

Tgt audience - <sup>not</sup> JINST?



## MAUS: The MICE Analysis User Software

### MICE Collaboration

NOTE: Need to finalize author list. Include from list of MAUS developers?

E-mail: durga@fnal.gov

collaboration

**ABSTRACT:** The Muon Ionization Cooling Experiment (MICE) has developed the MICE Analysis User Software (MAUS) to simulate and analyze experimental data. It serves as the primary codebase for the experiment, providing for offline batch simulation and reconstruction as well as online data quality checks. The software provides both traditional particle-physics functionalities such as track reconstruction and particle identification, and accelerator physics functions, such as calculating transfer matrices and emittances. The code design is object orientated, but has a top-level structure based on the Map-Reduce model. This allows for parallelization to support live data reconstruction during data-taking operations. MAUS allows users to develop in either Python or C++ and provides APIs for both. Various software engineering practices from industry are also used to ensure correct and maintainable code, including style, unit and integration tests, continuous integration and load testing, code reviews, and distributed version control. The software framework and the simulation and reconstruction capabilities are described.

**KEYWORDS:** MICE; Ionization Cooling; Software; Reconstruction; Simulation.

1?

---

## Contents

<b>1. Introduction</b>	<i>lc. or U.C.?</i>	<b>1</b>
1.1 The MICE experiment		1
1.2 Software Requirements		2
<b>2. MAUS</b>		<b>3</b>
2.1 Code design		3
2.2 Data Structure		6
2.2.1 Physics Data		6
2.2.2 Top Level Data Organisation		10
2.3 Data Flow		10
2.4 Testing		10
<b>3. Monte Carlo</b>		<b>11</b>
3.1 Beam generation		11
3.2 GEANT4		12
3.3 Geometry		12
3.4 Tracking, field maps and beam optics		13
3.5 Detector response and digitization		13
<b>4. Reconstruction</b>		<b>14</b>
4.1 Time of flight		14
4.2 Scintillating fiber trackers		14
4.3 KL calorimeter		15
4.4 Electron-muon ranger		15
4.5 Cherenkov		15
4.6 Global reconstruction		15
4.6.1 Global Track Matching		15
4.6.2 Global PID		16
4.7 Online reconstruction		16

- 
- has*
- 1. Introduction**
  - 2. 1.1 The MICE experiment** *capitalise or not? follow JINST (or whoever) template*
  - The Muon Ionization Cooling Experiment (MICE) sited at the STFC Rutherford Appleton Laboratory (RAL) *will deliver* the first demonstration of muon ionization cooling – the reduction of the phase-space of muon beams. Muon-beam cooling is essential for future facilities based on muon acceleration, such as the Neutrino Factory or Muon Collider [1, 2]. The experiment was designed
- [citation]*

7 to be built and operated in a staged manner. In the first stage, the muon beamline was commis-  
8 sioned [3] and characterized [4]. The present configuration shown in figure 1 will be used to study  
9 the factors that determine the performance of an ionization cooling channel and to observe for the  
10 first time the reduction in transverse emittance of a muon beam.

11 The MICE Muon Beam line is described in detail in [3]. There are 5 different detector sys-  
12 tems present on the beamline: time-of-flight (TOF) scintillators [5], threshold Cherenkov (CKOV)  
13 counters [6], scintillating fiber trackers [7], a sampling calorimeter (KL) [4], and the Electron Muon  
14 Ranger (EMR) – a totally active scintillating calorimeter [8]. The TOF detector system consists of  
15 three detector stations, TOF0, TOF1 and TOF2, each composed of two orthogonal layers of scintil-  
16 lator bars. The TOF system is used to determine particle identification (PID) via the time-of-flight  
17 between the stations. Each station also provides a low resolution image of the beam profile. The  
18 CKOV system consists of two aerogel threshold Cherenkov stations, CKOVA and CKOVB. The  
19 KL and EMR detectors, the former using scintillating fibers embedded in lead sheets, and the latter  
20 scintillating bars, form the downstream calorimeter system.

21 The tracker system consists of two scintillating fiber detectors, one upstream of the MICE  
22 cooling cell, the other downstream, in order to measure the change in emittance across the cooling  
23 cell. Each detector consists of 5 stations, each station having 3 fiber planes, allowing  
24 precision measurement of momentum and position to be made on a particle-by-particle basis.

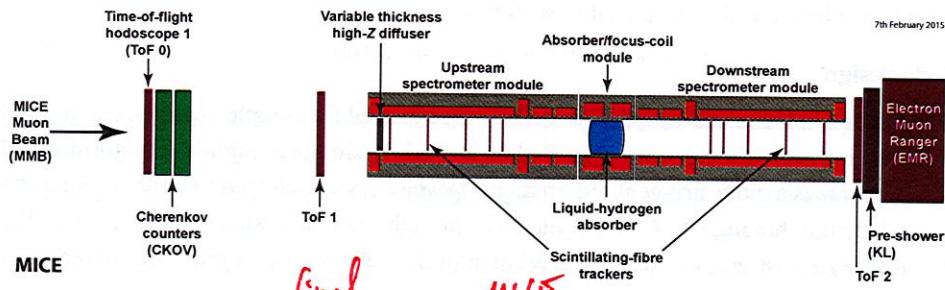


Figure 1. Schematic diagram of the configuration of the experiment. The red rectangles represent the coils of the spectrometer solenoids and focus coil. The individual coils of the spectrometer solenoids are labelled E1, C, E2, M1 and M2. The various detectors are also represented.

## 25 1.2 Software Requirements

26 The MICE software must serve both the accelerator-physics and the particle-physics needs of the  
27 experiment. Traditional particle-physics functionality includes reconstructing particle tracks, iden-  
28 tifying them, and simulating the response from various detectors, while the accelerator-physics  
29 aspect includes the calculation of transfer matrices and Twiss parameters and propagating the  
30 beam envelopes. All of these require a detailed description of the beamline, the geometries of  
31 the detectors, and the magnetic fields, as well as functionality to simulate the various detectors and  
32 reconstruct the detector outputs.

33 Given the complexity and the time-scale of the experiment, it is essential to ensure that the  
34 software can be maintained over the long-term. Good performance is also important in order to

35 ensure that the software can reconstruct data with sufficient speed to support live online monitoring  
36 of the experiment.

## 37 2. MAUS

38 The MICE Analysis User Software (MAUS) [9] is the experiment's simulation, reconstruction, and  
39 analysis software framework. MAUS provides a Monte Carlo (MC) simulation of the experiment,  
40 reconstruction of tracks and identification of particles from simulations and real data, and provides  
41 monitoring and diagnostics while running the experiment.

