Fred 3.50 status report

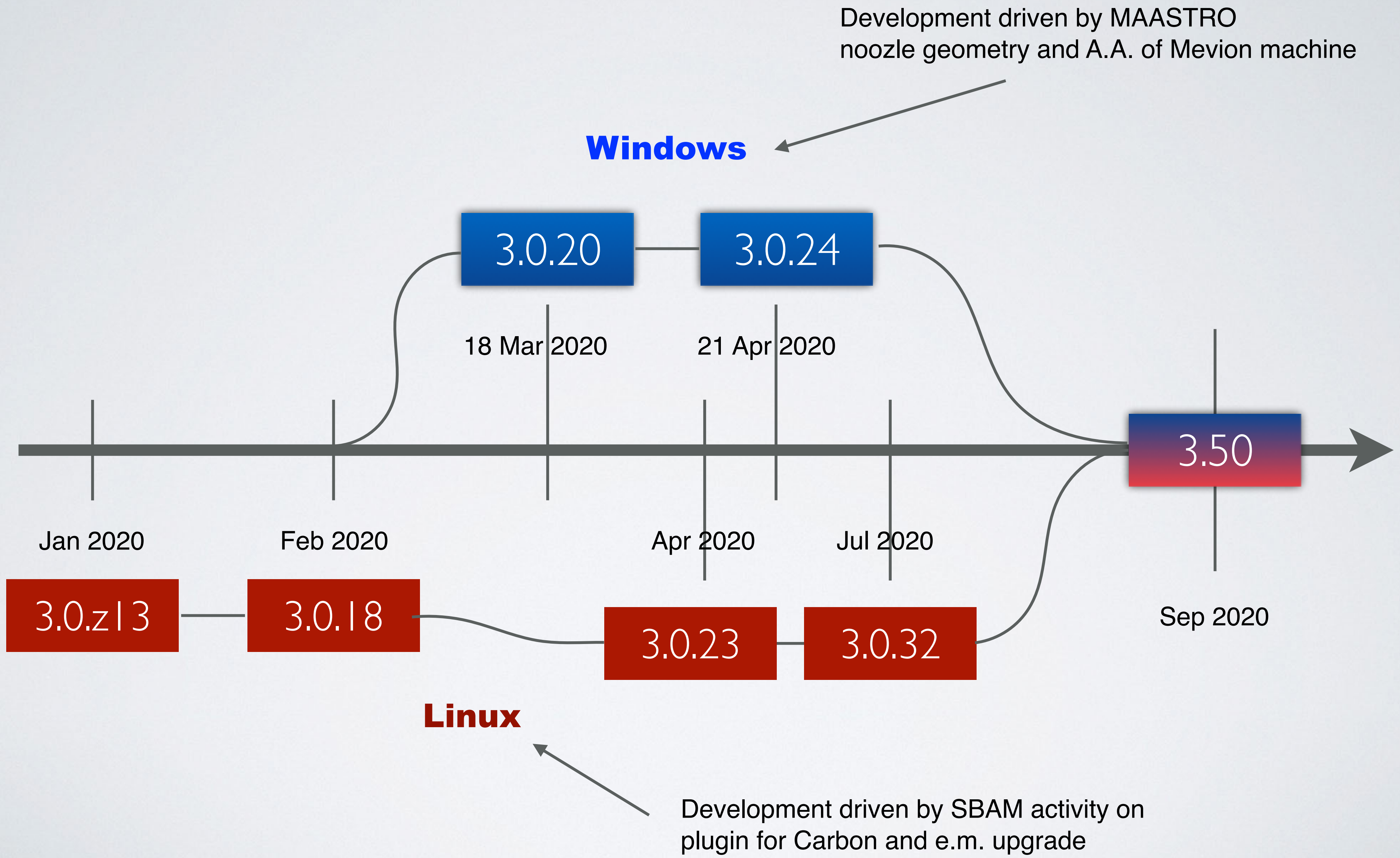
A. Schiavi

SBAI - URLS - 05/10/2020

FRED fast-MC platform

- particle tracking with class II MC algo in voxelized geometry
- **fred-p** : clinical stable version for proton therapy (Maastro; Kraków; PSI - Zurich)
- **fred-C** : plugin-level implementation (see Micol's PhD)
- **fred-em** : plugin level implementation, very good agreement in the therapeutic energy range for IORT (~10 MeV) and for Flash (70 MeV - 100 MeV) (see preparatory work by Gaia, Patrizia and Giacomo)
- **Optx** : ported to multicore parallelism; adapted to DMF; basis of next-generation DMF-aware Flash-therapy TPS

