MINFLUX New Data Format

MINFLUX Data Conversion

The exported Minflux data structure will be subject to changes in the new software version. The conversion from the previous to the new data structure can be done either via the new Imspector or using the attached MATLAB or Python scripts.

New Data Structure

The new structure consists of 20 fields. All fields, except for *dcr*, *lnc* and *loc,* are 1xN vectors for N localization events. Table 1 contains the description of the fields with the newly added highlighted in blue.

Table 1: New MINFLUX data format fields. New fields are highlighted in blue

Field Name	Class	Meaning
bot	logical	Begin of a trace
cfr	double	Center frequency ratio (efc/efo)
dcr	double	Detection channel ratio
ecc	uint32	Effective counts at offsets
eco	int32	Effective counts at center
efc	double	Effective Frequency at center
efo	double	Effective frequency at offset
eot	logical	End of trace
fbg	double	Estimated background frequency value
fnl	logical	Is final iteration
gri	int32	Grid index
itr	int32	Iteration index
loc	double	Localizations (potentially corrected)
lnc	double	Final localizations (no applied corrections)
sqi	int32	Sequence index
sta	int32	New status (number that encodes abort criteria)
thi	int32	Thread index
tid	int32	Trace ID
tim	double	From tic at itr 0
vld	logical	Is valid iteration

Previous Data Structure

The previous data structure consisted of 9 fields with *itr* being a nested structure of 21 fields. Table 2 contains the description of the fields with obsolete fields highlighted in red.

Table 2: Previous MINFLX data format fields. Obsolete fields are highlighted in red

Field Name	Class	Meaning
act	logical	Activation 1, when efo < fbg, 0 otherwise
dos	int32	Digital Output
gri	int32	Grid Index
itr	structure	Iteration
sky	int32	Maximum number of repetitions to localize an emitter
sqi	int32	Sequence index
tid	int32	Trace ID

tim	double	Time of localization of last iteration
vld	logical	Is valid iteration
		nested structure within iteration
tic	int64	Time stamp for every iteration (FPGA ticks 40 MHz = 25 ns)
loc	double	Localizations (potentially corrected)
lnc	double	Final localizations (no applied corrections)
eco	int32	Effective counts at offsets
ecc	int32	Effective counts at center
efo	double	Effective Frequency at center
efc	double	Effective frequency at offset
sta	int32	New status (number that encodes abort criteria)
cfr	double	Center frequency ratio (efc/efo)
dcr	double	Detection channel ratio
ext	double	Beampattern diameter
gvy	double	Galvo center position in y
gvx	double	Galvo center position in x
eoy	double	EOD position relative to galvo center in y
eox	double	EOD position relative to galvo center in x
dmz	double	Deformable mirror position in z
lcy	double	Localization position relative to beam in y
lcx	double	Localization position relative to beam in x
lcz	double	Localization position relative to beam in z
fbg	double	Estimated background frequency value

MINFLUX Data Conversion Scripts

Scripts for converting between the new and old data format are also available on the abberior wiki page.

1.1 Matlab script

Matlab script to convert .mat Minflux data from previous to new format:

```
% Load .mat file previous format
clear
load("/MATLAB Drive/NonameOLD.mat")

% Compute new indexes
keepidx = ~isnan(itr.lnc(:,:,1))';
keepidx = keepidx(:);

% Exclude obsolete fields
itr = rmfield(itr,...
    {'tic','ext', 'lcy', 'lcx', 'lcz',
'dmz','eox','eoy','gvx','gvy','ext','tic'});
clear act dos sky

% Transform variables
tid = Transform2New(repmat(tid',6),keepidx);
gri = Transform2New(repmat(gri',6),keepidx);
tim = Transform2New(repmat(tim',6),keepidx);
vld = Transform2New(repmat(vld',6),keepidx);
sqi = Transform2New(repmat(sqi',6),keepidx);
cfr = Transform2New(itr.cfr,keepidx);
dcr = Transform2New(itr.dcr,keepidx);
```

```
ecc = Transform2New(itr.ecc,keepidx);
eco = Transform2New(itr.eco,keepidx);
efc= Transform2New(itr.efc,keepidx);
efo = Transform2New(itr.efo,keepidx);
fbg = Transform2New(itr.fbg,keepidx);
loc = Transform2New(itr.loc,keepidx);
lnc = Transform2New(itr.lnc,keepidx);
sta = Transform2New(itr.sta,keepidx);
itr = Transform2New(itr.itr,keepidx)+1;


thi = int32(zeros(size(vld)));
bot = diff([1 tid])>0;
eot = diff([tid 1])>0;
fnl = itr == max(itr);

function [NewVar] = Transform2New(Var, idx)
    if length(size(Var)) == 2
        buffer = Var';
        buffer = buffer(:);
        NewVar = buffer(idx)';
    else
        NewVar = zeros(sum(idx),3);
        for ii = 1:3
        buffer = Var(:,:,ii)';
        buffer = buffer(:);
        NewVar(:,ii) = buffer(idx);
        end
    end
end

clear keepidx
```

