Flair Advanced Features

Advanced FLUKA Course

About

/fleə(r)/ n [U,C] natural or instinctive ability (to do something well, to select or recognize what is best, more useful, etc.

[*Oxford Advanced Dictionary of Current English*]

What is flair [1/2]

FLUKA Advanced Interface [http://www.fluka.org/flair]

- **All-in-one** User friendly graphical Interface;
- Minimum requirements on additional software;
- Working in an intermediate level
  Not hiding the inner functionality of FLUKA

Front-End interface:

- Fully featured **Input file Editor**
  - Mini-dialogs for each card, allows easy and almost error free editing
  - Uniform treatment of all FLUKA cards
  - Card grouping in categories and card filtering
  - Error checking and validation of the input file during editing
- **Geometry:** interactive visualization editing, transformation, optimizations and debugging (tomorrows talk);
- **Compilation** of the FLUKA Executable;
- **Running** and **monitoring** of the status of a/many run(s)

What is flair [2/2]

Back-End interface:

- Inspection of the output files (core dumps and directories)
- Output file(s) viewer dividing into sections
- Post processing (merging) the output data files
- Plot generation through an interface with gnuplot;

Other Goodies:

- Access to FLUKA manual as hyper text
- Checking for release updates of FLUKA and flair
- Nuclear wallet cards
- Library of materials
- DICOM images conversion to VOXEL geometries
- PET automatic geometry creator
- Programming python **API**
- Everything is accessible with keyboard shortcuts

Concepts: Flair Project

- Store in a single text file all relevant information:
  - Project notes
  - Links to needed files: input file, source routines, output files …
  - Multiple runs from the same input file, as well running status
  - Procedures on how to run the code
  - Rules on how to perform data merging
  - Information on how to post process and create plots of the results
- You can consider Flair as an editor for the project files.
- Can handle any FLUKA input format (reading & writing), but internally it works using the names format for the input, free with names for the geometry (Recommended way of working)
- The format is plain ASCII file with extension: .flair

Note: If you want to copy a project you need to copy also all linked files especially the input and source routines!

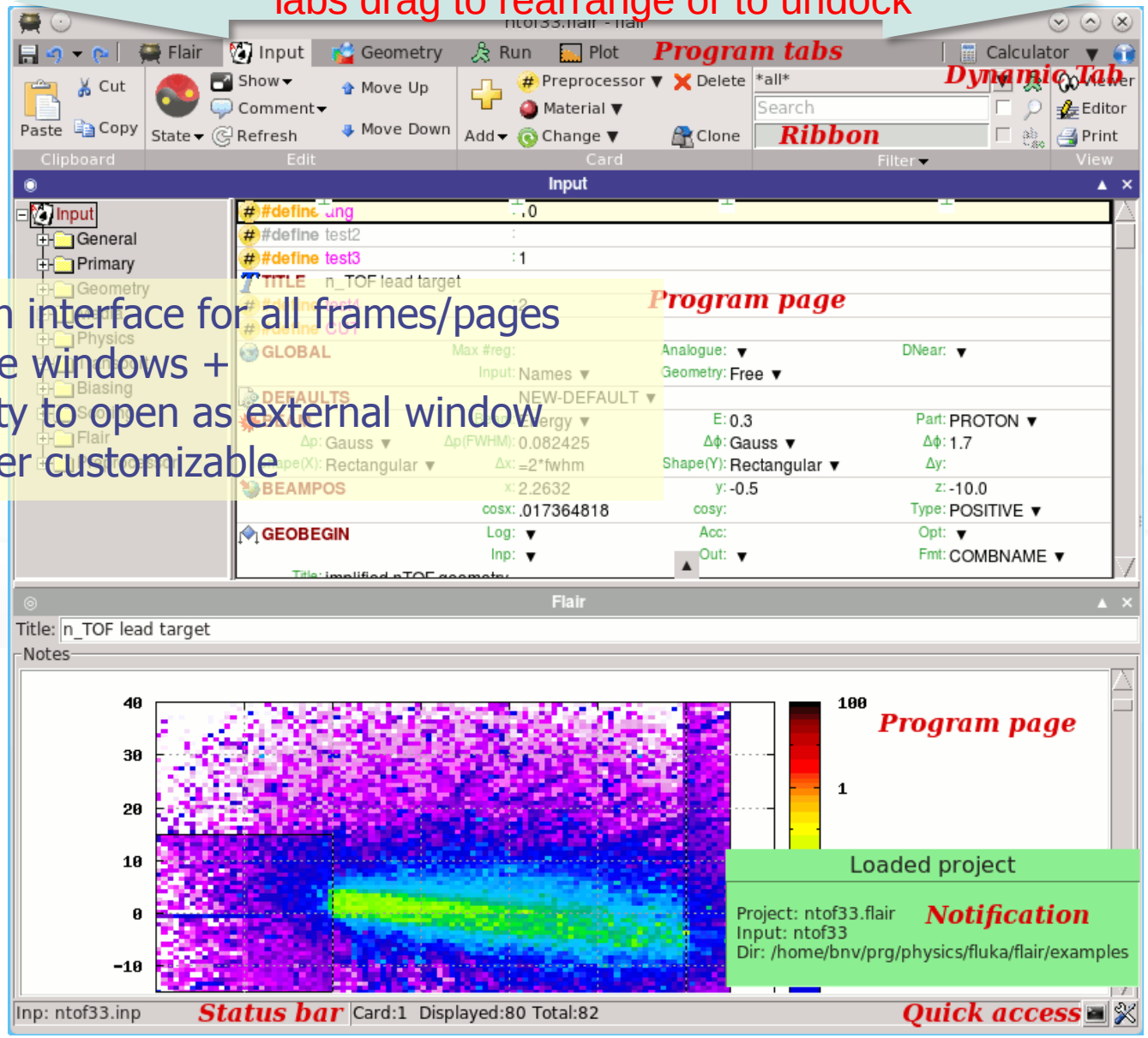
Command line options

Usage: flair [options] <filename | filename.flair | filename.inp>

Options:

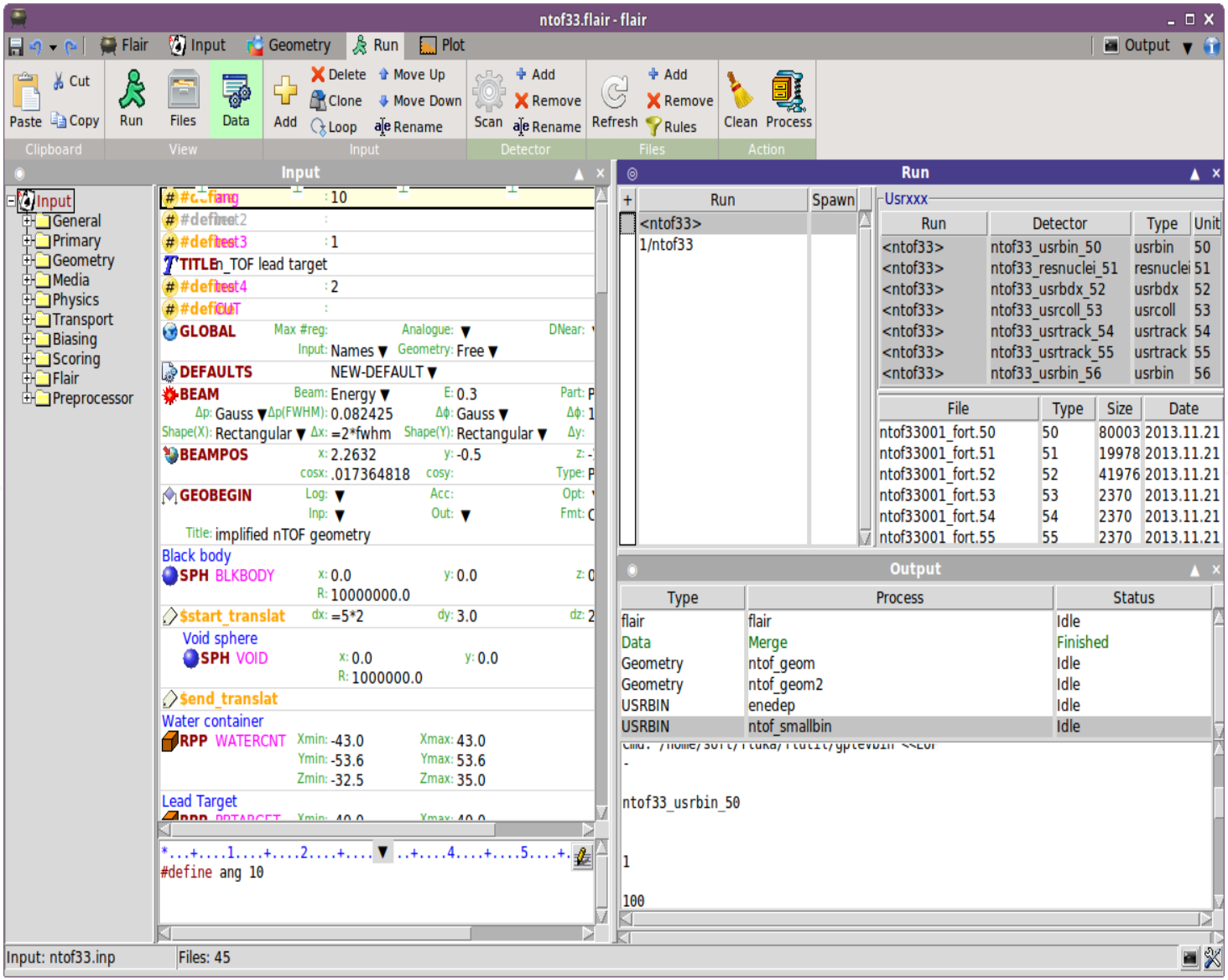
- -d/D Activate/Deactivate the beta-development features
- -e exe Use exe as fluka executable
- -i inputfile Fluka input file (w/o the .inp extension)
- **-r** Load most recent project
- -R # Load recent project (number 1..10 or filename)
- -l List recent projects
- -1 Load the first flair file in the folder