Version timeline



Development driven by MAASTRO noozle geometry and A.A. of Mevion machine
Windows
3.0.20
3.0.24
18 Mar 2020
21 Apr 2020
3.50
Jan 2020
Feb 2020
Apr 2020
Jul 2020
Sep 2020
3.0.z13
3.0.18
3.0.23
3.0.32
Linux
Development driven by SBAM activity on plugin for Carbon and e.m. upgrade

What’s new in v 3.50 for SBAM

- Input parsing and error reporting: no undefined or unknown parameters anymore => safer approach => WYWIWYG
- Multiple verbosity levels : 0 - 5 (minimal to debug)
- Plugin parameters moved inside plugin<…> section
- mhd scripts : major update of argument parsing and features
- Python preparser (experimental)

Verbosity levels

1. level is taken from **FRED_VERBOSE** env variable (if present)

2. built-in level is set to **3** (mid verbosity)

3. level can be then defined in input file (see below for sections)

4. cmdline option -V0…-V5 override any previous settings

```
verbose: all 0
verbose: physics 0
verbose: delivery 0
verbose: plugin 0
verbose: source 0
verbose: geometry 0
verbose: environment 0
verbose: input 0
verbose: materials 0
verbose: radiobiology 5
```

Plugin parameters

Now plugin parameters have to be put inside a multiline

plugin<...> section

eg. plugin code :

```
MCSmodel = getIntParam("MCSmodel",1);
cout <<"MCS model::" << MCSmodel << "    0-->Highland,
1-->Molierebethe" << endl;
```

Inputfile Before

```
...

MCSmode = 0
...
```

Inputfile Now

```
...
plugin<

MCSmode = 0

plugin>
...
```

Python preparser

Now input files are merged, stripped and evaluated by a python preparser.

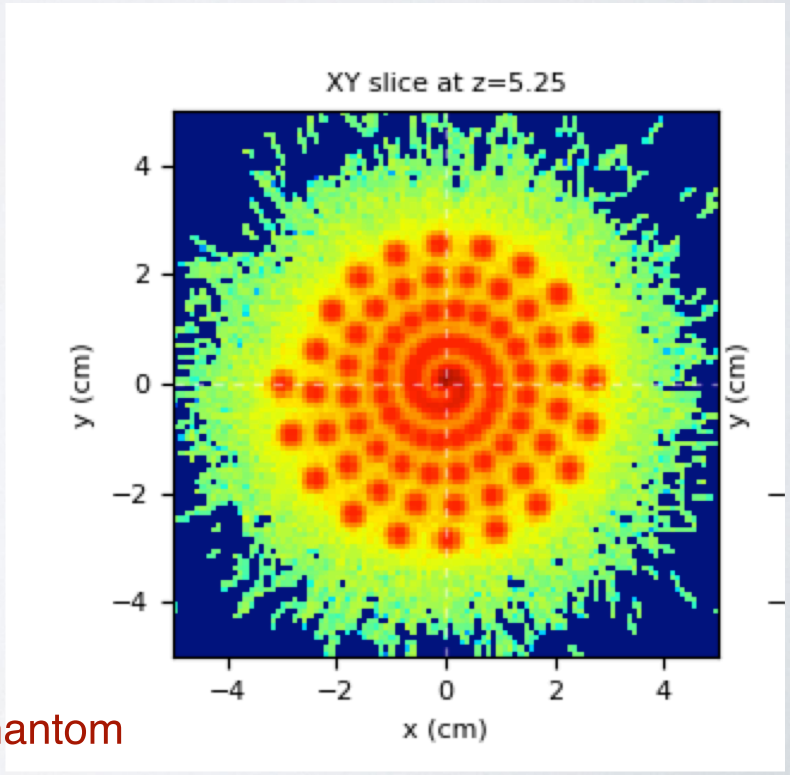
This allows to expand and extend syntax to pythonic language.

New constructs and statements that have been added:

- **def:** directive used to define parameters and variables
- **func:** directive used to define functions
- **for()<...>** directive implementing loops
- **if()<...>** directive for conditional execution

