FSR in Gauss: Generator's statistics

- What type of object is going in the FSR ?
- How are the objects accumulated ?
- In the job creating the objects, do they have to be accessed by algorithms other than the one accumulating them?
- Is there a need to propagate the objects to later processing steps?
- Is there a need to combine objects from the FSR of several input files? If so, with what operations, and when?

G.Corti, P.Robbe

LHCb Software Week - 19 June 2009

Use Cases

- **Counters are used in the generation phase of Gauss to obtain the efficiency of the Generator Level Cuts applied on the generated events, necessary to compute event yields:**
  - **For example, the usual cut is that the signal B hadron must have all stables particles from its decay chain inside the LHCb acceptance.**
- **Also other counters are computed to monitor generation quantities:**
  - **Cross-sections (bb, cc, ...)**
  - **« Luminosity » (i.e. number of interactions per event)**
  - **Fraction of B*(*), of B0/B+/Bs/b-baryons**
- **Indispensable ingredient to calculate the efficiency of channels selections**
  - **Using the same events (or at least a fraction) that will be processed by the analysis, i.e. guaranteeing the same settings**

Generation Counters

- One « counter » accumulates two quantities:
  - A = Total number
  - B = Number after cuts
- And a result which is the fraction: B/A and its binomial error, computed at the end of the job.
- The result is then printed at the end of each job, together with a description string:
  - For example:
    - Number of events for generator level cut, before : 6147, after : 2586
    - Efficiency of the generator level cut : 0.42069 +/- 0.0062966
- Also simpler counters where only number is printed
    - Luminosity

Log files example

```
Generation                  INFO ************************************************
Generation.FixedLumin...    INFO ***********   Luminosity counters   **************
Number of all events (including empty events) : 237057
Number of events with 0 interaction : 87233

Generation                  INFO ***********   Generation counters   **************
Number of generated events : 149824
Number of generated interactions : 237805
Number of generated interactions with >= 1b : 1724
Number of generated interactions with >= 3b : 27
Number of generated interactions with 1 prompt B : 22
Number of generated interactions with >= 1c : 15515
Number of generated interactions with >= 3c : 2068
Number of generated interactions with >= prompt C : 1379
Number of generated interactions with b and c : 319
Number of accepted events : 200
Number of interactions in accepted events : 408
```

simple counters

etc…

```
Generation.SignalRepe...    INFO *************   Signal counters   ****************
Number of events for generator level cut, before : 639, after : 110
Efficiency of the generator level cut : 0.17214 +/- 0.014934
Number of z-inverted events : 90

Number of events for generator particle level cut, before : 413, after : 69
Efficiency of the generator particle level cut : 0.16707 +/- 0.018356
Number of events for generator anti-particle level cut, before : 226, after : 41
Efficiency of the generator anti-particle level cut : 0.18142 +/- 0.025634

Number of signal B0 in sample : 123 [fraction : 0.615 +/- 0.034407]
Number of signal B~0 in sample : 77 [fraction : 0.385 +/- 0.034407]

Number of accepted B0 : 28 [fraction : 0.35897 +/- 0.054315]
Number of accepted B+ : 35 [fraction : 0.44872 +/- 0.056315]
```

efficiencies

fractions

etc…

Technical details

- **The counters are defined in the initialization by the *Generation* algorithm and the dedicated tools (*Pileup*, *SampleGeneration*)**
  - **simple member variables (e.g. n_bbEvents)**
  - **boost::arrays (string and int) and enum to define filling index**

```
namespace GenCounters {
  /// Type for hadron counter
  typedef boost::array< unsigned int , 5 > BHadronCounter ;
  typedef boost::array< unsigned int , 4 > DHadronCounter ;
  typedef boost::array< std::string  , 5 > BHadronCNames  ;
  typedef boost::array< std::string  , 4 > DHadronCNames  ;

  enum bHadronCounterType{ Bd = 0 , ///< counter of B0
                           Bu , ///< counter of B+
                           Bs , ///< counter of Bs0
                           bBaryon , ///< counter of b-baryon
                           Bc , ///< counter of Bc+
  } ;
```

- **The counters are accumulated in the execute or by the 'active methods' of the specified tools**

- **The counters are printed in the finalize() of the Generation algorithm that asks the tools to print theirs**

Technical details (cont.)