Interface



- Common interface for all frames/pages
- Dockable windows + Possibility to open as external window
- Fully User customizable

Interface - Multi docking



Interface

Keyboard:

Almost everything is possible with the keyboard see manual for shortcuts

Ctrl-Enter: Execute most important action

Ins/Del: Add or Delete

Mouse:

right-click anywhere to get a popup menu

Listboxes:

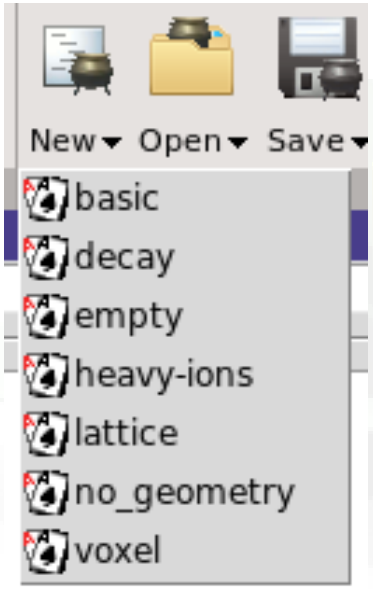
all listboxes are searchable. Typing only the characters (A-Z) and numbers (0-9) all other are ignored

LabelFrames:

can be collapsed/expanded by clicking on the label

Input Templates

- When requesting a new input or a new project flair will prompt to select an input template:



Default template: basic.inp

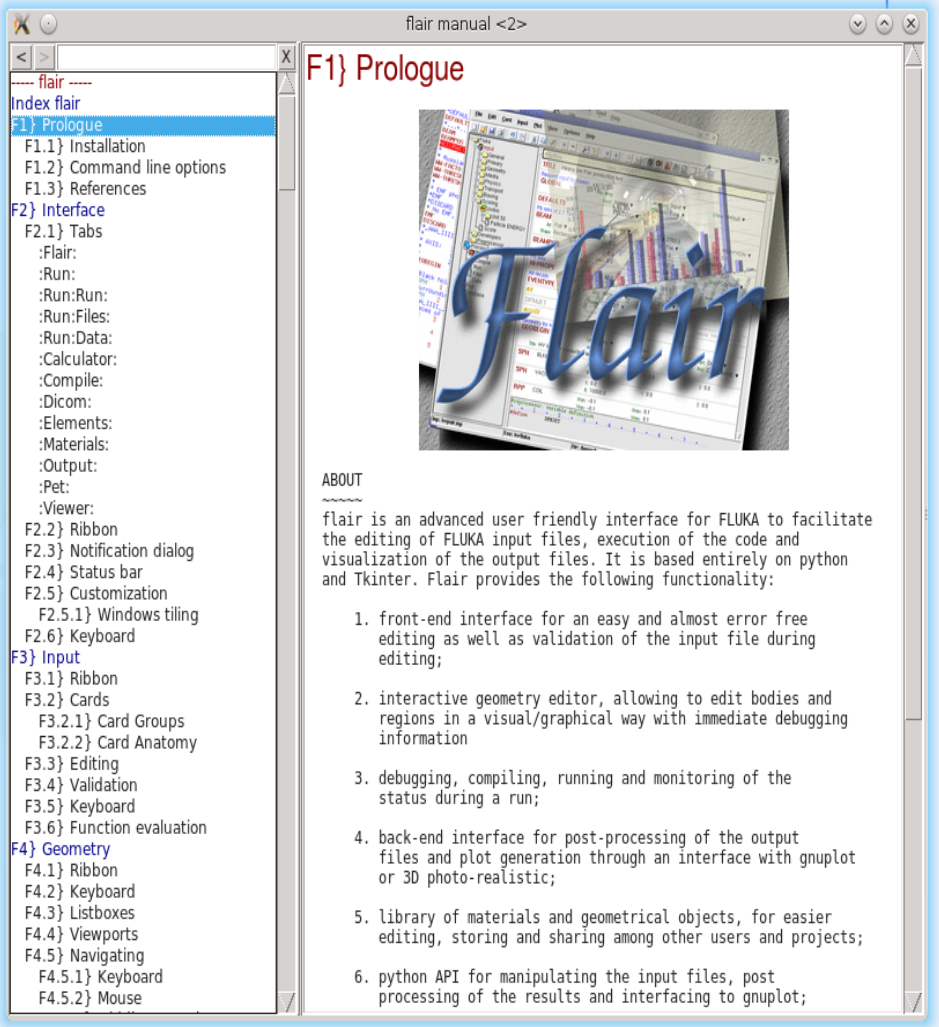
```
TITLE

GLOBAL                                              1.0       1.0
DEFAULTS                                                          NEW-DEFA
BEAM
BEAMPOS
GEOBEGIN                                                          COMBNAME
    0    0
* Black body
SPH blkbody    0.0 0.0 0.0 10000000.0
* Void sphere
SPH void       0.0 0.0 0.0 1000000.0
* Cylindrical target
RCC target     0.0 0.0 0.0 0.0 0.0 10.0 5.0
END
* Black hole
BLKBODY      5  +blkbody -void
* Void around
VOID         5  +void -target
* Target
TARGET       5  +target
END
GEOEND
* ..+....1....+....2....+....3....+....4....+....5....+....6....+....7..
ASSIGNMA    BLCKHOLE    BLKBODY
ASSIGNMA      VACUUM       VOID
ASSIGNMA      COPPER     TARGET
RANDOMIZ         1.0
START
STOP
```

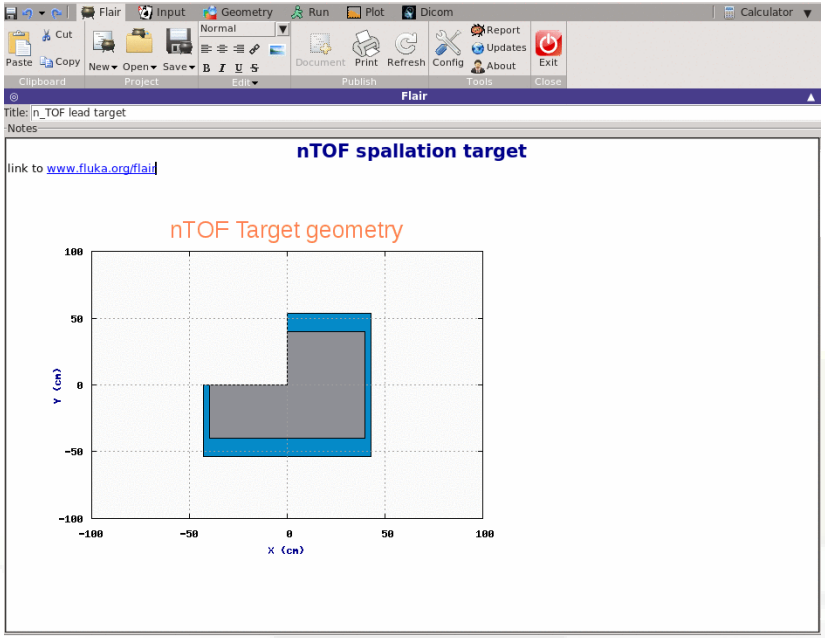
The user can create his own set of input templates. They are normal FLUKA input files and they have to be placed in the directory ~/.flair/templates (create the directory if not existing)

