Counters in Gaudi

Vanya BELYAEV

Counters

- Gaudi has class **StatEntity**, generic counter
- It can be operated in many ways:
  - Simple local object
  - Integrated with **GaudiCommon** base **GaudiAlgorithm**/**GaudiTool**
  - Through "statistical" services: **IStatSvc** **ICounterSvc**
  - With helper wrapper class **Stat**

Generic Counter

- Accumulate the additive value, called `Flag` **`double`**
- Keep the number of entries **`long`**
- Keep the minimal value **`double`**
- Keep the maximal value **`double`**
- Keep the sum of $\texttt{Flag}^2$ **`double`**
  - Required for estimation of RMS

A bit similar to *1-bin histogram*

Methods & Operations

- *Obvious:*

```
nEntries, flag, flag2,
  flagMean, flagRMS,
  flagMin, flagMax
reset, print,
  fillStream, toString
```

- *"The main"*

```
addFlag  ( double )
```

- *Less trivial:*

```
efficiency, eff,
  efficiencyErr,
  effErr
```

- Shortcuts for "addFlag": recommended

```
+= , -=          double
++, --
+=               counter
```

- Standalone operators:

```
+ , -
   counter +  counter
   counter +- double
   double  +  counter
```

"efficiency"

- If counter has been filled only with 0 and 1 it can be interpreted as "*binomial efficiency counter*":

  `efficiency` (shortcut: **`eff`**)

  `efficiencyErr` (shortcut: **`effErr`**)

  - (calculated properly using *binomial estimate*)
  - If the content can't be intepreted in this way, **`return -1`**
  - Note: **`effErr`** is never equal to 0...

- One can use counters directly:

```
StatEntity cnt;

for ( int i = 0 ; i < 10000 ; ++i )
{
   cnt += sin( i ) ;
}

std::cout << cnt.flagMean() << std::endl ;
```

GaudiCommon<TYPE>

- Algorithm/tool base class

Note the reference here

```
StatEntity& cnt = counter ( 'sin' )

for ( int i = 0 ; i < 10000 ; ++i )
{
   cnt += sin( i ) ;
}
```

- Counters are printed at finalization

GaudiPython

- Algorithm/tool base class

Note the reference here

```
cnt = self.counter ( 'sin' )

for i in range(0,10000) :
        cnt += math.sin( i )

print cnt
```

- "self" here is python image of GaudiAlgorithm
  GaudiAlgo from GaudiPython.GaudiAlgs
- Counters are printed at finalization

Technicalities

- The object is very light:
  - 1 `long` + 4 `doubles` (+strange field by ATLAS)
- The object is *not inefficient*
  - The main method is non-inlined
  - All "decorations" are inlined
- Many code lines are aware about it:

  `IStatSvc,ICounterSvc, IMonitorSvc`

TWiKi:

http://cern.ch/twiki/bin/view/LHCb/GaudiAlgorithm#GaudiAlgCounters

http://cern.ch/twiki/bin/view/LHCb/FAQ/LHCbFAQ#How_to_use_the_Statistical_Count

Patch #2873 15.03.2k+9

- Allows to extract *current counters* from Gaudi components (tools&algorithms)
- Allows to print *all current counters*
- Incredibly convinient for *interactivity* & *monitoring*
  - E.g. get summary info from ***Parallel*** (sub)jobs…

Python monitoring:
GaudiPython, Bender, Parallel, Ganga, DIRAC,…

```
>>> cmp=gaudi.algorithm('CmpName')   ## get tool/alg/iProperty
>>> cmt.StatPrint = True              ## print all counters
>>> counters  = cmp.counters()        ## get all counters
>>> for key in counters :             ## loop over counters
 ……    print counters[key].efficiency() ## print the counter
>>> cnt = cmt.getCounter( '#accept')  ## get counter
>>> print ' Accept: ', cnt.eff()      ## use it!
```

In my own "*to-do-list*"

- (for other, non FSR purposes)
- `DataObject` which has `StatEntity`
  - And behaves as `StatEntity` (cast)
- `DataObject` which has `Map<Key,StatEntity>`
- Both can be registered in TS (TES?) and *probably* satisfy at least some criteria for FSR

TWiki & LHCb FAQ

Using the generic statistical counters

The `GaudiAlgorithm` and `GaudiTool` base classes are equipped with the method `counter`, which returns a reference to the local instance of *the named coun* `StatEntity&`):

```
01000   const std::size_t nTracks = ... ;
01010
01020   // increment the counter:
01030   counter("#Tracks")+= nTracks ;
```