42 Installation is by a set of shell scripts with SCons [10] as the build tool. The codebase is main-  
43 tained with the GNU Bazaar revision control system [11] and is hosted on Launchpad [12]. MAUS  
44 has a number of dependencies on standard packages such as Python, ROOT [13] and GEANT4 [14]  
45 which are built as third party external libraries during the installation process. The officially sup-  
46 ported platform is Scientific Linux 6 [15] though developers successfully build on CentOS [16],  
47 Fedora [17], and Ubuntu [18] distributions.

48 Each of the MICE detector systems, described in section 1.1, are represented within MAUS.  
49 Their data structures are described in section 2.2 and their simulation and reconstruction algorithms  
50 in section 4. MAUS also provides "global" reconstruction routines, which combine data from  
51 individual detector systems to identify particle species by the likelihood method and a global track  
52 fit. These algorithms are also described in section 4.

### 53 2.1 Code design

54 MAUS is written in a mixture of Python and C++. C++ is used for complex or low-level algorithms  
55 where processing time is important, while Python is used for simple or high-level algorithms where  
56 development time is a more stringent requirement. Developers are allowed to write in either Python  
57 or C++ and Python bindings to C++ are handled through internal abstractions or SWIG [19]. In  
58 practice, all the reconstruction modules are written in C++ but support is provided for legacy mod-  
59 ules written in Python.

60 MAUS has an Application Programming Interface (API) that provides a framework on which  
61 developers can hang individual routines. The MAUS API provides MAUS developers with a well-  
62 defined environment for developing reconstruction code, while allowing independent development  
63 of the back-end and code-sharing of common elements, such as error handling and data-wrangling.

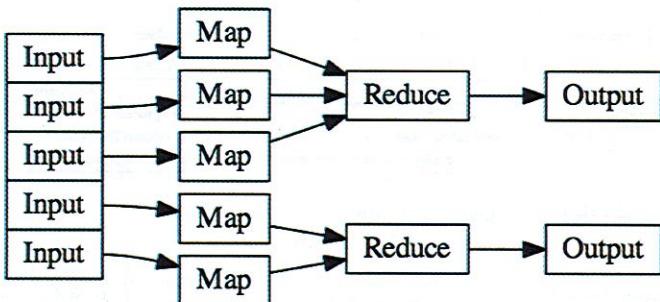
64 The MAUS data processing model is inspired by the Map-Reduce framework [20], which  
65 forms the core of the API design. Map-Reduce, illustrated in figure 2, is a useful model for paral-  
66 lelizing data processing on a large scale. For MAUS, the API was simplified to use *transformers*  
67 in place of maps, though these modules have retained the name *map*. A map process takes a sin-  
68 gle object as an input, which remains unaltered, and returns a new object as the output, whereas  
69 a transformer process alters the input object in place (in the case of MAUS this object is the *spill*  
70 class, see Section 2.2).

71 A *Module* is the basic building block of the MAUS API framework. Four types of module  
72 exist within MAUS:

73 1. **Inputters** generate input data either by reading data from files or sockets, or by generating  
74 an input beam;

(IS U.S. usage  
diff. fr. U.K.?)

- 75    2. **Mappers** modify the input data, for example by reconstructing signals from detectors, or  
 76    tracking particles to generate MC hits;
- 77    3. **Reducers** collate the mapped data and allow functionality that requires access to the entire  
 78    data set; and
- 79    4. **Outputters** save the data either by streaming over a socket or writing data ~~to disk~~.



**Figure 2.** A Map-Reduce framework.

80    Each module type follows a common, extensible, object-orientated class heirarchy, shown for the  
 81    case of the map and reduce modules in figure 3.

82    There are some objects that sit outside the scope of this modular framework but are never-  
 83    theless required by several of the modules. For instance, the detector geometries, magnetic fields,  
 84    and calibrations are required by the reconstruction and simulation modules, and objects such as  
 85    the electronics cabling maps are required to unpack data from the data acquisition (DAQ) source,  
 86    and error handling functionality is required by all of the modules. All these objects are accessed  
 87    through a static singleton *globals* class.

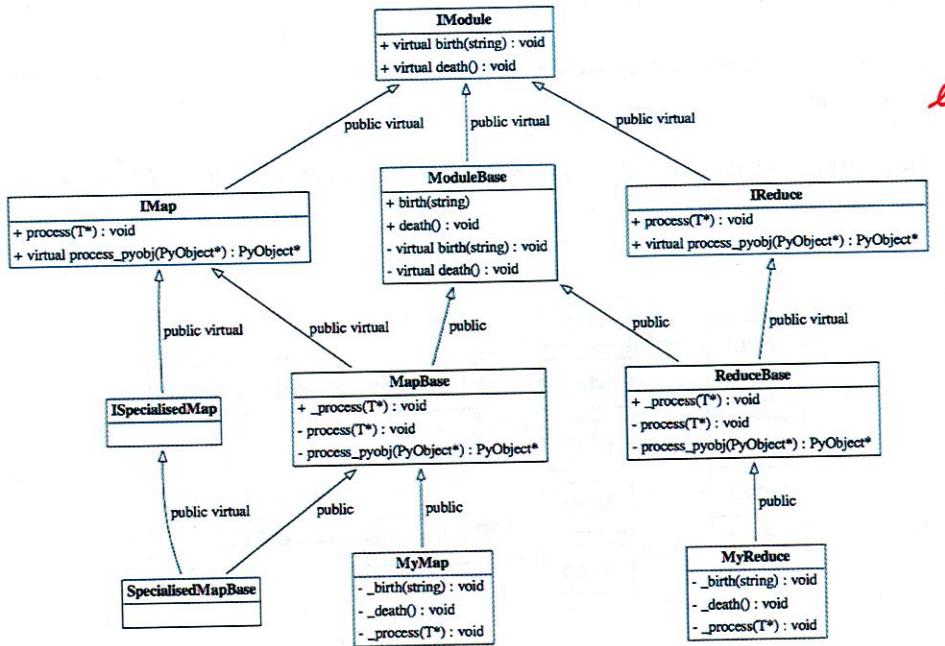
*in order*

88    MAUS has two execution concepts. A *job* refers to a single execution of the code, while a *run*  
 89    refers to the processing of data for a DAQ run or MC run. A job may contain many runs. Since data  
 90    are typically accessed from a single source and written to a single destination, Inputters and Out-  
 91    putters are initialized and destroyed at the beginning and end of a job. On the other hand, Mappers  
 92    and Reducers are initialized at the beginning of a run in order to allow run-specific information  
 93    such as electronic cabling maps, fields, and calibrations to be loaded.

