FAIRROOT TUTORIAL
Overview

- What is FairRoot?
- FairRoot Building Blocks
- Workflow
  - Simulation
  - Reconstruction/Analysis
- Geometry definition
- Parameter Manager
What is FairRoot????

- Simulation-, Reconstruction-, and Analysis-Framework
  - Started as 2 person project for CBM experiment at GSI
  - Now used by many experiments, also outside of GSI
- Decouple the detector simulation description from the simulation engine
  - Use different simulation engines (Geant3, Geant4, ...) with the same user code (Virtual Monte Carlo)
- Reuse existing software and tools
- Use standards as much as possible
- Code should run on all platforms
- The Framework should be
  - Easy to install
  - Easy to use
  - Should allow fast development cycles
  - Flexible to easily change experimental setup
  - Extensible for new developments
Simulation or reconstruction run is completely defined in a ROOT macro which is executed by ROOT.

Definition can also be done using Python (thanks to Thomas Ruf).

Main event loop steered by RunManager:
- FairRunSim for simulation runs
- FairRunAna for reconstruction or analysis runs
- FairRunOnline for online analysis

Different specialized run managers should be merged.
Run is defined by adding pieces to the RunManager

// ----- Create simulation run
FairRunSim* run = new FairRunSim();
run->SetName("TGeant4");
run->SetOutputFile("myoutput.root");
// ----- Create media
run->SetMaterials("media.geo");
// ----- Create geometry
FairModule* cave = new FairCave("CAVE");
cave->SetGeometryFileName("cave_vacuum.geo");
run->AddModule(cave);
FairDetector* tutdet = new FairTutorialDet1("TUTDET", kTRUE);
tutdet->SetGeometryFileName("double_sector.geo");
run->AddModule(tutdet);

// ----- Create Generator
FairBoxGenerator* boxGen = new FairBoxGenerator(211, 1);
boxGen->SetThetaRange(0., 8.);
boxGen->SetPRange(2., 2.5);
boxGen->SetPhiRange(0., 360.);
FairPrimaryGenerator* primGen = new FairPrimaryGenerator();
primGen->AddGenerator(boxGen);
run->SetGenerator(primGen);
// ----- Initialize simulation run
run->Init();
// ----- Start
run->Run(1000);
Framework classes which are used as they are

- **FairRunSim**
  - Define the experimental setup by calling member functions

- **FairPrimaryGenerator**
  - Define the primary beam conditions by calling member functions

- **FairMCApplication**
  - Interface between FairRoot and the MC engine
  - only used internally

Special Simulation Run Modes

- **FairRadLenManager**
- **FairRadMapManager**
- **FairRadGridManager**
Building Blocks (Simulation II)

- Classes which have to be implemented by the user
  - **FairModule**
    - Defines a geometry element which does not produce any MC points (e.g. a magnet)
    - May consist of many volumes
  - **FairDetector**
    - Defines a geometry element which has active volumes (e.g. a detector)
    - Derived from FairModule
    - Some Functions defined which are called from the MCApplication
      - When some specific points in the simulation are reached (e.g. begin of an event)
      - When a track is in an active volume (store MC information for later usage)
  - **FairGenerator**
    - Generate particles for one event and puts the particles on the Stack
  - **FairGenericStack**
    - Stack which keeps all the particles produced in one event
    - Primary particles from event generators
    - Particles produced by the MC program during the transport
    - Filter particles which are written to the output
  - **FairField**
    - Return field values for a given space point
FairRadLenManager for Radiation length info

Example: Contributions of different Functional parts of the PANDA MVD to the overall material budget

- (a) Fractional radiation length $[\times X_0]$ vs. $\theta$ [°]
- (b) Integrated distribution until $\theta = 140°$
  - Electronics: 28.5%
  - Support: 13.6%
  - Cooling: 37.7%
  - Cabling: 5.0%
What energy dose will be accumulated during a certain time of operation?

Create all physical volumes with correct material assignment

Run the simulation engine

FairRadMapManager will sum up every deposited energy in each volume in the geometry
Determine the particle fluency through a certain boundary (surface) and deduce a map.
Framework classes which are used as are

- **FairRunAna**
  - Define the tasks and the calling sequence

Classes which have to be implemented by the user

- **FairTask**
  - Register output data level
  - Connect input data level
  - Execute data from input data level and write output data level
  - E.g.: create from MC points the expected detector raw information
Building Blocks (General)

- **IO Manager**
  - **FairRootManager**
    - Organizes IO
    - New branches are registered from user code
    - Provide access to registered branches on request from user code

