HepMC and Root I/O

LPNHE Paris

Lars Sonnenschein

- HepMC Class Dictionaries
- Root GenReflex I/O
- Root Cint I/O
- Examples/Test executables
- Performances
- Conclusions

This work has been supported by a Marie Curie Early Stage Research Training Fellowship of the European Community's Sixth Framework Programme under contract number MRTN-CT-2006-035606, by the *Commissariat à l'Energie Atomique* and CNRS/*Institut National de Physique Nucléaire et de Physique des Particules*, France
and by the HEPtools EU Marie Curie Research Training Network under the contract number MRTN-CT-2006-035505.

HepMC Class Dictionaries

Two approaches

- GenReflex Dictionary:
  a) `genreflex` reads a C++ class header file,
  b) and reads a `xml` class selection file
  c) provides the class dictionary source code needed for object I/O based on Root
  d) example usage:

```
genreflex Classes_Rflx.h -s selection.xml \
-o Classes_Rflx.cc \
--gccxmlpath $GCCXMLPATH/bin -I$HEPMCINCPATH
```

- Cint Dictionary:
  a) `rootcint` reads a C++ class header file,
  b) and reads the classes to be added to Root via a shared library from a pre-processor directive header file (`LinkDef.h`)
  c) provides class dictionary source code needed for object I/O based on Root
  d) example usage:

```
rootcint -f Classes_Cint.cc -c -p -I$HEPMCINCPATH \
$ROOTCPPFLAGS Classes_Cint.h LinkDef.h
```

Example code in a Nutshell

- For the time being you can get the installation directory `HepMCRootIO` from `/afs/cern.ch/user/s/sonne/public/`

- It is based on autotools to simplify cross platform portability

- Installation steps:
  a) Root needs to be available, `$ROOTSYS` path has to be set
  b) HepMC needs to be available `$HEPMCPATH` has either to be set as environment variable or passed as an option to `configure`
  c) Optionally `$GCCXMLPATH` (for `genreflex`) can be specified (again, either as environment variable or passed as an option to `configure`)
  - `autoreconf -iv`
  - `./configure --prefix=/path_to_install_dir [HEPMCPATH=/...] [GCCXMLPATH=/...]`
  - `make`
  - `make install`

Test executables

- All test executables are produced in the `src` subdirectory and together with shared libraries in the specified installation directory (`bin` and `lib` subdirectories)

- Based on GenReflex:
  ◦ Executable: `writeEvtKey.x` based on source file `writeEvtKey.cc`
  Writes a simple `GenEvent` with one `GenParticle` and its `GenVertex` via the `TFile WriteObject()` function to the output file `writeEvtKey.root`

  ◦ Executable: `readEvtKey.x` based on source file `readEvtKey.cc`
  Read in a file of `GenEvent`'s stored in `genreflex` dictionary format (file name to be specified at command line) and print out each event.

  ◦ Executable: `testRootIO.x` based on source file `testRootIO.cc`
  Produce a simple `GenEvent` with one `GenParticle` and its `GenVertex`, store it in HepMC ASCII format and in `genreflex` dictionary format.
  Read the file in `genreflex` dictionary format back in and print each event.

Snippets of code: writeEvtKey.cc

```
gSystem->Load("libRflxHepMCdict");
ROOT::Cintex::Cintex::Enable();

HepMC::GenEvent* evt = new HepMC::GenEvent;
HepMC::GenParticle* part =
  new HepMC::GenParticle(HepMC::FourVector(10,20,30,40), 99);

HepMC::GenVertex* vtx = new HepMC::GenVertex();

vtx->add_particle_out(part);
evt->add_vertex( vtx );

std::string stevt = "Event_1";
const char* chevt = stevt.c_str();

TFile* fo = new TFile("writeEvtKey.root","RECREATE");
fo->WriteObject(evt, chevt);
```

Snippets of code: readEvtKey.cc

```
gSystem->Load("libRflxHepMCdict");
ROOT::Cintex::Cintex::Enable();

HepMC::GenEvent* evt;
TFile fi(argv[1]);
fi.GetListOfKeys()->Print();

TIter next(fi.GetListOfKeys());
TKey *key;
while ((key=(TKey*)next())) {
  fi.GetObject(key->GetName(), evt);

  if(evt) {
    std::cout << "Event: " << key->GetName() << std::endl;
    evt->print();
  }
  else {
    std::cout << "Null pointer!" << std::endl;
  }
}
```

Test executables (continued)

- Based on GenReflex (continued):
  ○ Executable: `testHepMCIKey0.x` based on source file `testHepMCIKey0.cc`
  Reads in a `HepMC ASCII` file, to be specified at the command line and writes out a file in `genreflex` dictionary format.

- Based on Cint:
  ○ Executable: `writeEvtTree.x` based on source file `writeEvtTree.cc`
  Writes a `TTree` with simple `GenEvent`'s with one `GenParticle` and its `GenVertex` to the output file `writeEvtTree.root`

  ○ Executable: `readEvtTree.x` based on source file `readEvtTree.cc`
  Read in a file of `GenEvent`'s stored in a `TTree` (file name to be specified at command line) and print out bytes read each event.
  Under construction, not working yet!

  ○ Executable: `testHepMCITree0.x` based on source file `testHepMCITree0.cc`
  Reads in a `HepMC ASCII` file, to be specified at the command line and writes out a `TTree` of `GenEvent`'s to a file.

Performances

- Test condition:
  10k GenEvent's with a simple GenParticle and its GenVertex written out to a file.

○ Writing out TTree is 4-5 times faster than TKey Objects.

○ Written TTree based on genreflex dictionary about 20-30% smaller in comparison to Cint dictionary.

- Test condition:
  100 busy Tevatron collider multi-jet GenEvent's generated with Pythia8 ($\geq$ 1 stable particle with $p_\perp > 20$ GeV and $|\eta_{\max}| < 3.0$) written out to a file.

○ a) TTree compression factor of about 3 compared to HEPMC ASCII file.
b) Object file based on TKey's about 20% larger in comparison to HEPMC ASCII file.

Conclusions

- Root I/O for HepMC in two alternative ways presented
- Provide platform portable framework with code examples
- TTree performance advantage in terms of speed
  ($\sim$ 4-5 times faster Output) and file size
  in comparison to TKey Objects and HepMC ASCII format
  (compression factor ~ 3 for realistic events)
- Users are invited to test both approaches