94    The principal data type in MAUS, which is passed from module to module, is the *spill*. A  
 95    single spill corresponds to data from the particle burst associated with a dip of the MICE target  
 96    [3]. A spill lasts  $\sim 3$  ms and contains several DAQ triggers. Data from a given trigger defines a  
 97    single MICE *event*. In the language of the Input-Map-Reduce-Output framework, an Input module  
 98    creates an instance of spill data, a Map module processes the spill (simulating, reconstructing, etc),  
 99    a Reduce module acts on a collection of spills when all the mappers finish, and finally an Output  
 100   module records the data to a given file format.

*which?*

*up to?*



**Figure 3.** The MAUS API class hierarchy for Map and Reduce modules. The input and output modules follow a related design.  $T$  represents a templated argument. “+” indicates the introduction of a virtual void method, defining an interface, while “-” indicates a class implements that method, fulfilling that aspect of the interface. The functions *process\_pyobj* are the main entry points for Python applications, *process* the entry points for C++ applications. The framework can be extended as many times as is necessary, as exemplified by the “SpecialisedMap” classes.

101 Modules can exchange spill data either as C++ pointers or JSON [21] objects. In Python, the  
 102 data format can be changed by using a converter module and in C++, mappers are templated to a  
 103 MAUS data type and an API handles any necessary conversion to that type (see Fig. 3). *nm seq?*

104 Data contained within the MAUS data structure (see Section 2.2) can be saved to permanent  
 105 storage in one of two formats. The default data format is a ROOT [13] binary and the secondary  
 106 format is JSON. ROOT is a standard high-energy physics analysis package, distributed with MAUS,  
 107 through which many of the analyses on MICE are performed. Each spill is stored as a single entry  
 108 in a ROOT TTree object. JSON is an ASCII data-tree format. Specific JSON parsers are available  
 109 – for example, the Python *json* library, and the C++ *JsonCpp* [22] parser come prepackaged with  
 110 MAUS.

111 In addition to storing the output from the Map modules, MAUS is also capable of storing  
 112 the data produced by the *Reducer* modules using a special *Image* class. This class is used by  
 113 Reducers to store images of monitoring histograms, efficiency plots, etc. *Image* data may only be  
 114 saved in JSON format.

why static or not?

115 **2.2 Data Structure**

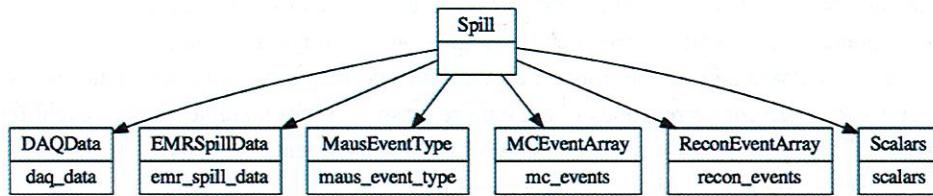
116 **2.2.1 Physics Data**

117 At the top of the MAUS data structure is the spill class which contains all the data from the simulation, raw real data and the reconstructed data. The spill is passed between modules and written to permanent storage. The data within a spill is organized into arrays of three possible event types:  
118 a *MCEvent* contains data which represents the simulation of a single particle traversing the experiment and the simulated detector responses; a *DAQEvent* corresponds to the real data for a single trigger; and a *ReconEvent* corresponds to the data reconstructed for a single particle event either arising from a MC particle or a real data trigger). These different branches of the MAUS data structure are shown diagrammatically in figures 4–9.

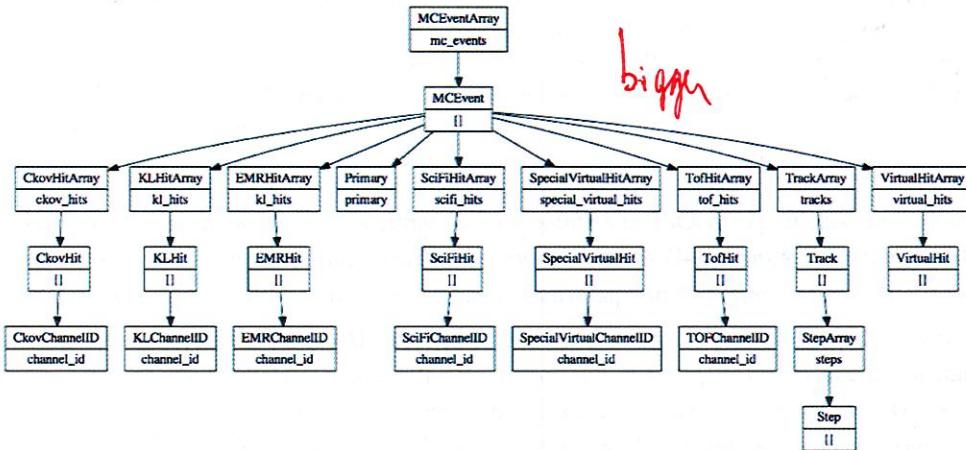
*both n one?*

125 The sub-structure of the the MC event class is shown in figure 5. The class is subdivided into events containing sensitive-detector hits (energy deposited, position, momentum) for each of the MICE detectors (see Section 1.1). The event also contains information about the primary particle that created the hits in the detectors.

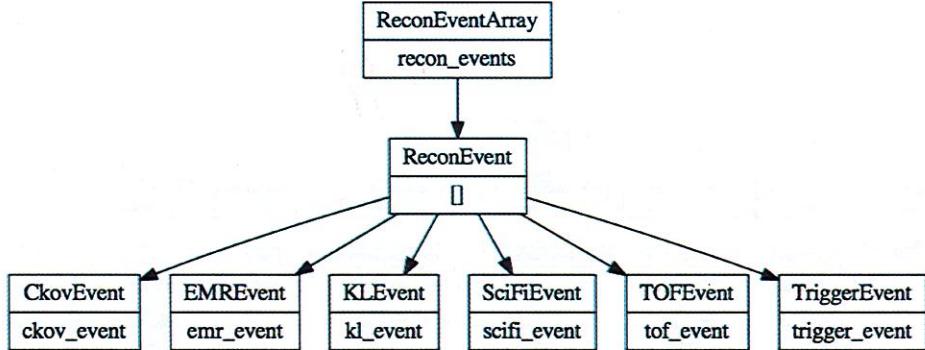
129 The sub-structure of the the reconstruction event class is shown in figure 6. The class is again subdivided into events representing each of the MICE detectors, together with the data from the trigger, and data for the global event reconstruction. Each detector class and the global reconstruction class has several further layers of reconstruction data. This is shown in figures 7–9.