- **Utility functions used throughout are encapsulated in a *GenCounter* namespace**
  - **to compute fraction and statistical error**

```
inline double fraction( const unsigned int A , const unsigned int B ) {
  return ( (double) A / (double) B ) ;
}
inline double err_fraction( const unsigned int A , const unsigned int B ) {
  return sqrt( A * ( 1. - ( (double) A / (double) B ) ) ) / ( (double) B ) ;
}
```

  - **to print in various forms: counter, ratio, efficiency**

```
inline void printEfficiency( MsgStream & theStream ,
                             const std::string & cutName ,
                             const unsigned int after ,
                             const unsigned int before ) {

template< typename T , std::size_t N >
inline void printArray( MsgStream & theStream ,
                        boost::array< T , N > A ,
                        boost::array< std::string , N > AName ,
                        const std::string & root ) {
```

  - **to setup and update some of the counter**

```
void setupExcitedCountersNames( ExcitedCNames & B ,
                                const std::string & root ) ;

void updateExcitedStatesCounters( const HepMC::GenEvent * theEvent ,
                                  ExcitedCounter & thebExcitedC ,
                                  ExcitedCounter & thecExcitedC ) ;
```

Counters Summary

- **Then a python script summarizes counters from several jobs (and productions):**
  - **Parse a string to find the numbers (A and B)**
  - **Then adds all A and B numbers**
  - **In order to compute the average efficiency and the errors.**

- **All counters implemented in Gauss for the generation phase are described here:**
  - **https://twiki.cern.ch/twiki/bin/view/LHCb/GengaussCounters**

- **Write the numbers on a web page and update them when more log files are available.**
  - **All reports for DC06 at**
    http://lhcb-release-area.web.cern.ch/LHCb-release-area/DOC/STATISTICS/

Snap-shot of counters

Content of generated interactions

- `Number of generated interactions with >= 1b :` **N5** Number of generated interactions containing at least 1 *b* quark.
- `Number of generated interactions with >= 3b :` **N6** Number of generated interactions containing at least 3 *b* quarks.
- `Number of generated interactions with 1 prompt B :` **N7** Number of generated interactions containing at least 1 prompt *B* (one *B* hadron without a *b* quark).
- `Number of accepted interactions with >= 1b :` **N12** Number of accepted interactions containing at least 1 *b* quark.
- `Number of accepted interactions with >= 3b :` **N13** Number of accepted interactions containing at least 3 *b* quarks.

Pythia Counters

- `0 All included subprocesses I DUMP DUMP I C1 I` Total cross-section of Minimum Bias events.

Computation of cross-sections

Quantity	Value	Error
b cross-section	C1* * N5 / N2	*C1 * sqrt(**N5** * (**N2** - **N5**) / **N2^3**)
Double *b* cross-section	C1* * N6 / N2	*C1 * sqrt(**N6** * (**N2** - **N6**) / **N2^3**)
Prompt *B* cross-section	C1* * N7 / N2	*C1 * sqrt(**N7** * (**N2** - **N7**) / **N2^3**)
c cross-section	C1* * N8 / N2	*C1 * sqrt(**N8** * (**N2** - **N8**) / **N2^3**)

Example of web report (DC06)

Event Type : 11164401 12962600 11912010 12912010 13912010 11876001 12103002 12103012 12103032 12103051 12103022

11164401 (Bd_D-rho+=DecProdCut) DC06-phys-v2-lumi2 -- Mon Jul 28 11:40:09 2008
Gauss v26r0 - DecFiles v14r1 - ParamFiles v5r0