- **Parameter Manager**
  - **FairRuntimeDb**
    - Bad name
    - Interface between data storage backend and user code
    - Provide always the valid parameters
    - Update of parameters is triggered by the RunManager when unique RunId changes
FairRoot Simulation: Workflow

Create Instances
// ----- Create simulation run ----------------------------------------
FairRunSim* run = new FairRunSim();
FairRuntimeDb* rtdb = run->GetRuntimeDb();
FairRoot Simulation: Workflow

- Define Output File
- Define MC Engine
- Define Parameter I/O
- Create IO Manager

Steering Macro → Run Manager → Parameter Manager → IO Manager

On construction of Run Manager
Code

```c
TString parFile = "params.root"
Bool_t kParameterMerged = kTRUE;
FairParRootFileIo* parOut = new FairParRootFileIo(kParameterMerged);
parOut->open(parFile.Data());
rtdb->setOutput(parOut);
rtdb->saveOutput();
rtdb->print();

TString mcEngine = "TGeant3"; // or "TGeant4"
TString outFile = "mc.test.root"
run->SetName(mcEngine); // Transport engine
run->SetOutputFile(outFile); // Output file
```
FairRoot Simulation: Workflow

Create Modules/Detectors
Define Media file
Create Generators
Create PrimaryGenerator
Create Field Class

Add everything to the Run Manager
Hand responsibility to Run Manager

Steering Macro
Parameter File(s)
Output File

Run Manager
Parameter Manager
IO Manager
Connect File
```cpp
// ----- Create media --------------------------------------------------------
run->SetMaterials("media.geo"); // Materials
// ---------------------------------------------------------------------------

// ----- Create geometry -----------------------------------------------------
FairModule* cave = new FairCave("CAVE");
cave->SetGeometryFileName("cave_vacuum.geo");
run->AddModule(cave);

FairDetector* tutdet = new FairTutorialDet1("TUTDET", kTRUE);
tutdet->SetGeometryFileName("double_sector.geo");
run->AddModule(tutdet);
// ---------------------------------------------------------------------------

// ----- Create PrimaryGenerator --------------------------------------------
FairPrimaryGenerator* primGen = new FairPrimaryGenerator();
FairBoxGenerator* boxGen = new FairBoxGenerator(partPdgC[chosenPart], 1);

boxGen->SetThetaRange (0., 5.);
boxGen->SetPRange (2., 2.01);
boxGen->SetPhiRange (0., 360.);
boxGen->SetDebug(kTRUE);

primGen->AddGenerator(boxGen);
run->SetGenerator(primGen);
```
FairRoot Simulation: Workflow

- Steering Macro
- Parameter File(s)
- Output File
- Media File

Init

Run Manager

Parameter Manager

Field

Module 1

Detector 1

PrimaryGenerator

Generator 1

Generator 2

Generator N
// ----- Initialize simulation run ------------------------------------
run->Init();
// --------------------------------------------------------------------
FairRoot Simulation: Workflow

- Steering Macro
- Parameter File(s)
- Output File
- Media File
- Field
- Module 1
- Detector 1

Create FairMCApplication

PrimaryGenerator
- Generator 1
- Generator 2
- Generator N

Run Manager
Parameter Manager
Define PrimaryGenerator as MC generator
FairRoot Simulation: Workflow

Register some parameter
Containers to store global info
(list of modules, gGeoManager, ...)

Steering Macro
Parameter File(s)
Output File

Field
Module 1
Detector 1

FairMCAppliation

Run Manager
Parameter Manager
IO Manager

PrimaryGenerator

Generator 1
Generator 2
...
Generator N

Florian Uhlig, R3BRoot Development Workshop
FairRoot Simulation: Workflow

Add field to FairMCApplication

- Steering Macro
- Parameter File(s)
- Output File

Run Manager

Parameter Manager

IO Manager

FairMCApplication

PrimaryGenerator

- Generator 1
- Generator 2
- ...
- Generator N

Field
Read MC configuration from $VMCWORKDIR/gconfig
Execute configuration macros

Florian Uhlig, R3BRoot Development Workshop
Create Instance of MC engine
Create Stack
Configure MC engine

Steering Macro
Parameter File(s)
Output File

Run Manager
Parameter Manager
IO Manager

Module 1
Detector 1

FairMCApplication

Stack

PrimaryGenerator
Generator 1
Generator 2
Generator N

TGeant4

Field

Florian Uhlig, R3BRoot Development Workshop
Run InitMC function of FairMCApplication
FairMCApplication calls Init function of MC engine
TGeant4 calls FairMCApplication to
- Register user defined particles
- Construct the geometry