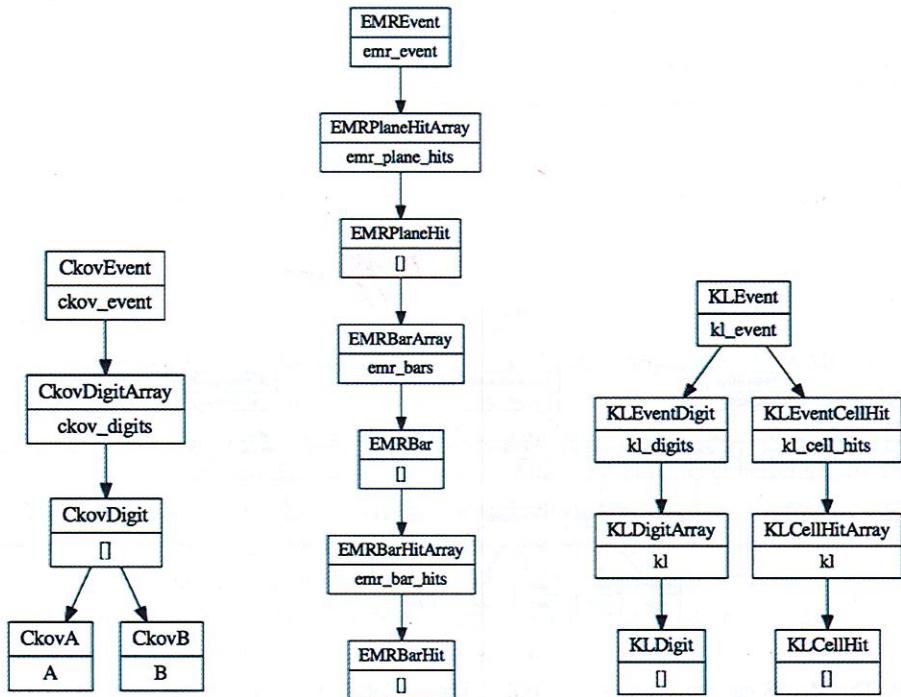
**Figure 4.** The MAUS output structure for a spill event. The top label in each box is the name of the C++ class and the bottom label is the json branch name.



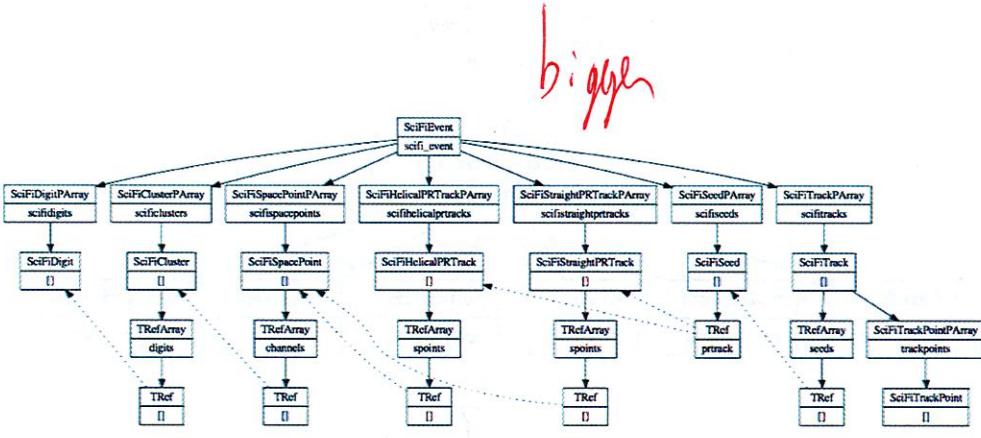
**Figure 5.** The MAUS data structure for MC events. The top label in each box is the name of the C++ class and the bottom label is the json branch name. [] indicates that child objects are array items.



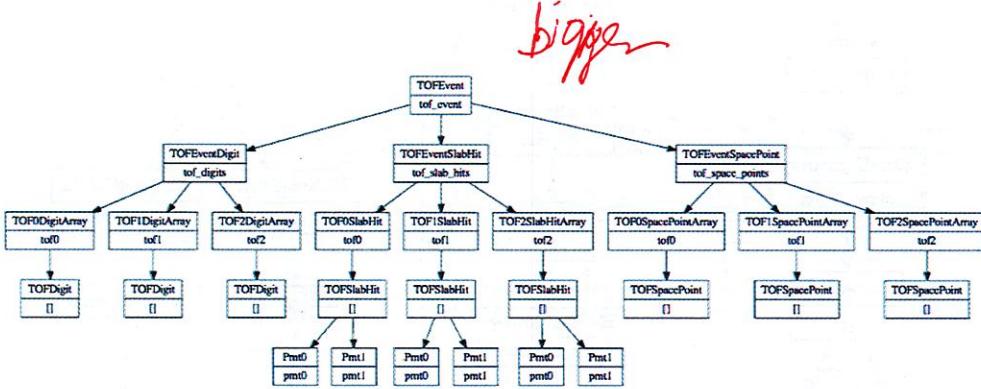
**Figure 6.** The MAUS data structure for reconstruction events. The top label in each box is the name of the C++ class and the bottom label is the json branch name.



**Figure 7.** The MAUS data structure for CKOV (left), EMR (middle) and KL (right) reconstruction events. The top label in each box is the name of the C++ class and the bottom label is the json branch name. [] indicates that child objects are array items.



**Figure 8.** The MAUS data structure for the tracker. The top label in each box is the name of the C++ class and the bottom label is the json branch name. [] indicates that child objects are array items.

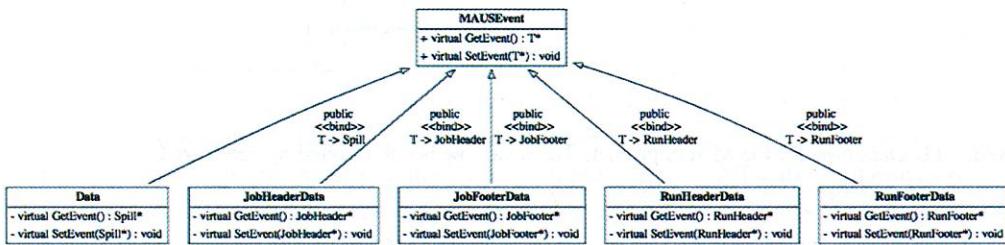


**Figure 9.** The MAUS data structure for the TOFs. The top label in each box is the name of the C++ class and the bottom label is the json branch name. [] indicates that child objects are array items.

