Pythia Workshop - Messaging Discussion

1. **Output Methods & Streams**
2. **Some useful features from Vincia**
    1. METHOD_NAME
    2. Verbosity Levels
3. **Inputs from Others & Discussion**

Peter Skands

April, 2020

Output Methods & Streams

What we have now:

```
infoPtr->errorMsg(string messageIn, string extraIn,
  showAlways=false);
```

Smart functionality: (definitely want to keep?)

User sets how many times same "messageIn" can be repeated during run;
Thereafter silenced but tally still incremented + summary available at end.

(2nd argument **`extraIn`** can vary from call to call, still incrementing same tally.)

Points to discuss?

Currently writes explicitly to **`cout`**, that's it.

Default for EVERYTHING cout; + allow user to set own.

Use cerr instead, if these are indeed **error** msgs? (other messages discussed later.)

Past: allowed user to supply own stream. Removed since had to be passed around as argument (and little used?). PhysicsBase + InfoPtr ➤ can reintroduce, if motivated?

Apart from stream direction, allow to call user method ("**`onError`**") ?

Callback: to discuss with Steve if CMS needs it. Check with others, eg. MCnet exp contacts, exp MC convenors?

Information, Warnings, Errors, and Aborts

Currently, we generally use 4 levels, self-imposed via 1st arg:

```
infoPtr->errorMsg("Info from CoupSUSY::initSUSY:" …
infoPtr->errorMsg("Warning in CoupSUSY::initSUSY:" …
infoPtr->errorMsg("Error from CoupSUSY::initSUSY:" …
infoPtr->errorMsg("Abort from CoupSUSY::initSUSY:" …
```

Looks nice in output and summary. Want to keep those aspects.

But might want to control streams independe

Ability to suppress, e.g., everything except e

Counter-intuitive to use "`errorMsg()`" for s
statement, so also many direct `cout` stateme

I think these 4 types of output make sens

Maybe 4 different methods, each with own st
callback option); by default cout for **info** and

Other aspects? Do we want a "debugMsg"?

5 types good.
Nishita: need for local verbosity / local debug output. Currently done with, e.g.: DBSUSY = settingsPtr->mode("SLHA:verbose") > 2 ? true : false;
Peter: was also done in Vincia by defaulting to global verbose level but allowing e.g., local ones like SLHA:verbose to override.
Phil: instead, for debug (which is the primary use case for local output), parse the __METHOD_NAME__ to check for specific matches, in list which can be specified by user.
Marius: Separate class for messaging (instead of doing it all in Info) ?
Torbjorn: prefer to flush debug statements out of code releases. Can be done in an automated clean way (especially if debug has its own method and debug levels).
Peter: Then at least archive (tag) the pre-flushed version as well so that we have an internal "debug-level" version of each release.
Option: maybe leave one (lowest) debug level in code (Phil: and call it report), so users can be told to run at least that, and we would have to do the rest ourselves.

__METHOD_NAME__

Cumbersome + lengthy to write method name all the time:

```
infoPtr->errorMsg("Error in StringFragmentation::finalRegion: problem");
```

(Also prone to mistakes if moving code between methods.)

In VINCIA, we use a preprocessor macro to automatically substitute method names:

```
infoPtr->errorMsg("Error in "+__METHOD_NAME__+": problem");
```

I think it was Helen who adapted it for VINCIA. (A modified version of a method from boost, distributed with GPLV2 license)

Note: moving "Error in" inside the errorMsg method (if warnings, info, and aborts have their own, see previous page), this would further reduce to just the business end:

```
infoPtr->errorMsg(__METHOD_NAME__,"problem");
```

Pre-processor macros?

Compiler dependent.

So __METHOD_NAME__ picks which one to use

```
#ifndef __METHOD_NAME__

#ifndef VINCIA_FUNCTION
#if ( defined(__GNUC__) || (defined(__MWERKS__) && (__MWERKS__ >= 0x3000)) \
|| (defined(__ICC) && (__ICC >= 600)) )
# define VINCIA_FUNCTION __PRETTY_FUNCTION__
#elif defined(__DMC__) && (__DMC__ >= 0x810)
# define VINCIA_FUNCTION __PRETTY_FUNCTION__
#elif defined(__FUNCSIG__)
# define VINCIA_FUNCTION __FUNCSIG__
#elif ( (defined(__INTEL_COMPILER) && (__INTEL_COMPILER >= 600)) \
|| (defined(__IBMCPP__) && (__IBMCPP__ >= 500)) )
# define VINCIA_FUNCTION __FUNCTION__
#elif defined(__BORLANDC__) && (__BORLANDC__ >= 0x550)
# define VINCIA_FUNCTION __FUNC__
#elif defined(__STDC_VERSION__) && (__STDC_VERSION__ >= 199901)
# define VINCIA_FUNCTION __func
#else
# define VINCIA_FUNCTION "unkno
#endif
#endif // end VINCIA_FUNCTION
```