Help

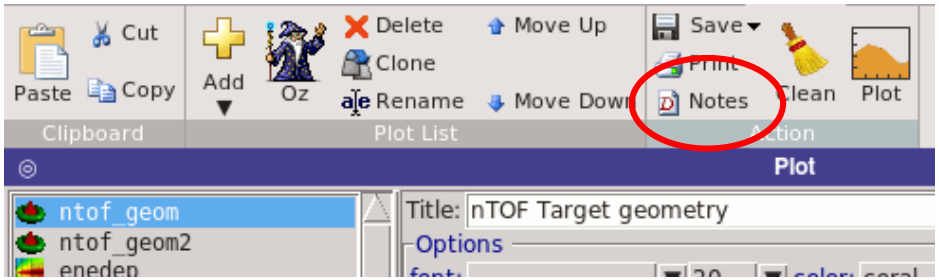
- [F1] or the help icon
- Displays both the flair and the installed fluka manual
- Searchable:
  - Filter pages only with specific keywords (keeps history of keywords)
  - Type searching in the listbox
  - Ctrl-F to search for a keyword inside the displayed page

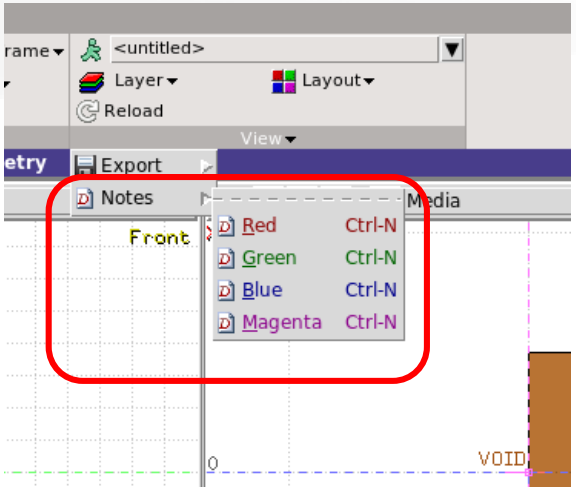


Page: Flair



- Notes, like a simple word processor. Accepts basic formatting, images and URLs to external documents.
- Any plot or geometry viewport can be inserted as an image inside the notes

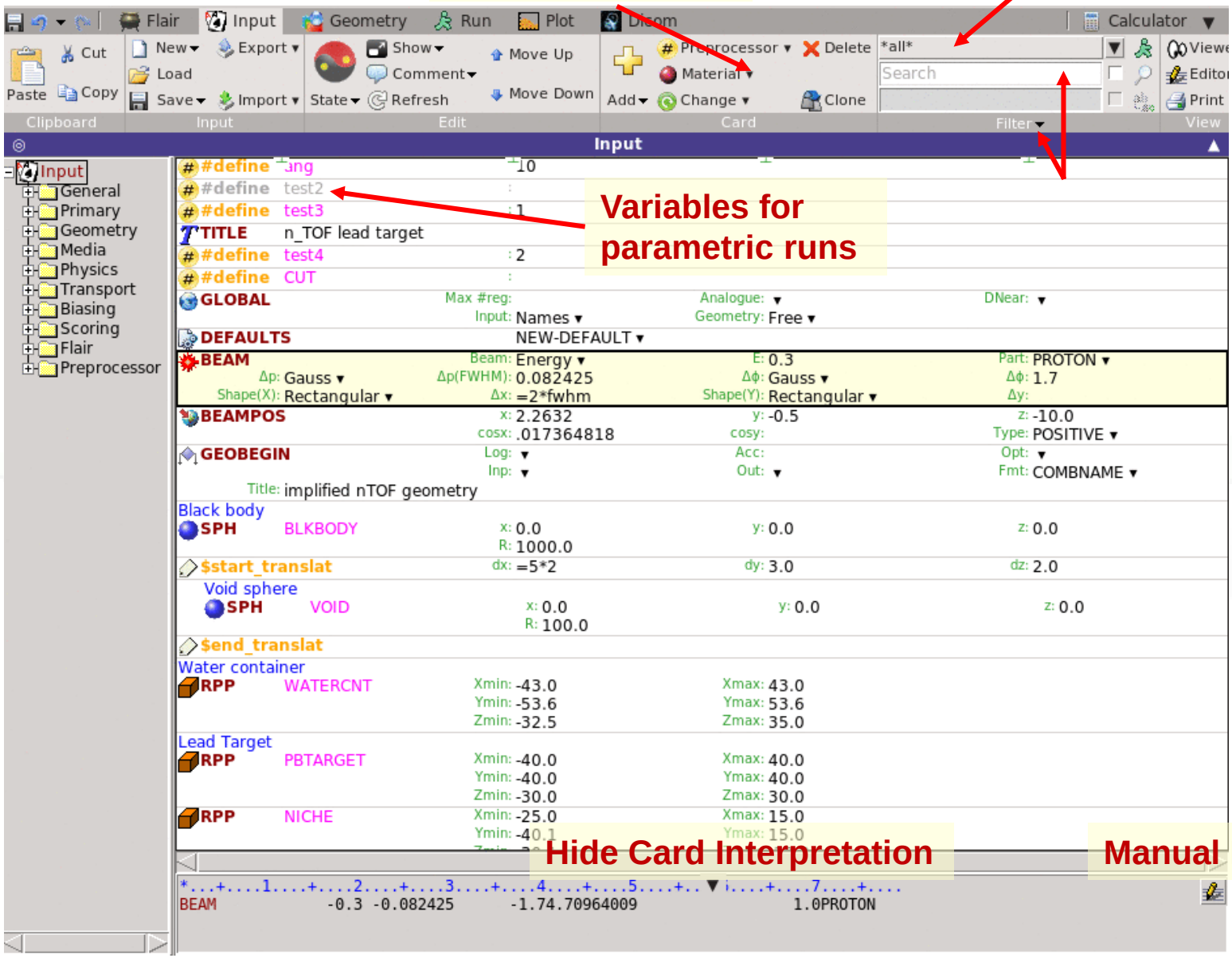




Input

Display Active Cards of A Run Configuration

Material Database



Variables for parametric runs

Hide Card Interpretation

Manual Editing

Input Editor – 2: Import / Export

Importing

- Input: merge parts or entire input file with the current
- Mcnp: import mcnp geometry into FLUKA. (experimental)
- Gdml: import gdml geometries into FLUKA. Use of parenthesis on Macro bodies (e.g. polycone, xrtu…) which can lead to complicated geometry for FLUKA to handle. (experimental)

Exporting

- Gnuplot: save active plot to a gnuplot script
- Makefile: create a makefile for compiling the executable
- Mcnp: save input in MCNP format: Geometry, Materials, Importances
- Povray: save geometry into povray 3D format

Input Editor - 3

- Drag'n'drop from the TAG of the cards
- Double click on card TAG to select all similar cards
- Editing multiple cards: select cards and modifying the value in one card will propagate the change to all similar selected cards
- Ctrl-Double-Click Show/Hide selected cards
- #if..#endif, $transform, $translat or $expand flair will enclose the selected cards with the #if #endif, or $start_xxx, $end_xxx transformation cards
- Popup Balloon tooltip displays short help:
  - for every option on every card
  - body description in the REGION expression
- Right-click: shows popup-menu
  - Quick filtering by REGION, MATERIAL, scoring etc…
- Easter Eggs: AWARI by Double-Right-Click on dialog showing the card representation as text at the bottom of the screen

Input Editor – 4

- Automatic indentation of nested #if..#endif and $start..$end directives.
- To refresh the display press Ctrl-R

- Each REGION can be split into many cards if needed to be used with preprocessor commands.
- Use as a name "&"

```
Void around
REGION VOID                                   Neigh: 5          Volume:
     Expr: +void -target
#if        BIAS ▼
   REGION &                  cont: -bias
#endif
```