Example: looper.inp

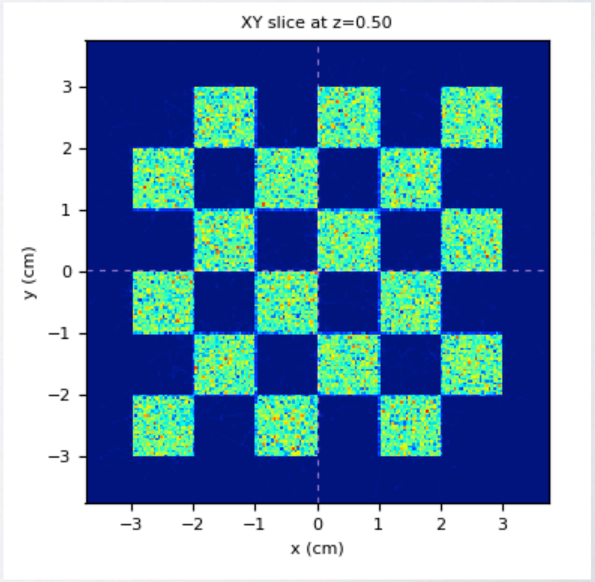
```
# create a spiraling irradiation pattern by displacing the phantom


region: phantom ; L=[10,10,10] ; voxels = [101,101,20]


def: N = 5 # number of turns
def: nspots = 100 # total number of spots
def: Rmax = 3 # major radius
def: Rmin =0 # minor radius


func: r(th) = Rmax-(Rmax-Rmin)*th/(N*2*pi)

for(th in np.linspace(0,N*2*pi,nspots))<

    def: x = r(th)*cos(th)
    def: y = r(th)*sin(th)
    transform: phantom move_to $x $y 0
    deliver: all

for>
```



NB: here it generates a series of geometry trasformations on the phantom

Example: chessboard.inp

```
# create a 'chessboard' irradiation pattern by displacing the field
# use a single spot with square cross-section

nprim=1e4

def: side = 5
def: spotSize = 1
def: spotSpacing = spotSize
def: nspot = int(side/spotSpacing)


pbXsec=box
pbFWHM=$spotSize

region: phantom ; L=[${side*1.5},${side*1.5},1] ; voxels = [200,200,1] ; pivot=[0.5,0.5,0]

for(ix in range(nspot+1))< # control points in x
    for(iy in range(nspot+1))< # control points in y

        if(mod(ix,2)==mod(iy,2))< # choose alternate squares
            def: x = -side/2 + ix*spotSpacing
            def: y = -side/2 + iy*spotSpacing
            transform: field_0 move_to $x $y -50
            deliver: all
        if>
    for>
for>
```

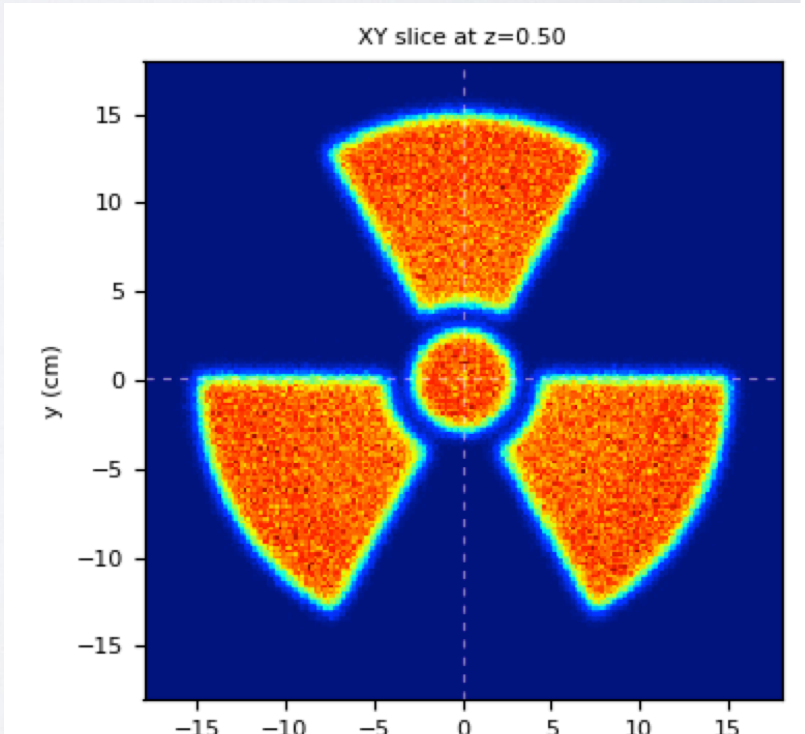


NB: here it generates a series of geometry trasformations on the field

Example: radiation_hazard.inp

```
# create a 'radiation hazard' irradiation pattern
# use many spots with gaussian cross-section (programmatically build an rtplan)
# use typical interspot spacing for uniform irradiation

def: R = 3
def: side = 6*R*2
def: spotSize = R/3

pbXsec=gauss
pbFWHM=$spotSize

func: thSpacing(r) = spotSize/r/3
…

# build rtplan

def: ipb = 0 # spot (i.e. pencil beam) index

# central disc
for(r in np.arange(0,R,spotSize/3))<
    for(th in np.linspace(0,2*pi,int(2*pi/thSpacing(r)),endpoint=False))<
        def: x = r*cos(th)
        def: y = r*sin(th)
        pb: $ipb 1 $x $y
        def: ipb = ipb+1
    for>
for>
…
…
```