Christian B: uneasy about the compiler-dependence. There are possibly alternatives …otherwise prefer not to do it.
Peter: willing to accept the (minimal) compiler dep since this is not a mission critical component.
Marius: If this is only for debug, it could be flushed from the code at release?
Phil: let's think about it.

Does it catch everything? Or enough to use more generally in Pythia?

Verbosity Levels

We have our 4 output types. (maybe 5, if debug is a separate one)

Is there a 1:1 correspondence with what level of output we want to see / get callbacks from?

For example?

I don't think so

-1: silent

0: aborts only

1: errors and aborts only

2: warnings, errors, and aborts

3: infos, warnings, errors, and aborts

4: debug output, warnings, errors, and aborts

Verbosity Levels

We have our 4 output types. (maybe 5, if debug is a separate one)

In VINCIA, we use verbosity levels to control what is printed

Mostly controls warnings, information and debug type output

(Errors and aborts normally always printed)

+ rough guideline text describing what kind of output should be done at each level

(In need of formalising / updating)

```
// Suppressed verbosity.
const int silent         = 0;
const int quiet          = 1;
// Normal verbosity for warnings.
const int normal         = 2;
// Extra verbosity for warnings.
const int quiteloud      = 3;
const int loud           = 4;
const int veryloud       = 5;
// Debug verbosity levels.
const int debug          = 6;
const int louddebug      = 7;
const int veryloudebug   = 8;
const int superdebug     = 9;
```

My impression: this has worked relatively well for us, but there are too many levels. Aim to consolidate to something like one level for 3+4+5 and one or two for 6+7+8+9.

Where we would likely go

```
// Completely silent operation.
const int silent        = -1;
// Minimal output. (Mainly aborts and errors.)
const int quiet         = 0;
// Normal verbosity. (Default.)
const int normal        = 1;
// Reporting mode, for users when reporting issues.
const int report        = 2;
// Full debug, for authors only.
const int debug         = 3;
```

Torbjorn: what about the initialisation info? How does that fit in?
Marius: keep them separate

(We also have some formatted num2str() and bool2str() methods)

Peter: not discussed yet

(Also note we do no caching; string handling very expensive/slow in C++)

Leif L : showed example from RIVET which conserves CPU, only executing the argument if the corresponding level is enabled. Obviates need for if (verbose == blah) at calling point …

Points from Marius

Marius: Get rid of cout, use logger class instead.
Leif G: in favour of getting rid of cout; would eg allow truly silent operation.
*Peter: Do we **always** want truly silent operation though, if there are errors? Perhaps allow error output to cout if complicated to get around (eg in functions).There are anyway not very many of these cases. (Perfection should not be the enemy of the good.)*
Leif G: if not critical, maybe just get rid of those outputs altogether? And if critical, program should abort anyway?
Peter: maybe. I don't think it is a big issue. Let's do rest of cleanup and see how big the problem that remains really is.
Peter: note also that we need to be able to output quite heterogenous data types like Nishita was mentioning for eg rank-3 tensors in SUSY.
Marius: maybe the logger provides a stream?
Torbjorn: write locally to a stringstream and then pass that as a string to the logger?

Marius: Constructor prints banner, cannot be silenced or pushed to different stream.
Torbjorn: we do have some reason to want to advertise our existence especially if run by other tool (chains), so silencing no, other stream yes.
Marius: move it from Constructor to Init. (Thus allowing user to define their own stream before banner is printed.)

Torbjorn: currently, all that seems to be requested are methods that allows to set the streams. (That's exactly what CMS asked for, according to Steve's contact.)
Marius: maybe users want more capabilities?
Stefan: could be a UserHook? Other framework might discourage novice users.
Leif L: the people likely to be doing this are advanced users.
All ?: let's start with what it seems we are actually being asked for, and see where to go from there.

Marius: question about multi-threading and Angantyr issues. Is it enough to just set the streams?
Stefan: yes, should be thread-safe. (It's redirection that is the problem.)