### 133 2.2.2 Top Level Data Organisation

134 In addition to the spill data, MAUS also contains structures for storing supplementary information  
 135 for each run and job. These are referred to as *JobHeader* and *JobFooter*, and *RunHeader* and  
 136 *RunFooter*. The former represents data from the start and end of a job, such as the MAUS release  
 137 version used to create it, and the latter data from the start and end of a run, such as the geometry  
 138 ID used for the data processing. This may be saved to permanent storage along with the spill.

139 In order to interface with ROOT, particularly in order to save data in the ROOT format, thin  
 140 wrappers for each of the top level classes, and a templated base class, were introduced. This  
 141 allows the ROOT TTree, in which the output data is stored (see Section 2.2.1), to be given a single  
 142 memory address to read from. The wrapper for Spill is called *Data*, while for each of RunHeader,  
 143 RunFooter, JobHeader and JobFooter, the respective wrapper class is just given the original class  
 144 name with “Data” appended e.g., *RunHeaderData*. The base class for each of the wrappers is called  
 145 *MAUSEvent*. The class hierarchy is illustrated in Figure 10. ?



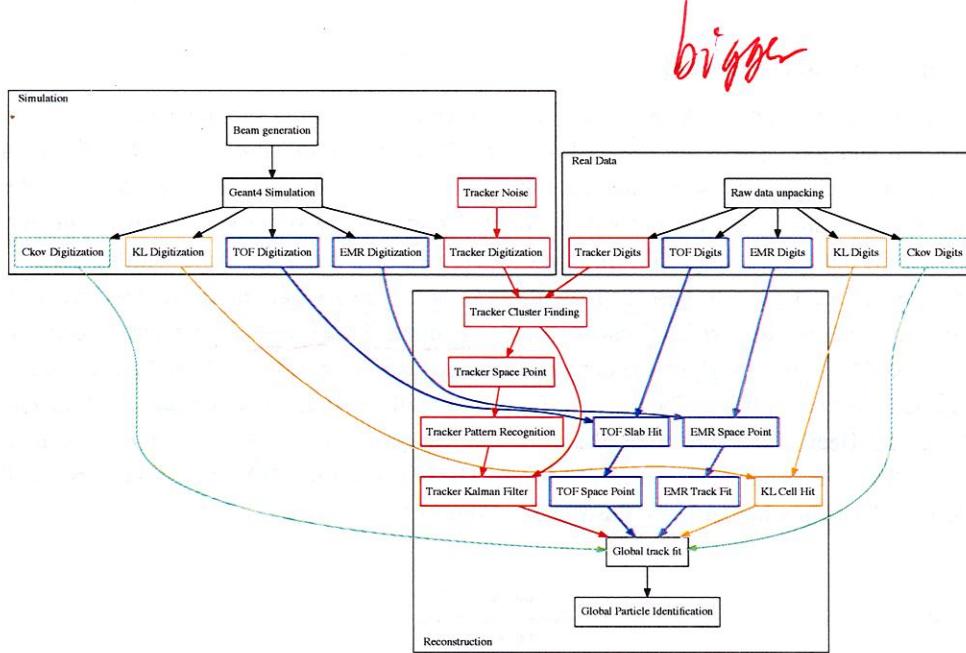
**Figure 10.** Class hierarchy for the wrappers and base class of the top-level classes of the MAUS data structure.

### 146 2.3 Data Flow

*depicted* diff.?  
 147 The MAUS data flow, showing the reconstruction chain for data originating from MC or real data,  
 148 is shown in figure 11. Each item in the diagram is implemented as an individual module. The  
 149 data flow is grouped into three principal areas: the simulation data flow used to generate digits  
 150 (electronics signals) from particle tracking; the real data flow used to generate digits from real  
 151 detector data; and the reconstruction data flow which illustrates how digits are built into higher  
 152 level objects and converted to parameters of interest. The reconstruction data flow is the same for  
 153 digits from real data and simulation. In the case of raw data, separate input modules are provided  
 154 to read either directly from the DAQ, or from archived data stored on disk. A reducer module for  
 155 each detector provides functionality to create summary histograms.

### 156 2.4 Testing

?  
 157 MAUS has a set of tests at the unit level and the integration level, together with code-style tests for  
 158 both Python and C++. Unit tests are implemented against a single function, while integration tests  
 159 operate against a complete workflow. Unit tests check that each function operates as intended by  
 160 the developer and achieve a high level of code coverage and good test complexity. Integration tests  
?



**Figure 11.** Data flow for the MAUS project. The data flow is color-coded by detector: CKOV - green, EMR - purple, KL - orange, TOF - blue, Tracker - red.

allow the overall performance of the code to be checked against specifications. The MAUS team provides unit test coverage that executes 70–80 % of the total code base. This level of coverage typically results in a code that performs the major workflows without any problem.

The MAUS codebase is built and tested using a Jenkins [23] continuous integration environment deployed on a cluster of servers. Builds and tests of the development branch are automatically triggered when there is a change to the codebase. Developers are asked to perform a build and test on a personal branch of the codebase using the test server before requesting a merge with the development trunk. This enables the MAUS team to make frequent clean releases. Typically MAUS works on a 4–8 week major-release cycle.

### 3. Monte Carlo

**A**MC simulation of MICE encompasses beam generation, geometrical description of detectors and fields, tracking of particles through detectors and digitization of the detectors' response to particle interactions.

#### 3.1 Beam generation

Several options are provided to generate an incident beam. Routines are provided to sample particles from a multivariate gaussian distribution or generate ensembles of identical particles ("pencil" beams). In addition, it is possible to produce time distributions that are either rectangular or triangular in time to give a simplistic representation of the MICE time distribution. Parameters, controlled by datacards, are available to control random seed generation, relative weighting of particle

?  
180 species and the transverse-longitudinal coupling in the beam. MAUS also allows the generation of  
181 a polarized beam by generating a spin vector from beam distributions.