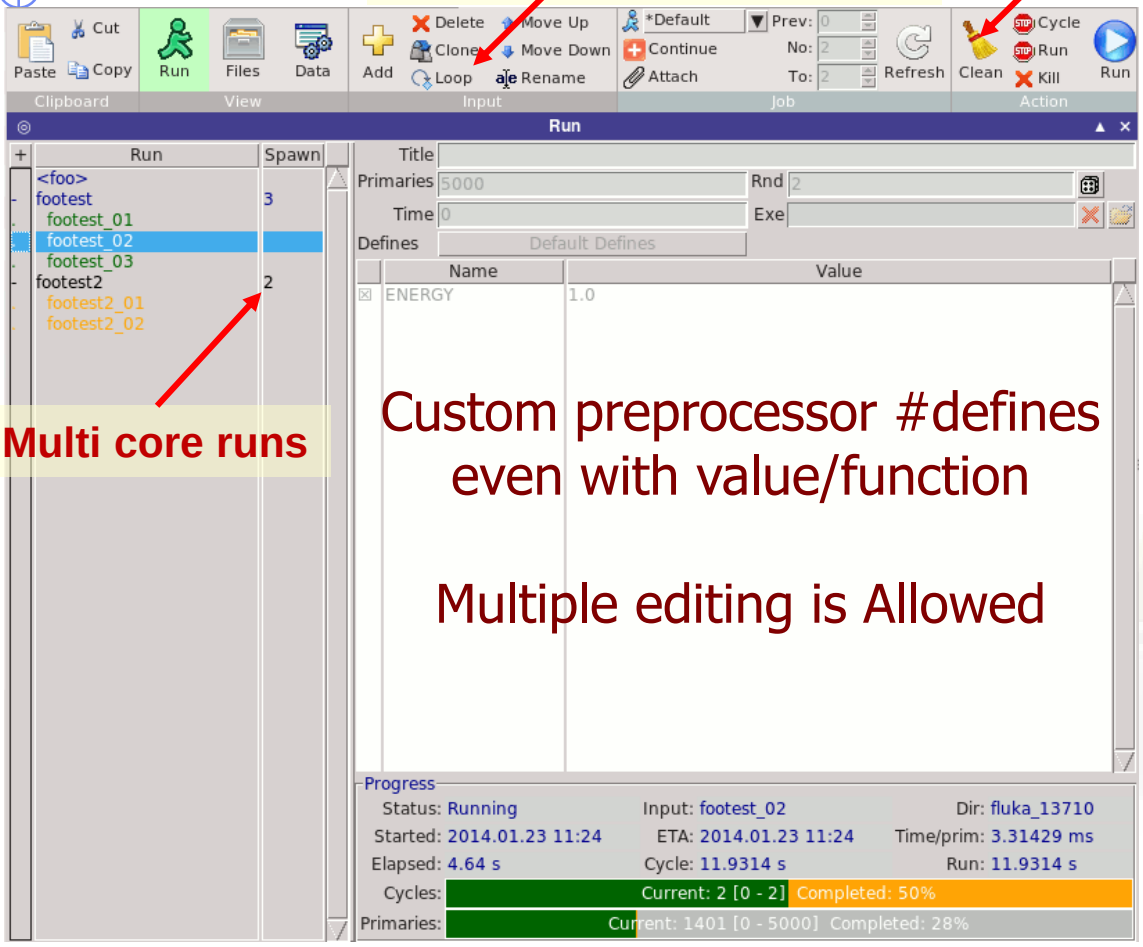
Input Editor – 5: Function Evaluation

- Using functions (starting with the **=** "equal" symbol like in Excel) as arguments to the cards can make the input easier (or more complex) to read and manipulate
- Check manual "F3.6 Function Evaluation" for all possible functions/constants
- Examples:
  - Units: =10*MeV, =5*inch, =10*um, =2*hour, =5*cm*fwhm
  - Physics: =p2E(10*MeV, Mp), =X0(Z,A), dE2dp(E,dE,m)
  - Card reference: =w(1)+5.0*cm, =b(TARGET,2), =C(BEAM,1,1)
  - Vector operations: =cross({1.,0.,.0}, {0.,1.,0.})
  - All mathematical functions are accepted

WARNING:

1. Functions are evaluated by flair and a FLUKA compatible input will be written.
2. If you modify with an external editor the input you need to reload it in flair make a change and save it.
3. The function evaluation is different than the $variable from FLUKA
4. Python evaluation: 1/2=0 1.0/2 = 0.5

Running



<inputname> refers to the input file AS IT IS in the input editor.

Create additional runs based on the same input file by overriding:

- Title
- Preprocessor definitions
- Random number seed
- Starting particles
- Execution timeout
- Executable

- Monitors the status of the run by inspecting the FLUKA output files. If timeout occurs try to re-Attach to the running process.
- The timeout is user-definable in the Preferences dialog

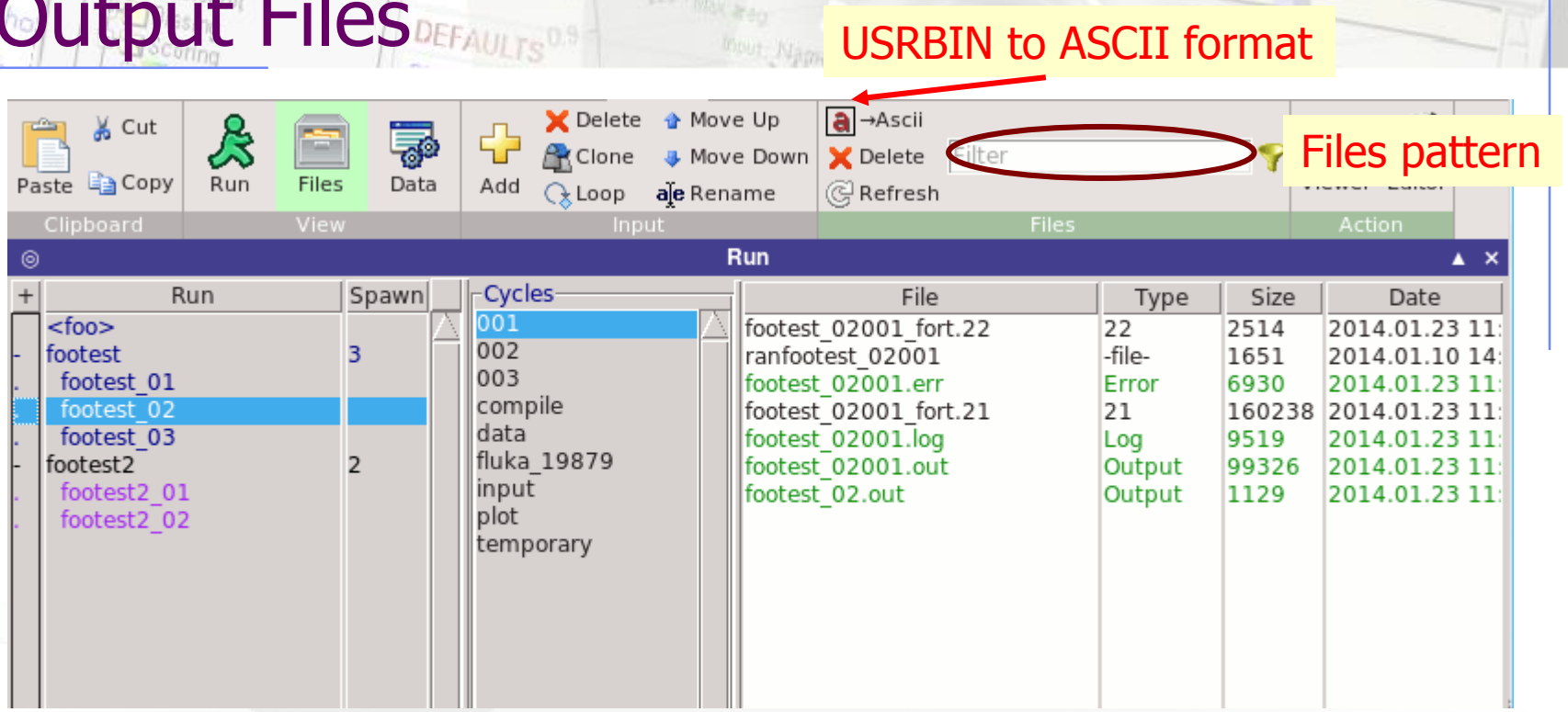
Running: How to use multicore CPU's

- Create clones of the current input e.g. test.inp named: test1.inp, test2.inp, test3.inp …
- Assign a different random number seed on each run (Rnd entry). Incremental from the parent run.
- Select all in the listbox and click Run
- Check "Preferences" for changing the naming format

Multiple Selection:

- To modify many runs at the same time, select them in the listbox
- The options will be "*disabled*"
- Right-click on the options you want to enable and modify them
- Modify the filters in Data processing for summing up all cycles from all runs (see later)

Output Files



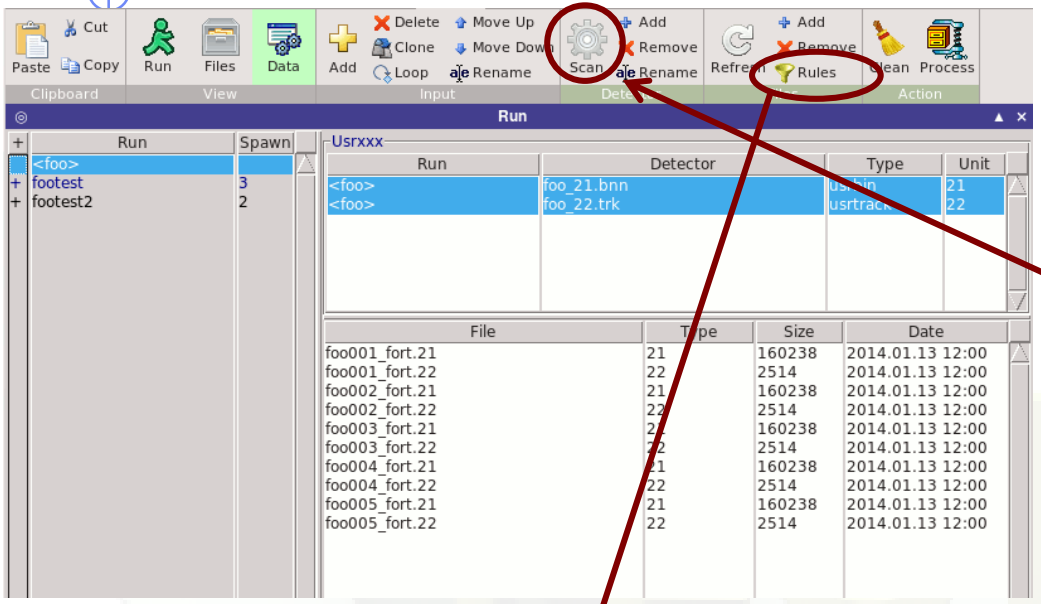
Inspect Output files generated by FLUKA classified per: Run/Cycle

