Random number generator issues for Run II

P. Murat, Fermilab)

October 1, 1998

1 Introduction

- major consumers of the random numbers: teh event generators and the detector simulation
- maintenance issue: simulation code has to be debuggable
  - if the executable crashes on the event number 10000 one doesn't want to regenerate previous 9999 events to start debugging the problem - one needs to be able to restart from exactly this event and to be able to reproduce the problem
- the status of **all** the random number generators used needs to be saved in the end of event processing
- there should be a single place in the Run II system which would know about all the random number generators in use and therefore it would be possible to save/restore the status of all of them
- MC generators: all use independent generators, need to decide what to do with them

- in Run I we had such a package - **R_N** (CDF-2459)
- underlying generator RANMAR : CALL RANMAR(a,n)
  - a -array of uniformly distributed (pseudo) random numbers
  - n - number of them
- fast, 900000000 independent sequences, period of each sequence $\mathbf{10^{43}}$
- multiple streams, initialized to generate different sequences
- save/restore operation
- it would be nice to have similar functionality in Run II - need a Random Number Manager
- 2 issues:
  - how the interface to this manager looks like
  - which package to use as underlying RN generation engine
- I've written an interface (an include file) for such a manager (the 1st iteration)

```
class TRn : public TObject {
protected:
// it is a singleton
  Trn*    fInstance;

  int     fInitialized;
// total number of streams;
  int     fNStreams;
// active stream
  int     fActiveStream;
// some of the CDF RN streams are reserved
// for quite specific purposes and can't be
// used arbitrarily, fReserved[i] = 1 if
// the stream is reserved, 0 otherwise
  int*    fReserved;
// buffer to keep the saved state of the object
  int*    fSaveBuffer;
// constructor and destructor are protected
  Trn();
  ~Trn();
```

```
public:
// ****** initialization/termination
  static int Init(int nstreams = 50);
  static int Terminate();
// ****** access methods

  TRn*    Instance() { return fInstance; }

// ****** generators (equivalent rn_gen.F)

  float  Uniform     (int stream);
  void   Rannor      (int stream, float& a, float& b);
  float  Gauss       (int stream, float mean, float sigma);
  float  Exp         (int stream, float tau);
  float  Gamma       (int stream, float mean);
  int    Poisson     (int stream, float mean);
  float  Landau      (int stream);

// not available from R_N

  float  BreitWigner (int stream, float mean, float gamma);

// if not using GEANT3 GRNDM

  void   Grndm       (float* rn, int n);
  void   Grndmq      (int seed1, int seed2, int seq, char* opt, int len);

// ****** utilities
  void   Print  (int stream);
  int    Status (int stream);
// save current status into an internal buffer
  int    Save   ();
// get first available stream and simultaneously
// reserve it
  int    GetStream ();
  ClassDef(Trn,1)
};
extern Trn*    gRn;
```

2 Underlying RN engine

We are using GEANT3, which has its own random number generator GRNDM cloned from CERN's RANEQU

- 215 independent sequences, separated by $\mathbf{10^9}$
- period of about $\mathbf{10^{18}}$
- 2 times slower than RANMAR
- if we want to implement a RN management we may
  - base it on GRNDM, otherwise we need to take GRNDM/GRNDMQ code out of GEANT and emulate them based on different generator
  - use RANMAR
  - use HepJamesRandom from CLHEP, which is a C++ rewrite of RANMAR
    today CLHEP RN engines do not allow to save the generator state to any place other than an external ASCII file
- GEANT4 approach: use CLHEP RN
  - the documentation doesn't discuss random numbers at all
  - looking at the code (G4MuEnergyLoss.cc): the default stream is used and it is the responsibility of the user to change it

3 MC generators

- HERWIG: uses HWRGEN (cloned from RANEQU)
- PYTHIA: uses PYR (cloned from RANMAR)
- ISAJET: uses CERN G900 routine RANF
- all of these have either
  * to be taken out of the libraries and emulated, in which case we need to maintain these libraries within CDF
  * they should be handled in a special way: RN manager should be aware of them and do save/restore for them explicitly, but then one always need to link against all the MC generator libraries
- in principle one can do the same with GEANT3
- also need to emulate RNDM etc or forbid direct use of those in the public code

conclusion: to make the simulation code debuggable we need to impose a certain discipline. There are several options we may choose from