182 Beam particles can also be read in from an external file created by G4Beamline [24] or ICOOL  
183 [25], as well as files in user-defined formats. In order to generate beams which are more realistic  
184 taking into account the geometry and fields of the actual MICE beamline, we use G4Beamline to  
185 model the MICE beam line from the target to a point upstream of the second quad triplet (upstream  
186 of Q4). The beam line settings, e.g., magnetic field strengths and number of particles to generate, are  
187 controlled through data-cards. The magnetic field strengths have been tuned to produce beams that  
188 are reasonably accurate descriptions of the real beam. Scripts to install G4Beamline are shipped  
189 with MAUS.

190 Once the beam is generated, the tracking and interactions of particles as they traverse the rest  
191 of the beamline and the MICE detectors is performed using GEANT4.

### 192 3.2 GEANT4

? *Nd figure*

193 The MICE Muon Beam line consists of a quadrupole triplet that captures pions produced when  
194 the MICE target intersects the ISIS proton beam, a pion-momentum-selection dipole, a supercon-  
195 ducting solenoid to focus and transport the particles to a second dipole that is used to select the  
196 muon-beam momentum and a transport channel composed of a further two quadrupole triplets. The  
197 GEANT4 simulation within MAUS starts 1 m downstream of the second beamline dipole magnet  
198 (D2). GEANT4 bindings are encoded in the Simulation module. GEANT4 groups particles by run,  
199 event and track. A GEANT4 run maps to a MICE spill; a GEANT4 event maps to a single inbound  
200 particle from the beamline; and a GEANT4 track corresponds to a single particle in the experiment.

201 GEANT provides a variety of reference physics processes to model the interactions of particles  
202 with matter. The default process in MAUS is “*QGSP\_BERT*” which causes GEANT4 to model  
203 hadron interactions using a Bertini cascade model up to 10 GeV/c. MAUS provides methods  
204 to setup the GEANT4 physical processes which allows the user to control processes with data-  
205 cards. Routines are also provided to interface the internal geometry representation in MAUS with  
206 GEANT4 descriptions. Finally, MAUS provides routines to extract particle data from the GEANT  
207 tracks at user-defined locations.

### 208 3.3 Geometry

209 MAUS uses an online Configurations Database to store all of its geometries. These geometries  
210 have been extracted from CAD drawings which are updated based on the most recent surveys and  
211 technical drawings available. The CAD drawings are translated to a geometry-specific subset of  
212 XML, the Geometry Description Markup Language (GDML) [26], prior to being recorded in the  
213 configuration database through the use of the FastRAD [27] commercial software package.

214 The GDML formatted description contains the beam-line elements and the positions of the de-  
215 tector survey points. Beam-line elements are described using tessellated solids to define the shapes  
216 of the physical volumes. The detectors themselves are described using an independently generated  
217 set of GDML files using GEANT4 standard volumes. An additional XML file is appended to the  
218 geometry description that assigns magnetic fields and associates the detectors to their locations in  
219 the GDML files. This file is initially written by the geometry maintainers and formatted to contain  
220 run-specific information during download.

*Capitalization?*

221 The GDML format has a number of benefits. The files can be read via a number of libraries in  
222 GEANT4 and ROOT for the purpose of independent validation. Because it is a subset of XML, the  
223 data contained in the GDML files are readily accessible through the application of the libxml2 [28]  
224 python extension. The GDML files are in turn translated into the MAUS readable geometry files  
225 either by directly accessing the data using the python extension (which is the method applied  
226 to the detector objects) or through the use of EXtensible Stylesheet Language Transformations  
227 (XSLT) [29].

*i.e.?*

*?*

### 228 3.4 Tracking, field maps and beam optics

229 MAUS tracking is performed using GEANT4. By default, MAUS uses 4<sup>th</sup> order Runge-Kutta  
230 (RK4) for tracking, although other routines are available. RK4 has been shown to have very good  
231 precision relative to the MICE detector resolutions, even for step sizes of several cm.

232 Magnetic field maps are implemented in a series of overlapping regions. On each tracking  
233 step, MAUS iterates over the list of fields, transforms to the local coordinate system of the field  
234 map, and calculates the field. The field values are transformed back into the global coordinate  
235 system, summed, and passed to GEANT4.

236 Numerous field types have been implemented within the MAUS framework. Solenoid fields  
237 can be calculated numerically from cylindrically symmetric 2D field maps, by taking derivatives  
238 of an on-axis solenoidal field or by using the sum of fields from a set of cylindrical current sheets.  
239 Multipole fields can be calculated from a 3D field map, or by taking derivatives from the usual  
240 multipole expansion formulae. Linear, quadratic and cubic interpolation routines have been imple-  
241 mented for field maps. Pillbox fields can be calculated by using the Bessel functions appropriate  
242 for a TM010 cavity or by reading a cylindrically symmetric field map.

243 Matrix transport routines for propagating particles and beams through these field maps have  
244 been implemented. Transport matrices are calculated by taking the numerical derivative of the  
245 tracking output and can be used to transport beam ellipses and single particles.

246 The accelerator modeling routines in MAUS have been validated against ICOOL and G4Beamline  
247 and have been used to model a number of beamlines and rings, including a "neutrino factory" front-  
248 end.

*(in order  
used)*

### 249 3.5 Detector response and digitization

250 The modelling of the detector response and electronics enables MAUS to provide data to test re-  
251 construction algorithms and estimate the uncertainties introduced by a detector and its readout.

252 The interaction of particles in material is modeled using GEANT4. A "sensitive detector" class  
253 for each detector processes GEANT4 hits in active detector volumes and stores hit information  
254 such as the volume that was hit, the energy deposited and the time of the hit. Each detector's  
255 digitization routine then simulates the electronics' response to these hits, modeling processes such  
256 as the photo-electron yield from a scintillator bar, attenuation in light guides and the pulse shape  
257 in the electronics. The data structure of the outputs from the digitizers are designed to match the  
258 output from the unpacking of real data from the DAQ.

259 **4. Reconstruction**

260 The reconstruction chain takes as its input either digitized hits from the MC or DAQ digits from  
261 real data. Regardless, the detector reconstruction algorithms, by requirement and design, operate  
262 the same way on both MC and real data.

263 **4.1 Time of flight**

264 There are three time-of-flight detectors in MICE which serve to distinguish particle type. The  
265 detectors are made of plastic scintillator and in each station there are orthogonal  $x$  and  $y$  planes  
266 with 7 or 10 slabs in each plane.

267 Each GEANT4 hit in the TOF is associated with a physical scintillator slab. The energy de-  
268 posited by a hit is first converted to units of photo-electrons. The photo-electron yield from a hit  
269 accounts for the light attenuation corresponding to the distance of the hit from the photomultiplier  
270 tube (PMT) and is then smeared by the photo-electron resolution. The yields from all hits in a  
271 given slab are then summed and the resultant yield is converted to ADC counts.