Steering Macro
Parameter File(s)
Output File

Run Manager
Parameter Manager
IO Manager

PrimaryGenerator
Generator 1
Generator 2
Generator N

FairMCApplication
Module 1
Detector 1

TGeant4 calls FairMCApplication to
- Register user defined particles
- Construct the geometry
FairMCApplication call ConstructGeometry function of all registered modules/detectors
Volumes are added to gGeoManager
Close geometry in gGeoManager
FairMCApplication call Initialize and Register function of all registered detectors

Florian Uhlig, R3BRoot Development Workshop
Detectors register their data containers

Florian Uhlig, R3BRoot Development Workshop
FairRoot Simulation: Workflow

Everything is initialized
- Run Manager
- FairMCApplication
- MC engine

Steering Macro
Parameter File(s)
Output File

Run Manager
Parameter Manager
IO Manager

FairMCApplication
Stack
PrimaryGenerator
- Generator 1
- Generator 2
- Generator N

TGeant4
Detector 1
Field
The execution of the events is completely steered by the MC engine, which calls FairRoot via several hooks which are implemented in FairMCApplication.
// ----- Start run ----------------------------------------------------------
run->Run(nEvents);
run->CreateGeometryFile("geofile_full.root");
// ------------------------------------------------------------------------
VirtualMC Hooks I

- Hooks are defined in TVirtualMCApplication
- BeginEvent(), BeginPrimary(), FinishPrimary(), PreTrack(), PostTrack()
  - Call functions if implemented in the detectors
- GeneratePrimaries()
  - Call PrimaryEventGenerator to fill the stack
  - Loop over registered generators
- FinishEvent()
  - Filter the stack (remove tracks which doesn’t pass the cuts)
  - Call functions if implemented in the detectors
  - Write data to output
VirtualMC Hooks II

- **Stepping()**
  - Called after each step from MC engine
  - Check if step is an active volume
    - If false return to MC engine for the next step
    - If true call the ProcessHit function of the respective detector
      - To be fast during initialization a map was filled which connects volume ids with pointers to the correct function
After all events are executed the output is written, the files are closed and other cleanup done.
Create Instances
FairRoot Reconstruction: Workflow

Define In/Out Files
Define Parameter I/O

Steering Macro
Run Manager
Parameter Manager
Define Tasks

- Steering Macro
- Parameter File(s)
- Input File(s)
- Output File
- Run Manager
- Parameter Manager
FairRoot Reconstruction: Workflow

- Steering Macro
- Parameter File(s)
- Input File(s) Output File
- Run Manager
- Parameter Manager

Event Loop
- Task 1
- Task 2
- Task N
FairRoot Reconstruction: Workflow

Event Loop

Task 1
Task 2
...
Task N

Run Manager

Parameter Manager

Initialize

Parameter File(s)
Input File(s) Output File
FairRoot Reconstruction: Workflow

- **Input File(s)**
- **Parameter File(s)**
- **Database**
- **Output File**

Run Manager

Parameter Manager

Event Loop

- Task 1
- Task 2
- Task N

Read Parameters
**FairRoot Reconstruction: Workflow**

- Parameter File(s)
- Database

- Input File(s)
- Output File

- Run Manager
- Parameter Manager
- TGeo Manager

- Event Loop
  - Task 1
  - Task 2
  - Task N

  "Initialize from Parameter input"
FairRoot Reconstruction: Workflow

**Input File(s)**
- Parameter File(s)
- Database
- Output File

**Output File**

**Parameter Manager**

**I/O Manager**

**TGeo Manager**

**Run Manager**

**Event Loop**
- Task 1
- Task 2
- Task N

**Initialize**
FairRoot Reconstruction: Workflow

- FairRoot
  - Run Manager
  - Parameter Manager
  - I/O Manager
  - TGeo Manager

Input File(s)
- Parameter File(s)
- Database

Connect
- Open
- Read

Event Loop
- Task 1
- Task 2
- ...
- Task N

Florian Uhlig, R3BRoot Development Workshop 28.07.15
FairRoot Reconstruction: Workflow

Event Loop

<table>
<thead>
<tr>
<th>Task 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 2</td>
</tr>
<tr>
<td>Task N</td>
</tr>
</tbody>
</table>

Initialize

FairRoot

Run Manager
Parameter Manager
I/O Manager
TGeo Manager

Parameter File(s)
Database
Input File(s)
Output File

Florian Uhlig, R3BRoot Development Workshop   28.07.15
FairRoot Reconstruction: Workflow

```
Input File(s)

Parameter File(s)
Database

Output File

Run Manager
Parameter Manager
I/O Manager
TGeo Manager

Event Loop
   Task 1
   Task 2
   ...
   Task N

Read
Connect
```