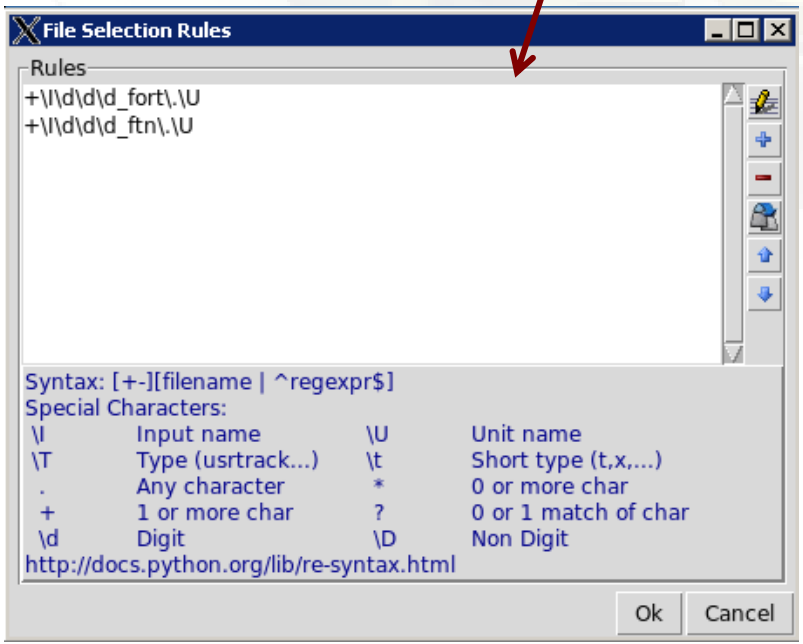
As well special output files from compilation data processing plotting and temporary

Double clicking opens:

- Files in the file Viewer
- coredumps in debugger

Data Processing

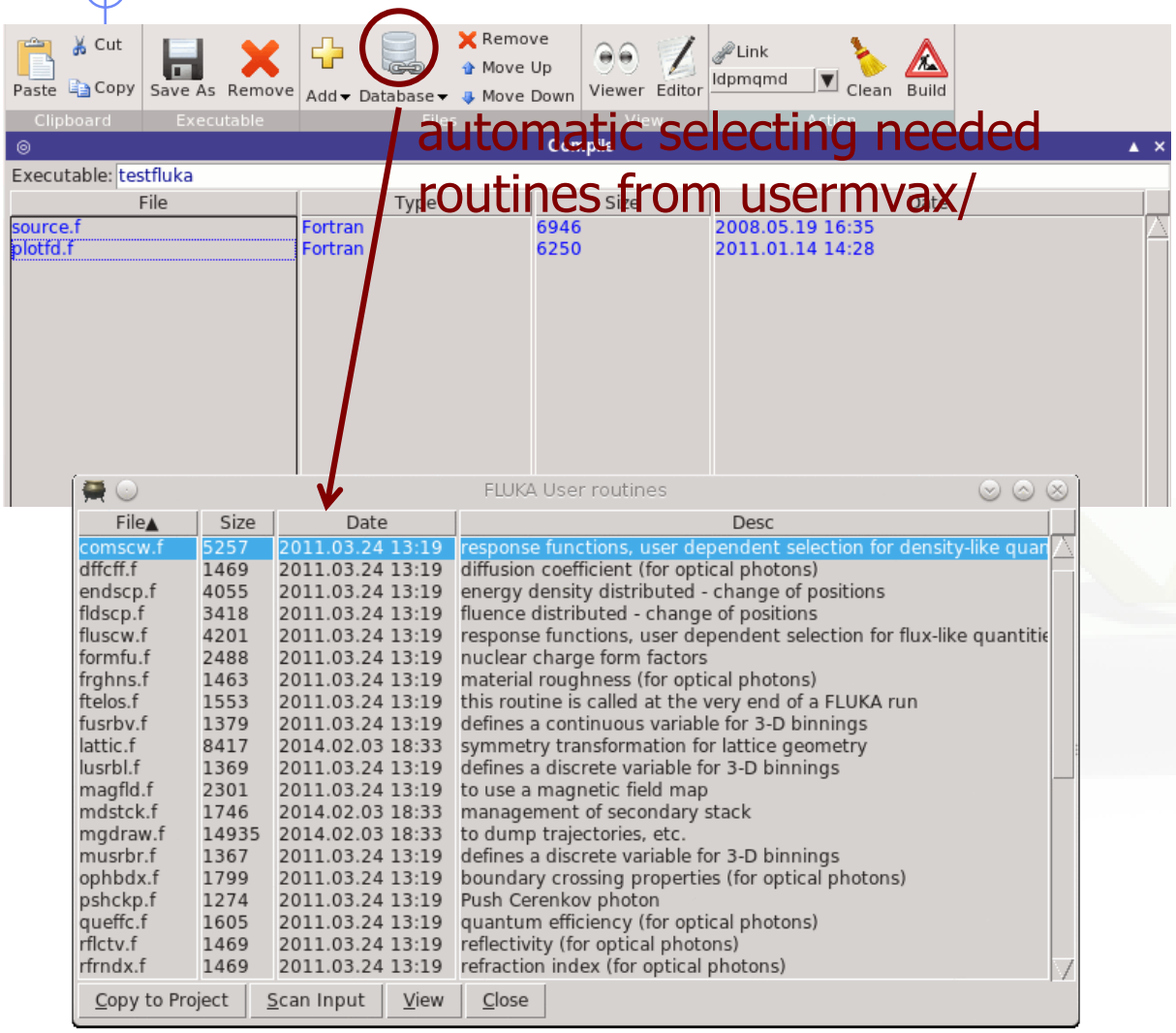




- Process all scoring BINARY output files for each Run.
- Name rules are defined in Preferences
- Automatically scan input for scoring cards
- +/- Modify file list by adding / removing items
- Dialog for editing scanning rules for files.

- To modify the rules for multiple scoring cards, select all Usrxxx before
- The default rules can be modified in the Preferences Dialog

Compiling



Filetypes accepted:

- Fortran: .f, .F, .for, .FOR
- C/C++: .c, cpp, .cxx, .cc
- Libraries: .a, .so

Automatic scanning of necessary user routines and copying them to project folder.

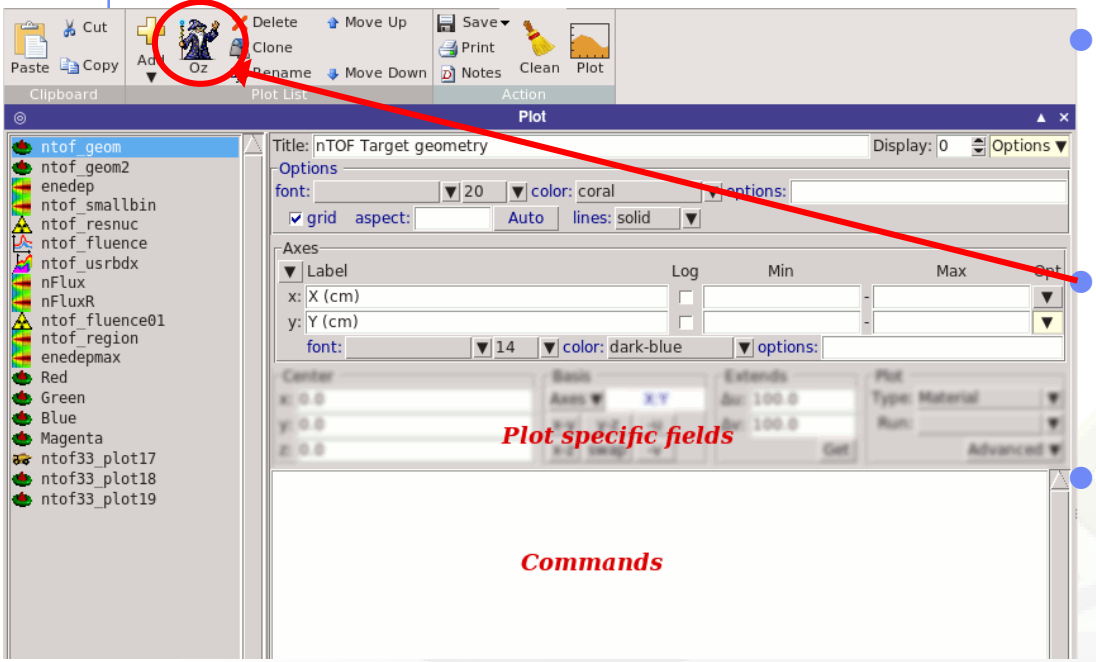
Build: behaves like a "makefile" compiles based on files timestamp when are newer

