1.1 Adding/configuring SmallData Production

In most cases, the producer file should only be modified between the two flags:

```
##########################################################
##
## User Input start -->
##
##########################################################

...

##########################################################
##
## <-- User Input end
##
##########################################################
```

This user section is split in two parts, one for run-dependent parameters, usually mainly involving the more complex area detector analysis, which is likely to vary from run to run, and a run-independent part, which allows for the addition of EPICS PVs or analog inputs/outputs for example.

Run-independant variables

This will describe how to tweak some of the default data:

This is done mostly in the block in the producer file below (<smalldata_tools>/producers/smd_producer.py).

```
##########################################################
# run independent parameters
##########################################################
#aliases for experiment specific PVs go here
epicsPV = ['s1h_w']
#tt calibration parameters (if passing None, init will use values used during recording
ttCalibPars = None # (or [] or [p0, p1, p2]
#aioParams=[[1],['laser'],[1.],[0.]]
aioParams=[]
```

EPICS PVs

Should you have user motors e.g. for sample motion for which you would like to save the position or if you are using a lakeshore for temperature control or have other remotely controlled data you would like to save, you will need to add those to the epicsPV line. For the exact names, ask the beam line staff. Please note that 's1h_w' is only an example/placeholder. You can get the list of possible EPICS variables names/aliases by using

```
"detnames exp=<expname>:run=<run#> -e"
```

This will only work when you have the analysis environment setup by calling:

```
source /reg/g/psdm/etc/psconda.sh
```

Please note that these PVs will appear in event-by-event, but they are NOT time matched. The time values are close, but can differ by up to second, usually it's better than that, but it is not reliable.

Timetool calibration parameters

If we take a tt calibration run, we obtain a set of parameters for the conversion of the peak position on the timetool in pixel to ps. This should then be used in the DAQ and nothing needs to be done. Should you realize that this steps has been missed for at least some of the data or have you extracted better parameters, you can put the obtained parameters into the ttCalibPars parameter. They will be used for any new productions (e.g. for reprocessing of runs taken before the calibration was in place). If it is set to "None" or an empty list, the calibration saved in the data is used.

In some hutches (XPP), this can be ignored, as the calibration constants are already implemented upstream and the ps time-tool values are reliable.

Analog Input

Values read in from our analog inputs are timestamped at 120Hz and saved in the data using the channel name. You can select channels to be saved, give them an alias and possibly apply a (linear) conversion.

`aioParams=[[1],['laser'],[1.],[0.]]`

First argument (necessary if using): list of channel numbers of interest (from 0-15)

Second: argument (necessary if using): list of channel names

Third: argument (optional, defaults to 1.): conversion factor

Fourth: argument (optional, defaults to 0.): offset

Addition of the time tool or Wave8 traces

If you have been told you might have to reanalyze the time tool data or you have past experience and feel you can gain by doing so, you can add the event-by-event trace of the OPAL camera used for the time tool like this:

```
############################################################
## User Input start -->
############################################################
##adding raw timetool traces:
#defaultDets.append(ttRawDetector(env=ds.env()))
##adding wave8 traces:
#defaultDets.append(wave8Detector('Wave8WF'))
############################################################
## <-- User Input end
############################################################
```

Time tool analysis

XTCAV values

Run-dependent variables: Area detectors analysis

In order to keep the filesizes small to avoid issue when analysis the smallData files, we try to extract the important information gleaned from areaDetectors at an event-by-event basis and only save these pieces of data. Implementation examples of each of the analysis functions that are readily available can be found in `<smalldata_tools>/producers/smalldata_producer_template.py`.

Please reach out to your data and controls POC to discuss analysis needs for your experiments. The POC will make sure the functions you need are prepared in the main producer file, and you should only have to change the argument's values, such as region of interest or thresholds, according the ongoing experimental conditions.

Generally speaking, the parameters for each analysis functions are set from a run-dependent logic as shown here:

Analysis function parameters

```
def getFuncParam(run):
        """
    """
    if isinstance(run,str):
        run=int(run)
    ret_dict = {}
    if run<21:
            func_dict = {}
        func_dict['param1'] = <param>
        func_dict['param2'] = <param>
        func_dict['param3'] = <param>
        ret_dict['detname'] = func_dict
        else:
            func_dict = {}
        func_dict['param1'] = <param>
        func_dict['param2'] = <param>
        func_dict['param3'] = <param>
        ret_dict['detname'] = func_dict
        return ret_dict
```

Several analysis functions or detectors can be set up this way for example:

- ROI
- Azimuthal integration
- Droplet
- Projection
- ...

See 1.2 Area Detector treatment with DetObject and child pages for details on the different analysis functions.