1.2 Python script

```
# -*- coding: utf-8 -*-
"""
Created on Mon Apr 15 14:59:02 2024


@author: m.lima
"""
import numpy as np
# load .npy file previous format
mfxOld=np.load("C:\\Users\\...\\  NonameOLD.npy")
# get indemfxDataNes to keep
kidx = ~np.isnan(mfxOld['itr']['loc'][:,:,0].flatten())
numEl = np.count_nonzero(kidx)
reps = np.shape(mfxOld['itr']['itr'])[1]
```

```
# Define data type for new fields

newDtype = [('vld', '?'), ('fnl', '?'), ('bot', '?'),('eot', '?'), ('sta', 'u1'),('tim', '<f8'), ('tid', '<u4'),('gri', '<u4'), ('thi', 'u1'),('sqi', 'u1'), ('itr', '<i4'), ('loc', '<f8', (3,)), ('lnc', '<f8', (3,)), ('eco', '<u4'), ('ecc', '<u4'), ('efo', '<f4'),('efc', '<f4'), ('fbg', '<f4'), ('cfr', '<f2'), ('dcr', '<f2', (2,))]


# Create an empty array with the specified datatype

mfxDataN = np.zeros(numEl, dtype=newDtype)


def Transform2New(Var, idmfxDataN):
    if Var.ndim == 2:
        NewVar  = Var.flatten()[idmfxDataN]
        return NewVar
    else:
        NewVar = np.zeros((sum(idmfxDataN), 3))
        for ii in range(3):
            buffer =  Var[:,:,ii].flatten()
            NewVar[:,ii] = buffer[idmfxDataN]
    return NewVar


def RepeatCols(Var,colN):
    NewVar = np.zeros((np.shape(Var)[0], colN))
    for i in range(0,colN):
        NewVar[:,i] = Var
    return NewVar


mfxDataN['tid'] = Transform2New(RepeatCols(mfxOld['tid'],reps), kidx)

mfxDataN['gri'] = Transform2New(RepeatCols(mfxOld['gri'],reps), kidx)

mfxDataN['tim'] = Transform2New(RepeatCols(mfxOld['tim'],reps), kidx)

mfxDataN['vld'] = Transform2New(RepeatCols(mfxOld['vld'],reps), kidx)

mfxDataN['sqi'] = Transform2New(RepeatCols(mfxOld['sqi'],reps), kidx)


# fields from previous nested itr array

mfxDataN['cfr'] = Transform2New(mfxOld['itr']['cfr'], kidx)

mfxDataN['dcr'][:,0] = Transform2New(mfxOld['itr']['dcr'], kidx)
```

```
mfxDataN['ecc'] = Transform2New(mfxOld['itr']['ecc'], kidx)
mfxDataN['eco'] = Transform2New(mfxOld['itr']['eco'], kidx)
mfxDataN['efc'] = Transform2New(mfxOld['itr']['efc'], kidx)
mfxDataN['efo'] = Transform2New(mfxOld['itr']['efo'], kidx)
mfxDataN['fbg'] = Transform2New(mfxOld['itr']['fbg'], kidx)
mfxDataN['sta'] = Transform2New(mfxOld['itr']['sta'], kidx)
mfxDataN['itr']= mfxOld['itr']['itr'].flatten()[kidx]
mfxDataN['loc']= Transform2New(mfxOld['itr']['loc'], kidx)
mfxDataN['lnc']= Transform2New(mfxOld['itr']['lnc'], kidx)

# new fields
mfxDataN['bot'] = np.insert(np.diff(mfxDataN['tid'])> 0, 0, bool(1))
mfxDataN['eot'] = np.insert(np.diff(mfxDataN['tid'])> 0, numEl-1, bool(1))
mfxDataN['fnl'] = mfxDataN['itr'] == np.max(mfxDataN['itr'])
mfxDataN['thi'] = np.zeros(numEl)

del kidx, numEl, newDtype,mfxOld, reps
```