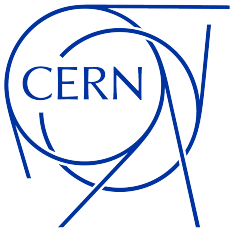

Universal Import Routines for Accelerator Measurement Data

Accelerator Systems Department
Radio Frequency Group-Beams & RF Studies (BR) Section

Date:August 17, 2023

By: Akasha Kabboul
Supervisors: Simon Albright, Mariangela Marchi

Abstract

Longitudinal measurement data is commonly stored in plaintext files or the logging system NXCALS. To facilitate its analysis, we have developed standardized routines that enable the import of these files and their conversion into standard Python data structures. These routines provide a convenient and efficient means of handling longitudinal measurement data, facilitating further analysis and exploration in Python-based environments.

Contents

1 Introduction

Longitudinal phase space tomography is essential in most of the CERN synchrotrons to measure longitudinal beam parameters. Up until recently, this task has relied on a Fortran95-based code. However, a notable shift occurred in 2021 with the release of an updated version of the code, now written in Python and C++[1]. This shift will benefit from a new Python tool for importing accelerator measurement data and integrating it into the tomography code.

Accelerator measurement data is commonly stored in several ways (.dat files, .asc files, data logged on to NXCALS etc.). This variety in the type of storage makes it more difficult to import this data for analysis. Thus, the aim of this project is to create universal import routines for this data.
The raw data is typically stored in 1D or 2D arrays, which can be visualized as waterfalls as shown in Fig. 1.

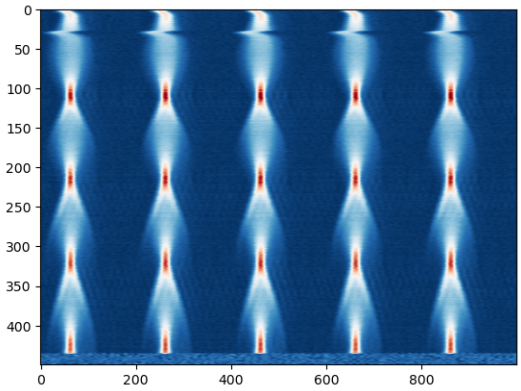

Figure 1: Bunch Profiles

2 Data Classes

The main aim of the project is to store the accelerator measurement data in dataclasses. Dataclasses "provide a decorator and functions for automatically adding generated special methods to user-defined classes" [2]. Dataclasses are a convenient and concise way of defining classes that are primarily meant to store data. Using dataclasses makes it easier for the user to get the value of a specific property or parameter by calling it using *result.parametername*, where parametername is the data entry required.

Two different dataclasses have been created: one that stores data from .asc files, and another to store data from .dat files and data logged on to NXCALS. The reason for this categorization is the difference between the data provided in each of the two categories. While the data in .dat files and logged on NXCALS contains similar fields, data in .asc files contains a different set of parameters.

3 Data Extraction from .dat and .asc files

The process of writing codes that work for .dat and .asc files is fairly similar as they are both ASCII files, however, there is a difference between the parameters stored in each file-type. The user has to input the file path, the function then opens the file and iterates over each line. Then, to store the value, the function first checks if the field name belongs to an "integer list" (which contains all the parameters whose values should be stored as an integer), if it does, the value is stored as an integer, otherwise, it is stored as a float.

4 Data Extraction from NXCALS

"NXCALS accepts data from different data sources/systems. Each system keeps the time-based data for its entities in NXCALS system i.e. each entity has a state modelled by a set of properties with their values at a given moment in time"[3]. Data logged onto NXCALS must be extracted differently. Although we want the same output, the process of acquiring the data requires either a time window when measurements were logged onto the system or a timestamp for a specific entry. Additionally, a ring number is required because we are dealing with measurements in the Proton Synchrotron Booster (PSB), which is made up of four superimposed synchrotron rings[4]. The code then produces an output similar to that of the .dat and .asc file functions.

The following is an example of the function used to extract tomoscope data from NXCALS using a time window:

```
window_data = DevicePropertyDataQuery.builder(spark).system('CMW')\
              .startTime(str(start_time))\
              .endTime(str(end_time)).entity()\
              .parameter(f'BA{Ring_Number}.TOMOINPUT/{prop_name}')\
              .build().select(param)
```

The timestamp function has the following format:

```
timestamp_data = DataQuery.builder(spark).byEntities() \
                 .system('CMW').atTime(timestamp) \
                 .entity() \
                 .keyValue('device', f'BA{Ring_Number}.TOMOINPUT') \
                 .keyValue('property', prop_na).build().select(param)
```

Here, the system specified is "CMW" or Controls Middleware which is a framework that provides communication infrastructure for all CERN accelerators and it enables us to acquire data from all types of equipment[5].