NB: here it generates an rtplan



Material re-definition

Where to look for stopping tables

Original material definition: has to aligned with stopping tables mat.prop file

```
fSPTablesDir=“sptables"

material: C ; rho =  2.0 ; Ipot =  81.0 ; Lrad = 42.70 ; composition=[C]

material:  Graphite ; basedOn = C; rho=2.0 ; Ipot = 78 ; Lrad =42.7
```

Re-definition of material properties:

- density
- ionization potential
- radiation length

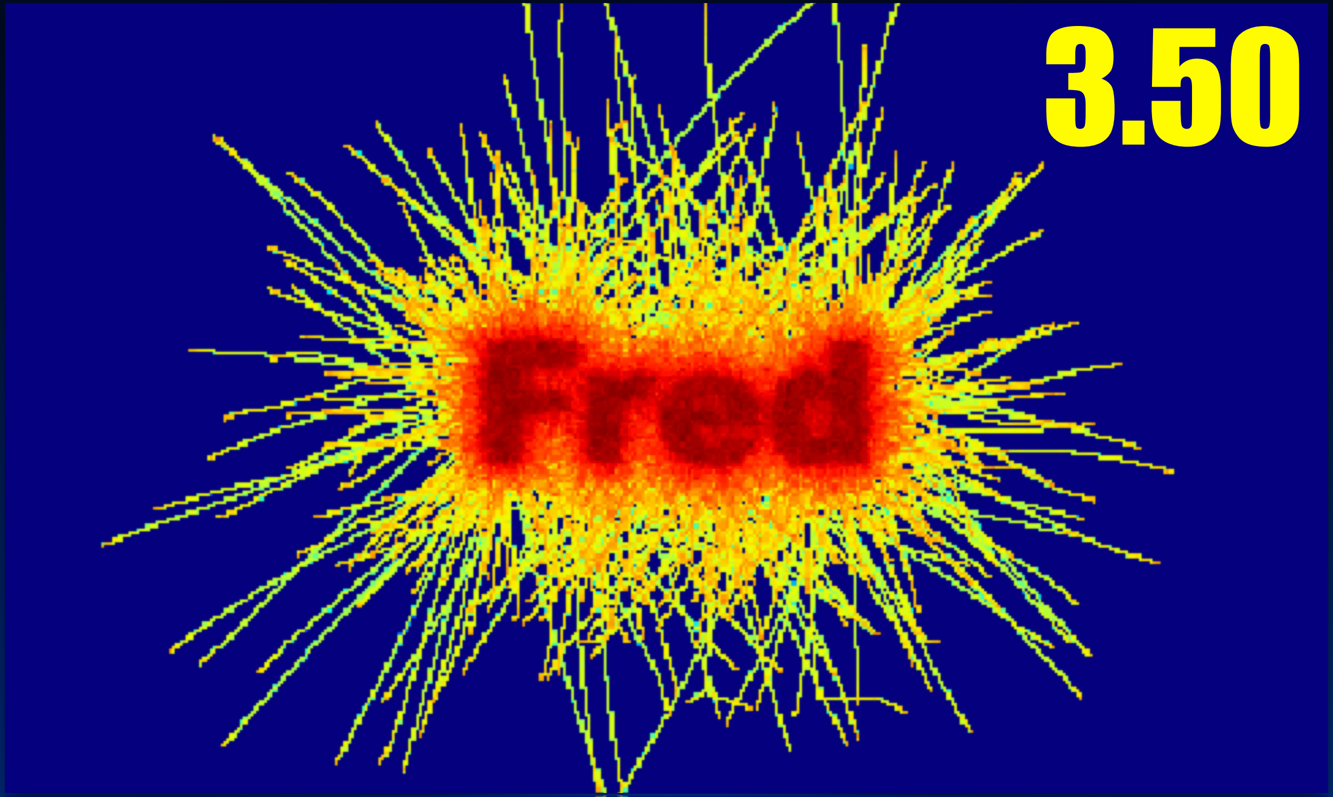
mhd_*.py scripts

The python scripts that can help manipulation and visualization of mhd map.

Where: in the **$FREDDIR/curr/scripts/mhd_scripts**

They are automatically added to the user $PATH and they change and update with every new version. If you change current fred version, you change also the mhd_*.py scripts

Please use these tools: if you want to make changes, rename them and use symbolic links (not copies!)



NEXT STABLE RELEASE