Counters Interaction								
Mean Number of generated Pile-Up interactions	Mean Number of non-empty generated Pile-Up interactions	Mean Number of accepted Pile-Up interactions	b cross-section (mb)	Double b cross-section (mb)	c cross-section (mb)	Double c cross-section (mb)	Prompt D cross-section (mb)	b and c cross-section (mb)
0.6826	1.3798	1.6806	0.69925	0.00463	3.6435	0.17208	1.47612	0.04578
± 0.000145	± 0.000349	± 0.009353	± 0.001388	± 0.000113	± 0.00312	± 0.000690	± 0.002008	± 0.000356

Hadron Counters														
Generator level cut efficiency	Fraction of accepted B0	Fraction of accepted B+	Fraction of accepted Bs0	Fraction of accepted b-Baryon	Fraction of accepted D0	Fraction of accepted D+	Fraction of accepted Ds+	Fraction of accepted c-Baryon	Fraction of accepted B	Fraction of accepted B*	Fraction of accepted B**	Fraction of accepted D	Fraction of accepted D*	Fraction of accepted D**
0.1627	0.3930	0.4135	0.1034	0.0901	0.5156	0.1518	0.1985	0.1341	0.2009	0.6163	0.1828	1.0000	0.0000	0.0000
± 0.000929	± 0.003013	± 0.003038	± 0.001878	± 0.001766	± 0.003389	± 0.002433	± 0.002704	± 0.002311	± 0.001767	± 0.002145	± 0.001705	± 0.000000	± 0.000000	± 0.000000

Signal Counters	
particle cut efficiency	anti-particle cut efficiency
0.1625	0.1629
± 0.001315	± 0.001313

Number of accepted events/generated events: 51500 / 26897726 = 0.001915
Number of interactions in accepted events/generated interactions: 86549 / 37113332 = 0.002332

Statistics done (script version 080118) with 102 jobs from the following ProdIDs:

00002095

Answer's to the questions

- **What type of object is going in the FSR (e.g. counters, ratios, averages)?**
  - **the integer counter before/after the cut**
  - **the 'name'/'meaning' of the counter**
  - **additional identifiers of the 'conditions': i.e. eventtype (also in header), DecFiles version (partially in header via application version)**

- **How are the objects accumulated (How are they identified, what operations are needed on them, and when (execute, finalize etc.))?**
  - **they are indentified internally via an enum, in the printout by their 'name'**
  - **they are added at execute**
  - **ratio and binomial error calculated at finalize of Generation algorithm when *printCounters* of algorithm or tools are called**

- **In the job creating the objects, do they have to be accessed by algorithms other than the one accumulating them?**
  - **yes, only the Generation algorithm prints them out**
  - **the script manipulates and does the calculations for the final numbers**

Answers to the questions (cont.)

- **Is there a need to propagate the objects to later processing steps (i.e. should they be copied to the output file if read from the input file)?**
  - **Yes as the .sim files are not normally kept**
  - **It would also give the exact numbers for the subset one may be looking at**

- **Is there a need to combine objects from the FSR of several input files? If so, with what operations, and when?**
  - **Yes! (but only for the same EventType)**
  - **The basic counters "A" and "B" need to be added**
  - **We will at least run a job either once at the end of a production or for a big production few times that combines the FSR of all the input files and provides**

    - the average efficiencies and their errors

    - the cross sections

    and publish a summary on the web

Conclusion

- Generator Statistics would benefit a lot from FSR to replace scripts reading log files:
  - complicated bookkeeping with old method → automatic procedure with FSR
  - script method relies on string identification i.e. less then ideal for maintenance
  - requests by various people to run script on their private productions would be easier to fulfill
  - must be transmitted to event file that is kept
- Other statistics for monitoring the simulation (for example num of MCHits in a detector) for each job may also make use of FSR