Measurements are grouped by identifiers of the form: DEVICE/Property#field. The function first identifies the device whose measurements are needed, then it iterates over all the properties and their corresponding fields, and outputs their values. For instance, the properties and fields we're dealing with are shown in Table 1:

Property	**Fields**
Acquisition	pickUpSensitivity, profiles
AcquisitionSettings	binsPerFrame, binWidth, deltaTurns, nFramesAcquired, nFramesIgnored
MachineParameters	avRadius, bendRadius, bField, bFieldDot, charge, gammaT, restMass
PreProcessSettings	leftEmptyBins, lowerCut, rebin, rightEmptyBins, upperCut
ReconstructionSettings	machineRefFrame, maxEnergy, nIterations, nSteps, partPerCellSqrt, phaseSpaceExtend, reconstructProfile, synchBin
RFSettings	hPrinciple, hRatio, phaseDifference, vDotHigher, vDotPrinciple, vHigher, vPrinciple
ScopeSettings	cTime, delay, ring, timeSpan, vOffset, vScale
SpaceCharge	couplingCoefficient, impedance, useSpaceCharge

Table 1: Table showing the properties and fields on NXCALS

5 Code Layout

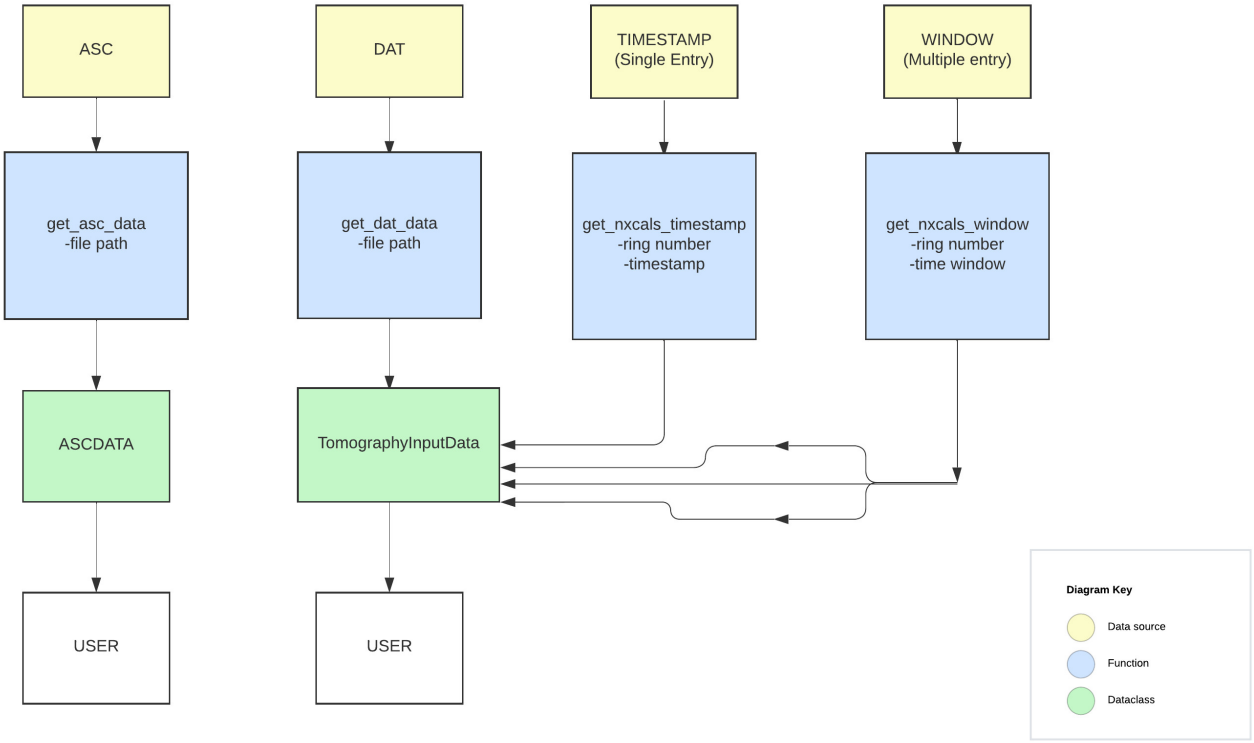

Figure 2: Flowchart visualizing the code

6 Conclusion

The project focused on developing improved methods for importing and processing accelerator measurement data. This was done by creating import routines for data stored in plaintext file formats or data logged on the NXCALS system. To achieve this, we used data classes, which are designed for storing data in a concise and organized manner. Two distinct data classes are created to accommodate data from .asc files and data from .dat files/NXCALS.

To extract data from the NXCALS system, which requires specific time window inputs or a timestamp for acquiring measurements from specific devices, two different functions were developed. These require a PSB ring number, which is used to specify the device, and either a time window or timestamp. These functions produce outputs similar to the .dat function.

Ultimately, the project aims to simplify data access and processing for users, allowing them to retrieve specific data fields directly using straightforward commands instead of converting data between formats.

References

[1] Longitudinal Phase Space Tomography Version 3 https://cds.cern.ch/record/2750116/files/Longitudinal%20Phase%20Space%20Tomography%20Version%203.pdf

[2] The Proton Synchrotron Booster https://home.cern/science/accelerators/proton-synchrotron-booster

[3] NXCALS Documentation https://nxcals-docs.web.cern.ch/current/

[4] Data Classes https://docs.python.org/3/library/dataclasses.html

[5] CERN. "CMW - Controls Middleware." CERN, https://wikis.cern.ch/display/MW/CMW+-+Controls+Middleware