The current content of the generic counter could be inspected:

```
01000   const StatEntity& cnt = ... ;
01010   always()
01020       << " #of entries " << cnt.nEntries() << endmsg
01030       << " Total sum   " << cnt.flag()         << endmsg
01040       << " Mean value  " << cnt.flagMean () << endmsg
01050       << " RMS "  << cnt.flagRMS() << endmsg
01060       << " Minimal value " << cnt.flagMin() << endmsg
01070       << " Maximal value " << cnt.flagMax () << endmsg ;
```

If the content of the generic counter allows the interpretation as a binominal efficiency counter one can ask for efficiency and its uncertainty:

```
01000   always()
01010       << " Efficiency " << cnt.eff()
01020       << " +-"          << cnt.effErr()     << endmsg;
```

All counters will be printed at the end of the job in a table if the property `"StatPrint"` is activated (default):

```
Counter          SUCCESS Number of counters : 8
Counter          SUCCESS      Counter     |     #     |    sum     | mean/eff^* | rms/err^*  |    min     |    max     |
Counter          SUCCESS  "G"             |     26490 |   -326.309 |  -0.012318 |    0.99484 |    -3.8836 |     3.8273 |
Counter          SUCCESS  "Gneg"          |      2712 |       2712 |     1.0000 |    0.00000 |     1.0000 |     1.0000 |
Counter          SUCCESS  "Gpos"          |      2688 |       2688 |     1.0000 |    0.00000 |     1.0000 |     1.0000 |
Counter          SUCCESS  "NG"            |     26490 |      26490 |     1.0000 |    0.00000 |     1.0000 |     1.0000 |
Counter          SUCCESS *"eff"           |      5400 |       2688 |( 49.77778 +- 0.6804071)%|  -------   |  -------   |
Counter          SUCCESS  "executed"      |      5400 |       5400 |     1.0000 |    0.00000 |     1.0000 |     1.0000 |
Counter          SUCCESS  "g2"            |      5400 |    5341.386 |    0.98915 |     1.3535 | 1.8914e-006 |    13.130 |
Counter          SUCCESS  "gauss"         |      5400 |  -8.611382 | -0.0015947 |    0.99456 |    -3.6235 |     3.5079 |
```

How to use *the Statistical Counters* ?

There are three ways for usage of *the statistical counters* in Gaudi framework:

1. Local counters through `GaudiAlgorithm` and `GaudiTool` base classes
2. Counter form *Statistical Service* `IStatSvc`
3. Counters from *Counter Service* `ICounterSvc`

All three approaches deal with the same objects = the generic counters of the C+ type `StatEntity` and have the different ap counters, used withing the same *algorithm* or *tool*. The counters are naturally grouped by components. The second approach same counter from different components. The third approach is essentially the same as the second approach, but in addition worth to note that the second and the third approaches unavoidably have some CPU overhead with respect to the first approa

The generic counters in `GaudiAlgorithm` and `GaudiTool` base classes

See GaudiAlgorithm reference

How can I use `IStatSvc` for counters?

For *the global* counters, the approach with the usage of `IStatSvc` is more preferrable, however here since the generic count `Stat` is suitable to preserve *the object/instance semantic* instead *the pointer semantic*:

```
01000   IStatSvc* statSvc = ... ;
01010
01020   // get the counter form the service ("book on-demand")
01030   Stat stat ( statSvc, "Total Energy" );
01040
01050   // increment it:
01060   stat += eTotal ;
01070
01080   // check the underlying generic counter (optional)
01090   const StatEntity* counter = stat->counter() ;
01100
01110   // increment the counter in the service (in one go)  ("book-on-demand")
01120   Stat nTr ( statSvc , "#Tracks" , nTracks ) ;
```

The table of II counters will be printed at the end of the job, if the property `"StatPrintOutTable"` activated:

```
******Stat******      INFO ********************************************************************************
******Stat******      INFO  The Final stat Table (ordered)
******Stat******      INFO ********************************************************************************
******Stat******      INFO       Counter      |     #      |    sum     | mean/eff^* | rms/err^*  |
******Stat******      INFO  "counter1"        |      20000 |   2000.452 |    0.10002 |2.2600e-005 |
******Stat******      INFO *"eff"             |      20000 |      10000 |( 50.00000 +- 0.3535534)%|
******Stat******      INFO  "counter3"        |      20000 |      13000 |    0.65000 |    0.35000 |
******Stat******      INFO  "counter2"        | 100020000 |    9993260 |   0.099913 |   0.010999 |
******Stat******      INFO ********************************************************************************
```

Summary

- Gaudi does contain simple&powerful generic counter, coherent with many components
- If one needs counters (e.g. for FSR) the first look at `StatEntity`
  - No need to reinvent the wheel
- Counters which can be registered in TES are in pipeline
  - the primary goal: some private monitoring