272 The time of the hit in the slab is propagated to the PMTs at either end of the slab. The propa-  
273 gated time is then smeared by the PMT time resolution and converted to TDC counts. Calibration  
274 corrections based on real data are then added to the TDC values so that, at the reconstruction stage,  
275 they can be corrected just as is done with real data.

276 The reconstruction proceeds in two main steps. First, the slab-hit-reconstruction takes indi-  
277 vidual PMT digits and associates them to reconstruct the hit in the slab. If there are multiple hits  
278 associated with a PMT, the hit which is earliest in time is taken to be the real hit. Then, if both  
279 PMTs on a slab have fired, the slab is considered to have a valid hit. The TDC values are converted  
280 to time and the hit time and charge associated with the slab hit are taken to be the average of the two  
281 PMT times and charges respectively. In addition, the product of the PMT charges is also calculated  
282 and stored. Secondly, individual slab hits are used to form space-points. A space point in the TOF  
283 is a combination of  $x$  and  $y$  slab hits. All combinations of  $x$  and  $y$  slab hits in a given station are  
284 treated as space point candidates. Calibration corrections, stored in the Configurations Database,  
285 are applied to these hit times and if the reconstructed space-point is consistent with the resolution  
286 of the detector, the combination is said to be a valid space point. The TOF has been shown to  
287 provide good time resolutions at the 60 ps level [5].

288 **4.2 Scintillating fiber trackers**

289 The scintillating fiber trackers are the central piece of the reconstruction. As mentioned in Sec-  
290 tion 1.1, there are two trackers, one upstream and the other downstream of an absorber, situated  
291 within solenoidal magnetic fields. The trackers measure the emittance before and after particles  
292 pass through the absorber.

293 The tracker software algorithms and performance are described in detail in [30]. Digits are  
294 the most basic unit fed into the main reconstruction module, each digit representing a signal from  
295 one channel. Digits from adjacent channels are assumed to come from the same particle and are  
296 grouped to form clusters. Clusters from channels which intersect each other, in at least two planes  
297 from the same station, are used to form space-points, giving  $x$  and  $y$  positions where a particle  
298 intersected a station. Once space-points have been found, they are associated with individual tracks

299 through pattern recognition (PR), giving straight or helical PR tracks. These tracks, and the space-  
300 points associated with them, are then sent to the final track fit. To avoid biases that may come from  
301 space-point reconstruction, the Kalman filter uses only reconstructed clusters as input.

How so?

### 302 4.3 KL calorimeter

303 Hit-level reconstruction of the KL is implemented in MAUS. Individual PMT hits are unpacked  
304 from the DAQ or simulated from MC and the reconstruction associates them to identify the slabs  
305 that were hit and calculates the charge and charge-product corresponding to each slab hit. The KL  
306 has been used successfully to estimate the pion contamination in the MICE muon beamline [31].

### 307 4.4 Electron-muon ranger

308 Hit-level reconstruction of the EMR is implemented in MAUS. The integrated ADC count and  
309 time over threshold are calculated for each bar that was hit. The EMR reconstructs a wide range of  
310 variables that can be used for particle identification and momentum reconstruction. The software  
311 and performance of the detector are described in detail in [32].

EMR

### 312 4.5 Cherenkov

313 The CKOV reconstruction takes the raw flash-ADC data, subtracts pedestals, calculates the charge  
314 and applies calibrations to determine the photo-electron yield.

### 315 4.6 Global reconstruction

316 The aim of the Global Reconstruction is to take the reconstructed outputs from individual detectors  
317 and ~~to~~ tie them together to form a global track. A likelihood for each particle hypothesis is also  
318 calculated.

1D

#### 319 4.6.1 Global Track Matching

320 Global track matching is performed by collating particle hits (TOFs 0, 1 and 2, KL and Ckov) and  
321 tracks (Trackers and EMR) from each detector using their individual reconstruction and combining  
322 them using a RK4 method to propagate particles between these detectors. The tracking is performed  
323 outwards from the cooling channel, the upstream tracker through TOF0; and downstream tracker  
324 through EMR. It is also available as a commissioning tool providing through-going tracks from  
325 TOF1 to EMR, in the absence of magnetic fields. Track points are matched to form tracks using  
326 a RK4 method. Initially this is done independently for the upstream and downstream (i.e., either  
327 side of the absorber) sections of the beamline. As the trackers provide the most accurate position  
328 reconstruction, they are used as starting points for track matching, propagating hits outwards into  
329 the other detectors and then comparing the propagated position to the measured hit in the detector.  
330 The acceptance criterion for a hit belonging to a track is an agreement within the detector's solution  
331 with an additional allowance for multiple scattering. Track matching is currently performed for all  
332 TOFs, KL and EMR.

333 The RK4 propagation requires the mass and charge of the particle to be known. Hence, it is  
334 necessary to perform track matching for all particle types (muons, pions, and electrons). Tracks for  
335 all possible PID hypotheses are then passed to the Global PID algorithms.

A and B?  
comes?  
? mc? Why not  
helical?

H

No eqn's? Why?

336 **4.6.2 Global PID**

337 **DR note: This is not used/tested in MAUS production – should this stay? Comments?**

Why not?

338 Global particle identification in MICE typically requires the combination of several detectors.  
339 The time-of-flight between TOF detectors can be used to calculate velocity, which is compared  
340 with the momentum measured in the trackers to identify the particle type. For all but very low  $p_t$   
341 events, charge can be determined from the direction of helical motion in the trackers. Additional  
342 information can be obtained from the CKOV, KL and EMR detectors. The global particle identi-  
343 fication framework is designed to tie this disparate information into a set of hypotheses of particle  
344 types, with an estimate of the likelihood of each hypothesis.

345 The Global PID in MAUS uses a log-likelihood method to identify the particle species of a  
346 global track. It is based upon a framework of PID variables. Simulated tracks are used to produce  
347 probability density functions (PDFs) of the PID variables. These are then compared with the PID  
348 variables for tracks in real data to obtain a set of likelihoods for the PIDs of the track.