Compile: Forces compile of the selected files

Clean: cleanup of all produced files

When you are unsure, click on "Clean" before "Build"

Plots



- It is important to set a **unique** filename for each plot. This filename will be used for every auxiliary file that the plot needs (the extension will change)
- The Wizard button creates automatically one plot for each processed unit (From the default input file)
- Multiple plots can be selected and modified. All changes on enabled fields will be propagated on selected plots
- Right-click on disabled items to enable/disable the field during multiple editing
- All plots share the same header and footer
- Commands: allows to enter gnuplot commands for further customization

Plot Types

- Geometry — For geometry plots
- USRBIN — For plotting the output of USRBIN
- USR-1D — To plot single differential quantities from cards USRBDX, USRTRACK, USRCOLL, USRYIELD
- USR-2D — To plot double differential from USRBDX
- RESNUCLE — To plot 1d or 2d distributions of RESNUCLEi
- USERDUMP — To plot the output of USERDUMP. Useful for visualizing the source distribution (ToDo)

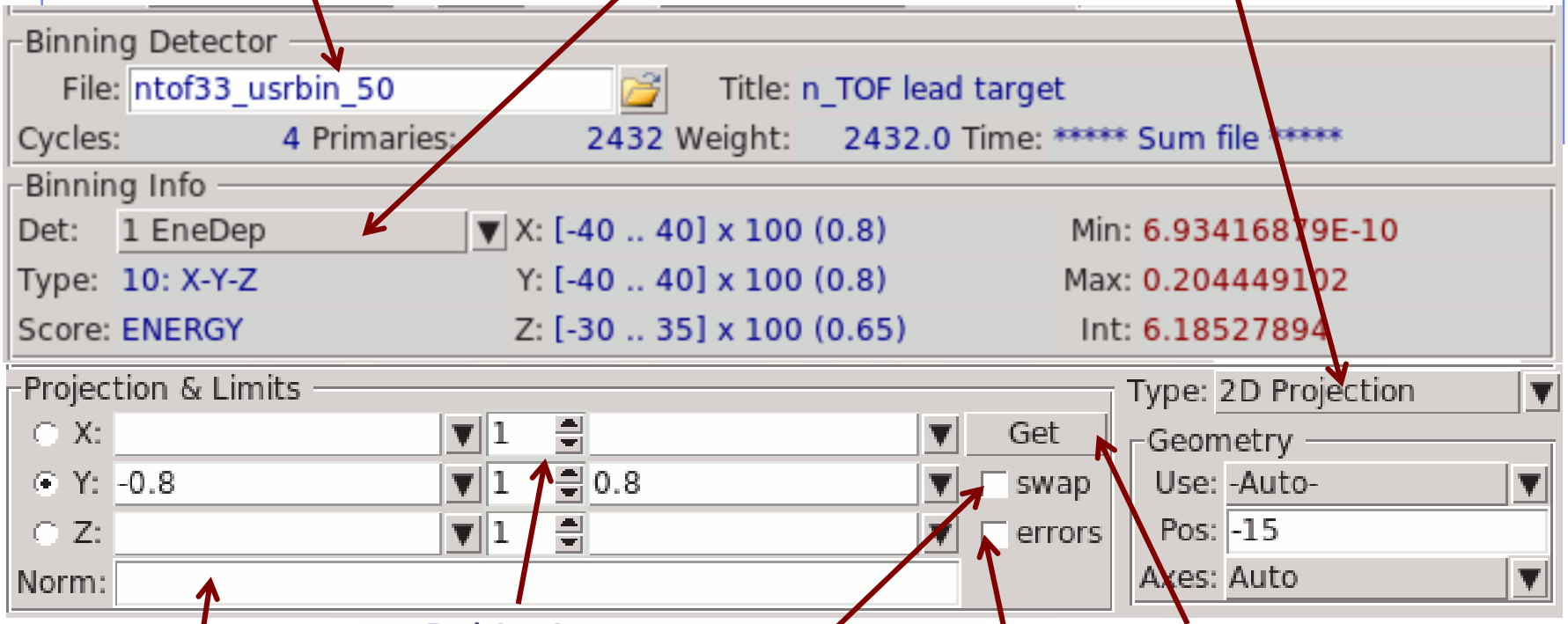
General Tips

- In the Configuration Dialog you can set global commands to execute before or after any plot
- The output page displays all the commands that are sent to gnuplot. As well as the errors. In case of problem always consult the output window!
- Multiple displays can be used to compare plots
- Advanced options are displayed by clicking the "**Options**"/"**Advanced**" buttons
- Additional axes can be found in the "Axes" button
- In the Gnuplot commands you can fully customize the plot by adding manually gnuplot commands:
- Special commands:
  - plot, splot — with no options, defines the order where flair should insert the plot or splot command.
  - replot <plot-cmd> — append extra plots to the one generated by flair

USRBIN Plots

Normalization could be plotted:

- 2D projection, 1D projection
- Trace of the maximum
- Full width at half maximum

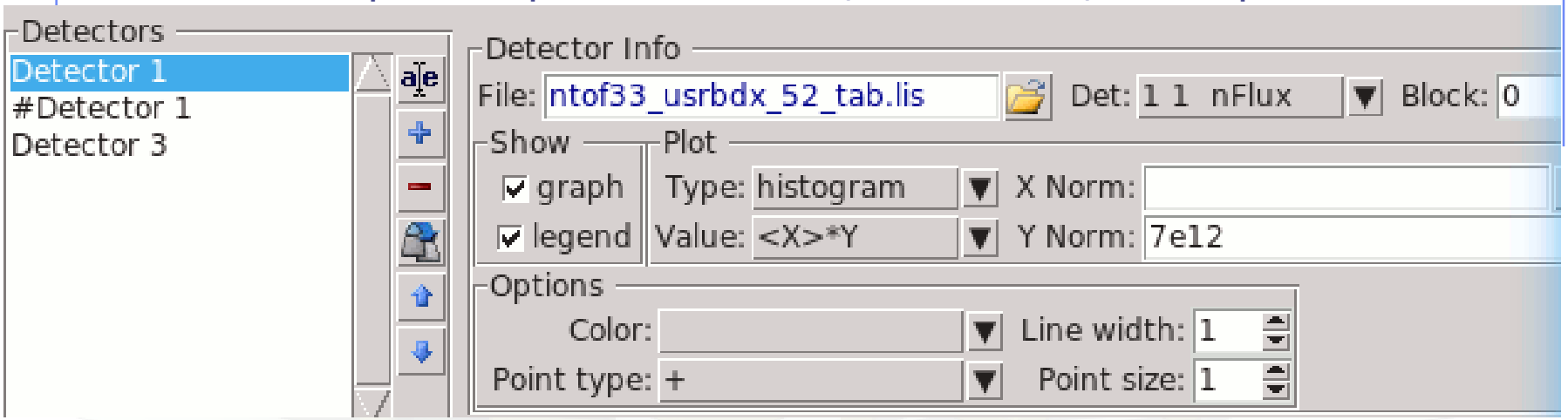


Normalization could be used as:
number or expression evaluating in a number 65e-3/2.7
function with x as variable. e.g E2T(x*65e-3/2.7)-293
with the function defined in the Gnuplot commands
E2T(x) = ((3.00629e-08*x-0.000108436)*x+1.01097)*x+311.839