Florian Uhlig, R3BRoot Development Workshop  28.07.15
FairRoot Reconstruction: Workflow

**Inputs**
- Parameter File(s)
- Database
- Input File(s)
- Output File

**Execution Components**
- Run Manager
- Parameter Manager
- I/O Manager
- TGeo Manager

**Event Loop**
- Task 1
- Task 2
- Task N
FairRoot Reconstruction: Workflow

Florian Uhlig, R3BRoot Development Workshop 28.07.15
FairRoot Reconstruction: Workflow

Flowchart showing the workflow of FairRoot:
- **Run Manager**
- **Parameter Manager**
- **I/O Manager**
- **TGeo Manager**

Input File(s) and Output File are connected to the I/O Manager.
Parameter File(s) and Database are connected to the Parameter Manager.

The Event Loop consists of:
- Task 1
- Task 2
- Task N

The Event Loop includes:
- Read Data
- Do something
- Write Data

Write Data To File is shown below the I/O Manager.
FairRoot Reconstruction: Workflow

**Input File(s)**
- Parameter File(s)
- Database
- Event Loop

**Output File**
- Write Data
- Close File

**Task Loop**
- Task 1
- Task 2
- Task N

**Event Loop**
- Write/Update Parameters
- Close File

**FairRoot**
- Run Manager
- Parameter Manager
- I/O Manager
- TGeo Manager

**Finish**
Fair Parameter Manager

- Single Tool to access parameters
- Parameters are stored in parameter containers
- Parameter Manager has different storage backends
  - ASCII files
  - ROOT files
  - FairDB (optional)
- FairDB can write/read parameters from various databases
  - MySQL
  - Postgresql
  - Oracle
  - sqlite
Parameter container

---

### Generic parameter container

- **name**
- **type**: Int_t, Float_t, Double_t, Char_t, Text_t, UChar_t, class type
- **value(s)**

---

Any class derived from TObject decoded in the analysis interface by ROOT streamer (not in ascii)

---

```
# Format:
# parameter_name:parameter_type parameter_value
# PndTutPar

[TUTParDefault]
top_pitch: Double_t 0.01
top_anchor: Double_t -3.0
nr_fe_top: Int_t 10
fe_Type: Text_t APV25
```

---

Florian Uhlig, R3BRoot Development Workshop  28.07.15
User has to implement two function in class derived from FairGenericParSet

```c
void FairTutPar::putParams(FairParamList* list)
{
  if(!list) { return; }
  list->add("top_pitch", fTopPitch);
  list->add("top_anchor", fTopAnchor);
  list->add("nr_fe_top", fTopNrFE);
  list->add("fe_Type", (Text_t*) &fFeType);
}

Bool_t FairTutPar::getParams(FairParamList* list)
{
  if (!list) { return kFALSE; }
  if (!list->fill("top_pitch", &fTopPitch)) { return kFALSE; }
  if (!list->fill("top_anchor", &fTopAnchor)) { return kFALSE; }
  if (!list->fill("nr_fe_top", &fTopNrFE)) { return kFALSE; }
  Text_t feName[80];
  if (!list->fill("fe_Type", feName,80)) { return kFALSE; }
  fFeType = feName;
  return kTRUE;
}
```
The input and output of the parameter manager is defined in the macro

```c
// ----- Parameter database --------------------------------------------
FairRuntimeDb* rtdb = fRun->GetRuntimeDb();

FairParRootFileIo* parInput1 = new FairParRootFileIo();
parInput1->open("par.root","READ"); //RECREATE etc. like in TFile from root
rtdb->setFirstInput(parInput1);

FairParAsciiFileIo* parInput2 = new FairParAsciiFileIo();
parInput2->open("ascii-example.par","in");
rtdb->setSecondInput(parInput2);

Bool_t kParameterMerged=kTRUE;
FairParRootFileIo* parOutput = new FairParRootFileIo(kParameterMerged);
parOutput->open("par_merged.root","RECREATE"); //RECREATE etc. like in TFile from root
rtdb->setOutput(parOutput);
rtdb->saveOutput();
```
Workflow

- Define in your task which parameter container you need

```cpp
void CbmTrdDigitizer::SetParContainers()
{
    fDigiPar = static_cast<CbmTrdDigiPar*>(FairRunAna::Instance()->GetRuntimeDb()->getContainer("CbmTrdDigiPar"));
}
```

- FairRoot will take care that the containers are properly filled, when calling the Init or ReInit functions of FairTask
- The ReInit is called when run conditions change (runID)
- Use parameter container class (fDigiPar) in the task to access the actual valid parameters