349 The input to the Global PID is several potential tracks from global track matching. Each  
350 of these tracks was matched for a given particle hypothesis. The Global PID then takes each track  
351 and determines the most likely PID following a series of steps:

- 352 1. Each track is copied into an intermediate track; *why not p? ..*
- 353 2. For each potential PID hypothesis  $x$ , the log-likelihood is calculated using the PID variables;
- 354 3. The track is assigned an object containing the log-likelihood for each hypothesis;
- 355 4. From the log-likelihoods, the confidence level, C.L., for a track having a PID  $x$  is calculated  
356 and the PID is set to the hypothesis with the best C.L. *What if ambig?*

357 **4.7 Online reconstruction**

358 During data taking, it is essential to visualize a detector's performance and have diagnostic tools  
359 to identify and debug unexpected behavior. This is accomplished through summary histograms  
360 of high and low-level quantities from detectors. The implementation is through a custom multi-  
361 threaded application based on a producer-consumer pattern with thread-safe FIFO buffers. Raw  
362 data produced by the DAQ is streamed through a network and consumed by individual detector  
363 mappers described in section 3. The reconstructed outputs produced by the mappers are in turn  
364 consumed by the reducers. The mappers and reducers are distributed among the threads to bal-  
365 ance the load. Finally, outputs from the reducers are written as histogram images. Though the  
366 framework for the online reconstruction is based on parallelized processing of spills, the recon-  
367 struction modules are the same as those used for offline processing. A lightweight tool based on  
368 Django [33] provides live web-based visualization of the histogram images as and when they are  
369 created.

Summary ..

370 **Acknowledgments**

371 The work described here was made possible by grants from Department of Energy and National  
372 Science Foundation (USA), the Instituto Nazionale di Fisica Nucleare (Italy), the Science and

373 Technology Facilities Council (UK), the European Community under the European Commission  
374 Framework Programme 7 (AIDA project, grant agreement no. 262025, TIARA project, grant  
375 agreement no. 261905, and EuCARD), the Japan Society for the Promotion of Science and the  
376 Swiss National Science Foundation, in the framework of the SCOPES programme. We gratefully  
377 acknowledge all sources of support. We are grateful to the support given to us by the staff of  
378 the STFC Rutherford Appleton and Daresbury Laboratories. We acknowledge the use of Grid  
379 computing resources deployed and operated by GridPP [?] in the UK.

380 **References**

- 381 [1] The IDS-NF collaboration. International design study for the neutrino factory: Interim design report,  
382 2011. IDS-NF-020, [www.ids-nf.org/wiki/FrontPage/Documentation](http://www.ids-nf.org/wiki/FrontPage/Documentation).
- 383 [2] Steve Geer. Muon Colliders and Neutrino Factories. *Annual Review of Nuclear and Particle Science*,  
384 59:345 – 367, 2009.
- 385 [3] M. Bogomilov et al. The MICE Muon Beam on ISIS and the beam-line instrumentation of the Muon  
386 Ionization Cooling Experiment. *JINST*, 7:P05009, 2012.
- 387 [4] D. Adams et al. Characterisation of the muon beams for the Muon Ionisation Cooling Experiment.  
388 *Eur. Phys. J.*, C73(10):2582, 2013.
- 389 [5] R. Bertoni et al. The design and commissioning of the MICE upstream time-of-flight system. *Nucl.*  
390 *Instrum. Meth.*, A615:14 – 26, 2010.
- 391 [6] L. Cremaldi, D. A. Sanders, P. Sonnek, D. J. Summers, and J. Reidy, Jr. A cherenkov radiation  
392 detector with high density aerogels. *IEEE Trans. Nucl. Sci.*, 56:1475–1478, 2009.
- 393 [7] M. Ellis, P.R. Hobson, P. Kyberd, J.J. Nebrensky, A. Bross, et al. The Design, construction and  
394 performance of the MICE scintillating fibre trackers. *Nucl. Instrum. Meth.*, A659:136–153, 2011.
- 395 [8] R. Asfandiyarov et al. The design and construction of the MICE Electron-Muon Ranger. *JINST*,  
396 11(10):T10007, 2016.
- 397 [9] C. Tunnell and C Rogers. MAUS: MICE Analysis User Software. In *Proc. 2011 International*  
398 *Particle Accelerator Conference, San Sebastian*, 2011. MOPZ013.
- 399 [10] <http://scons.org/>.
- 400 [11] <https://bazaar.canonical.com>.
- 401 [12] <http://launchpad.net>.
- 402 [13] R. Brun and F. Rademakers. Root - an object oriented data analysis framework. *Nucl. Instrum. Meth.*,  
403 A389:81 – 86, 1997. <http://root.cern.ch/>.
- 404 [14] S Agostinelli et al. Geant4 - a simulation toolkit. *Nucl. Instrum. Meth.*, A506:250 – 303, 2003.
- 405 [15] <https://scientificlinux.org>.
- 406 [16] <https://centos.org>.
- 407 [17] <https://getfedora.org>.
- 408 [18] <https://ubuntu.com>.
- 409 [19] <http://www.swig.org>.

When expect  
^ a final  
report?

- 410 [20] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *Proceedings*  
411 *of OSDI04*, 2004. <http://research.google.com/archive/mapreduce.html>.
- 412 [21] <http://json.org>.
- 413 [22] <https://github.com/open-source-parsers/jsoncpp>.
- 414 [23] <http://jenkins-ci.org>.
- 415 [24] Thomas J. Roberts and Daniel M. Kaplan. G4BeamLine programme for matter dominated beamlines.  
416 In *Proc. 2007 Particle Accelerator Conference, Albuquerque*, 2007. THPAN103.
- 417 [25] R. C. Fernow. Icool: A simulation code for ionization cooling of muon beams. In *Proc. 1999 Particle*  
418 *Accelerator Conference, New York*, 1999.
- 419 [26] J. McCormick R. Chytracek. Geometry description markup language for physics simulation and  
420 analysis applications. *IEEE Trans. Nucl. Sci.*, 53:2892–2896, 2006.
- 421 [27] <http://fastrad.net>.
- 422 [28] <https://xmlsoft.org>.
- 423 [29] <https://www.w3.org/standards/xml/transformation>.
- 424 [30] A. Dobbs, C. Hunt, K. Long, E. Santos, M.A. Uchida, P. Kyberd, C. Heidt, S. Blot, and E. Overton.  
425 The reconstruction software for the mice scintillating fibre trackers. *JINST*, 11(12):T12001, 2016.
- 426 [31] D. Adams et al. Pion Contamination in the MICE Muon Beam. *JINST*, 11(03):P03001, 2016.
- 427 [32] D. Adams et al. Electron-muon ranger: performance in the MICE muon beam. *JINST*,  
428 10(12):P12012, 2015.
- 429 [33] <https://www.djangoproject.com/>.

*Use std spellings*