Draw errors. (combined with log)
Correct only if one slice is used

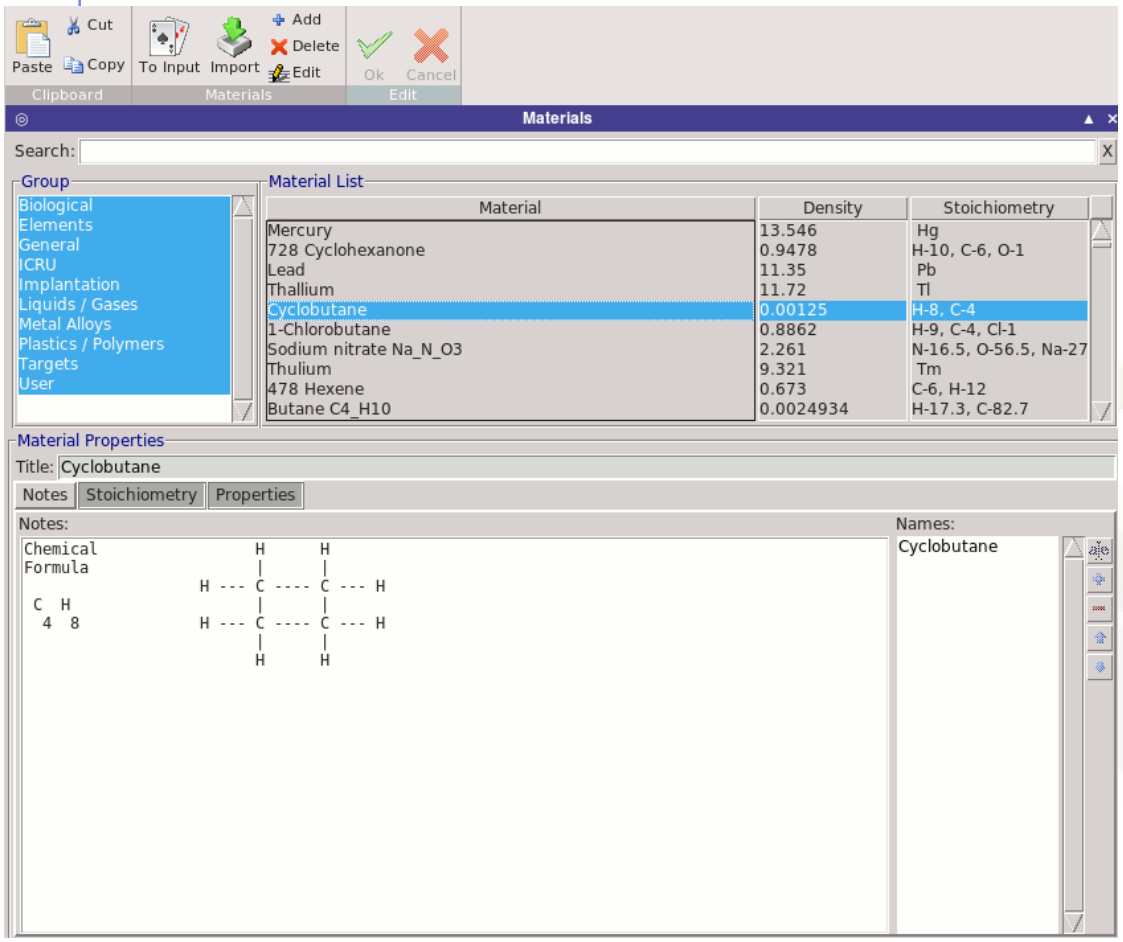
USR-1D

- Multiple 1D plots can be displayed: USRBDX, USRTRACK, USRYIELD as well the "plot" output from USRBIN, RESNUCLEI,… or experimental data



- Detectors: listbox to add graphs
- File: detector file name (_tab.lis or .dat)
- Det: Detector name or index
- Block: multi-dimensional information, splitted into blocks
- Type: drawing type (histogram, histerrors, lines,…)
- Value: Y, <X>*Y, Dx*Y
- X Norm: to change units (1/eV,…)
- Y Norm: normalization function like in USRBIN

Materials Database

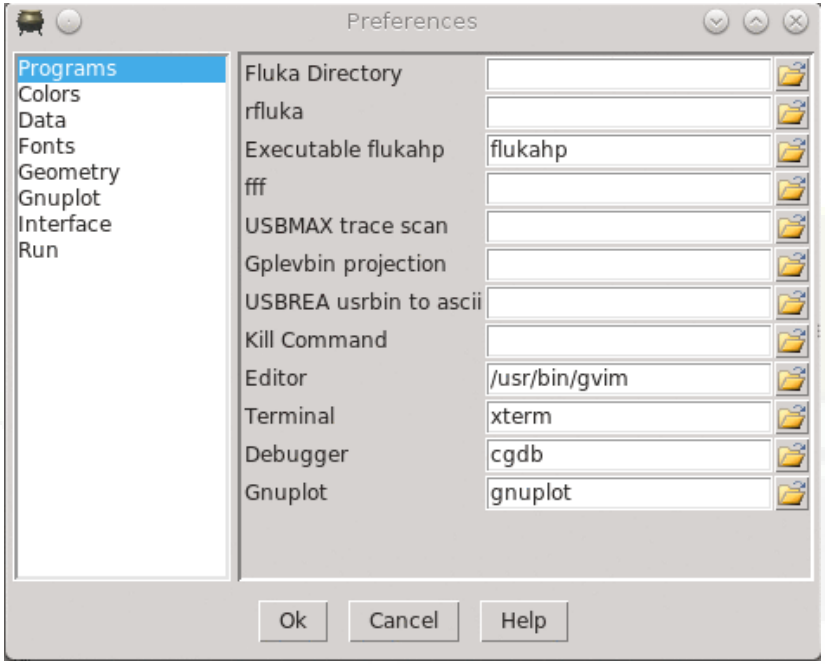


Edit the material database

- search database
- insert material to input add/del material edit material
- Grab materials from input
- Modify Stoichiometry and properties of material

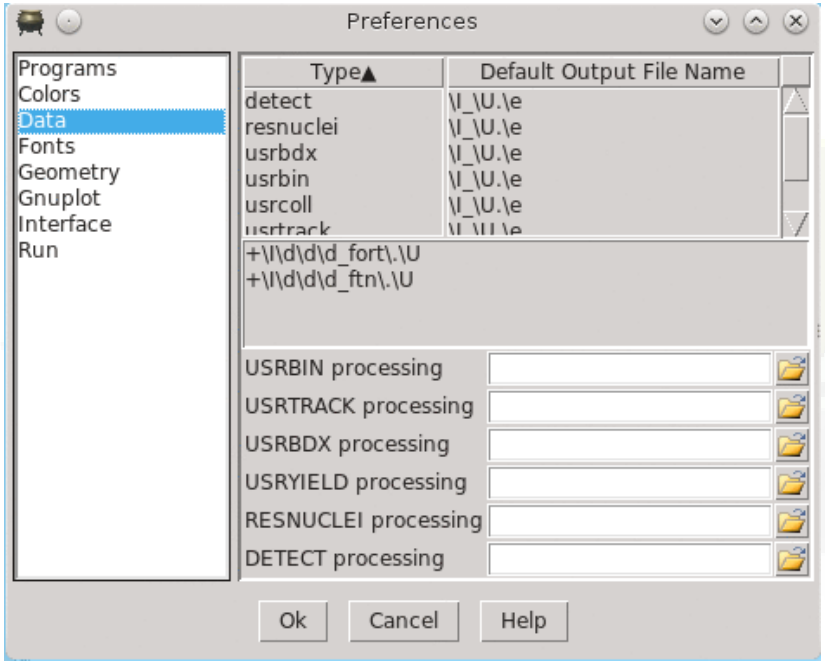
WARNING: When modifying the database a local copy will be created in ~/.flair folder, with the user changes

Configuration Dialog: Programs



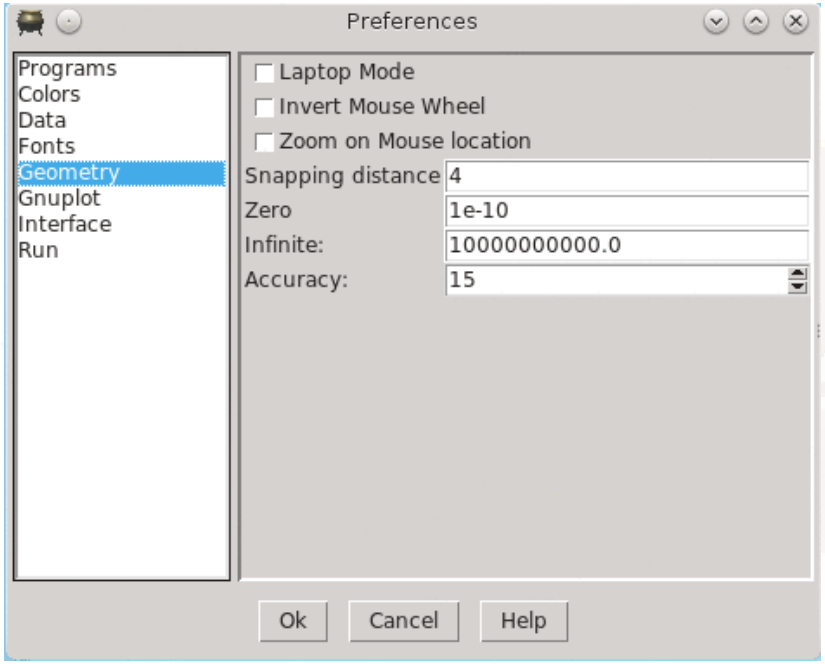
- Set FLUKA directory
- Override default programs to use
- Processing programs are in the “Data” section

Configuration Dialog: Data



- Define how to generate the automatic filenames
  - \I will be replaced by input
  - \T by card name
  - \t by card character
    - usrbdx x
    - usrbin b
    - usrcoll c
    - usrtrack t
    - usryield y
    - resnuclei r
  - \U the abs(unit-number)

Configuration Dialog: Geometry



Laptop Mode:

- check to swap middle with right mouse buttons. Middle button is used in GeometryEditor for panning, zooming, rotating etc...

