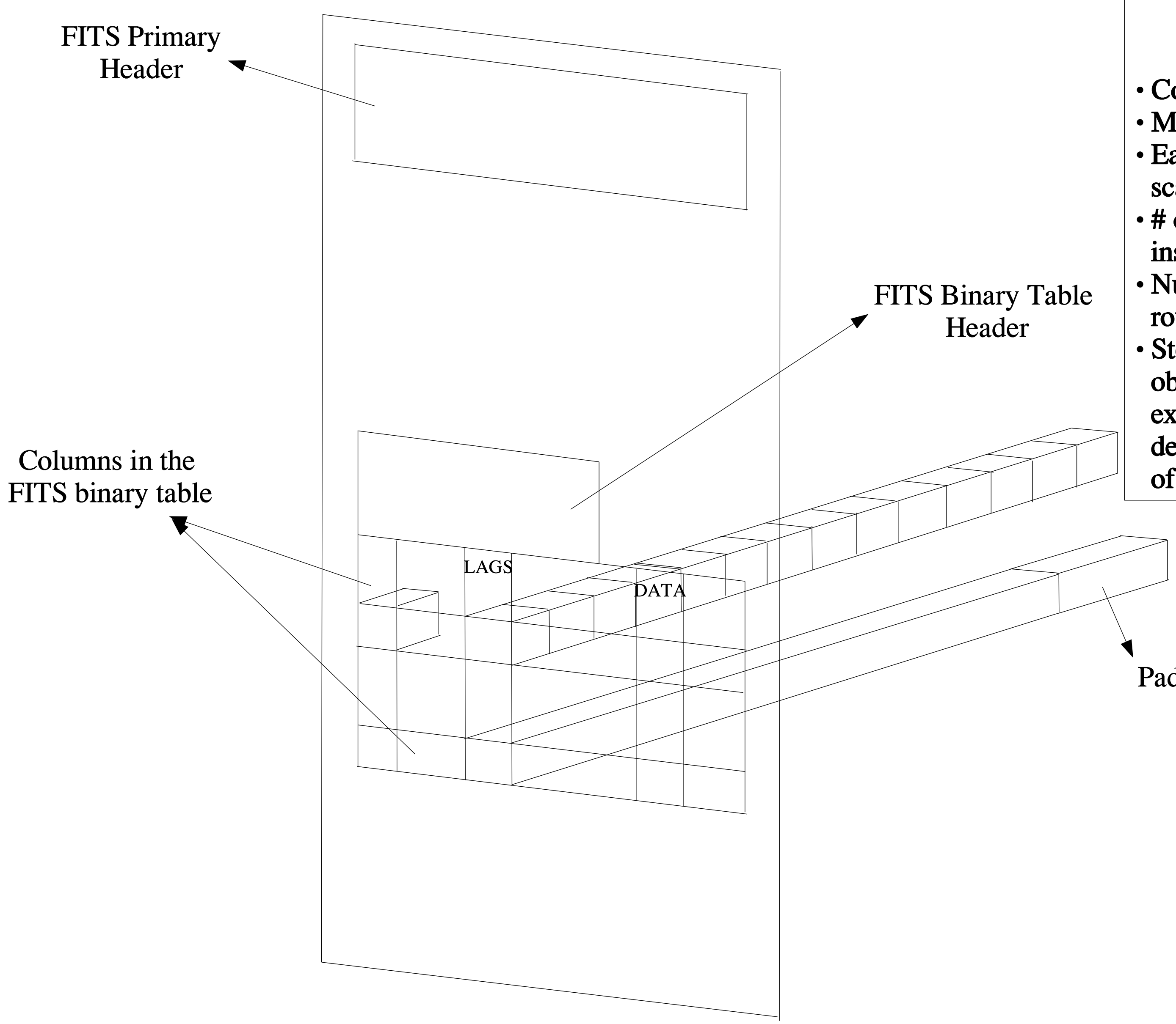
Keywords: exposure chopfreq chopwait
nodtime nodwait

Observes a source using chopping and nodding; the resulting data is output as a calibrated spectrum. This is the program used for “normal” observing.

Example:

- `nobs exposure=100 nodtime=10 nodwait=2 chopfreq=0.5 chopwait=0.1` observes a source using chopping and nodding with a total on-sky integration time of 100 s; the number of chop and nod cycles is determined automatically from the supplied parameters

SDFITS Archive File



Features

- Conforms to FITS standard.
- Multiple binary tables.
- Each table stores multiple scans.
- # of scans in a table = # of instruments (data sources).
- Number of data points in each row can differ for vector data.
- Stores information about each observation as well, for example, elevation, nstrument details, longitude and latitude of observation.

Class FITS File

FITS Primary Header

SIMPLE = T
BITPIX = 16
NAXIS=4
NAXIS1 = 1024
NAXIS2 = 1
NAXIS3 = 1
NAXIS4 = 1
EXTEND = T
BSCALE =
BZERO =
DATAMIN =
DATAMAX =
...

REFRAC =
SITEELEV =
SITELAT =
SITELONG =
TAMBIENT =
TAUIMAGE =
TAUZENIT =
TCAL =
TEMPSCAL =
TIMESYS =
TRANSITI =
TEX =
...

FITS Primary Array – contains the observation data

Features

- Conforms to FITS standard.
- Each file stores data from one scan
- Stores information about each observation as well, for example, elevation, instrument details, longitude and latitude of observation.