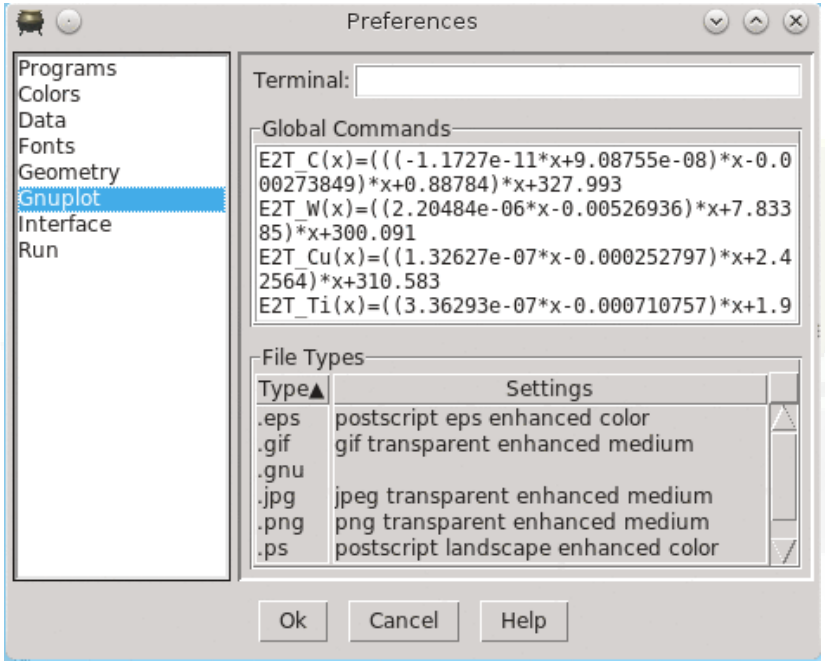
Zero:

Infinite:

Accuracy:

- same as in the Bodies Transformation dialog

Configuration Dialog: Gnuplot



Terminal:

additional options to supply to default terminal

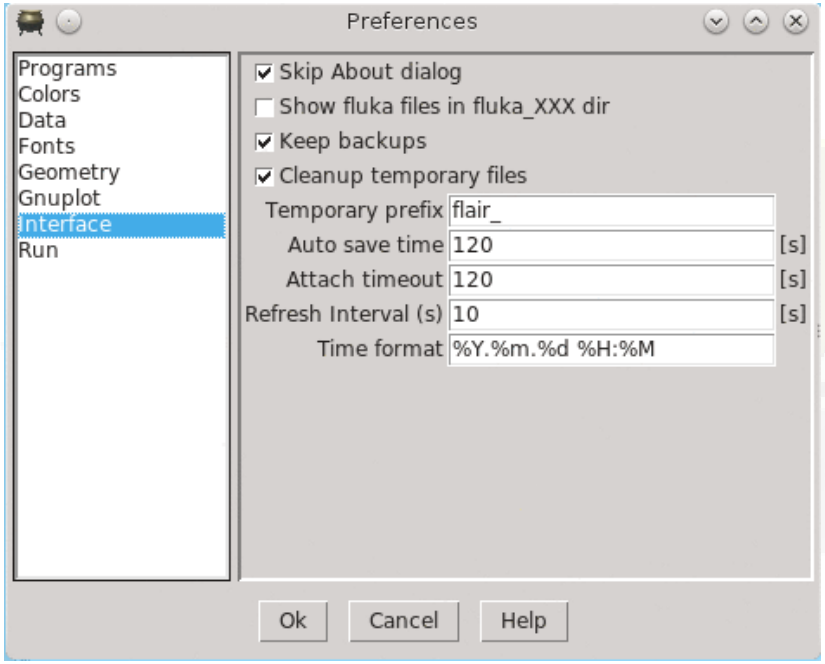
Global Commands:

gnuplot commands to be executed before any plot

File Types:

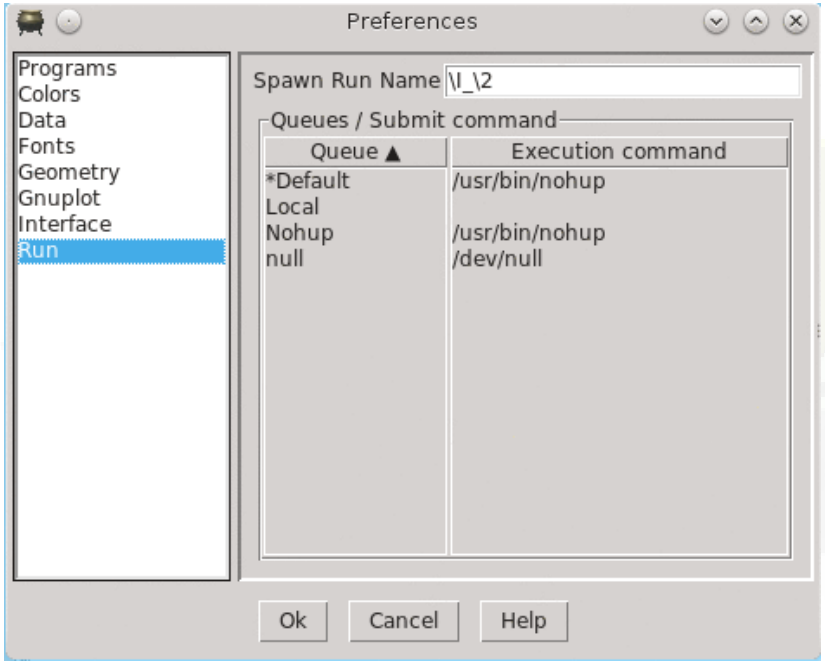
Right-click: to Add/Delete/Modify file types.

Configuration Dialog: Interface



- General interface settings
- Keep backups when files are saved as (file~)
- Automatically Cleanup temporary files. Disable only if you want to inspect files after Debug or Plot when an error occurs
- Key time to reset the type-in search in listboxes
- Balloon delay time
- Time format for files (follows python&C syntax)
- Time out to attach to a running simulation
- Automatic refresh interval of information

Configuration Dialog: Run



- Define rules for "spawn"ed named generation
  Default: \I\a
  \I input name
  \a aa,ab,ac..
  \A AA, AB, AC…
  \n 1,2,3,…
  \2 01, 02, 03, …
  \3 001, 002, 003…
  \4 0001, 0002, 0003,…
  e.g. To run in a subdirectory each job **\2/\I**
- Custom queues can be added in the list box (Right-click to add/remove a queue)
- Example qfluka.sh to submit on pbs/torque cluster system can be found on the flair web site

Programming Interface: API

There is work presently going on to decouple the functionality from the interface, some of the basic classes can be used to input processing

file: Input.py - to manipulate input files

import Input

Input.init([database]) to initialize the database of cards

Most commonly used classes:

Card	containing the description of each card
Input	manipulating the FLUKA input file

file: Project.py - to manipulate project files

API: class Card

Constructor: Input.Card(tag, what [,comment [,extra]])

what is a list starting with what[0]=sdum

Important Methods:

setWhat(n, value)	set value to what#n
nwhats()	return number of whats
what(n)	return value of what#n
numWhat(n)	return numeric value of what#n
intWhat(n)	return integer value of what#n
clone()	return a copy of the card
setEnable(e)	enable/disable card

API: class Input

Constructor: Input.Input()

initialize the structure to hold an input file

Important Variables:

cardlist	a list with pointers to cards
cards	a dictionary with pointers to cards grouped per tag

Important Methods:

read(filename)	read input from file
write(filename)	write input to filename
addCard(card,pos)	add card to position pos (or end of file)
delCard(pos)	delete card from position pos
preprocess()	preprocess input to check for active cards
setEnable(e)	enable/disable card

API: class Project

Constructor: Project.Project()

initialize the structure to hold a project file

Important Methods:

clear()	to re-initialize project
load(filename)	load project from file filename
save([filename])	save project to filename
runCmd(run)	create run command

API: example

Read an input file and modify the random number seed

```
import Input
inp = Input.Input()
inp.read("test.inp")
try:
    rndcard = self.cards["RANDOMIZ"][0]
    rndcard.setWhat(2,5723)
except:
    print "No RANDOMIZe card found"
    sys.exit(0)
inp.write("test2.inp")
```

API: .flair file structure

```
# comments
Variable: Value
Notes:
    multi-line values are terminated with
    Ctrl-L

Run: name
    ...                          Block of Run related information
    Data:
    ...                          ... Including Data processing information
    EndData
EndRun
Plot: name
...                              Plot related informations
EndPlot
```