Logging in MITK

Deprecation of “std::cout”

~~std::cout << “some console output” << std::endl;~~

~~printf(“ …… \n”);~~

Why?

std::cout is

- not thread-safe
- allows neither filtering nor categorizing
- may not be available on all platforms

Our logging approach in MITK:

LOG_INFO as a compatible (and more comfortable) replacement for std::cout

```
LOG_INFO << “some console output”;
```

LOG_INFO features

LOG_INFO also automagically adds following to each log message:

- File & Line
- Level (INFO, WARN, ERROR, FATAL, DEBUG)
- Functionname
- Modulename

The Logging backend can provide thread-safety and optionally support:

- Thread-ID
- Time

Categories

by using the ( ) operator its possible to specify categories:

```
LOG_INFO("openCherry") << "job system initialized";
```

sub-categories are defined by a '.':

```
LOG_INFO("openCherry.ui") << "sub-categories";
```

example of defining a custom macro:

```
#define CHERRY_INFO LOG_INFO("openCherry")
```

custom macros can use same syntax:

```
CHERRY_INFO("ui") << "syntax stays";
```

you may create sub-categories (they'll be delimited by a '.'):

```
#define CHERRYUI_INFO CHERRY_INFO("ui")
```

"signature":

```
LOG_INFO << "no category";
LOG_INFO( const char * ) << "single category";
LOG_INFO( const char * )( const char * ) << "sub categories";
```

4+1 Levels

```
void otherMethod()
{

  ; INFO, for standard information messages
  LOG_INFO(“dataStorage") << “new data storage created”;

  ; WARN, to inform about problems, that may affect performance or quality
  LOG_WARN(“gpgpu") << “no openGL hardware support detected, using software emulation”;

  ; ERROR, if its not possible to solve a problem
  LOG_ERROR(“dataStorage.fileLoader") << “input data corrupt”;

  ; FATAL, if the application cant continue running
  LOG_FATAL(“openCherry.ui”) << “cant open main qt window, EXITING”;

  ; DEBUG, special level, that can be enabled by a special C Preprocessor flag
  LOG_DEBUG(“rendering.mapper”) << “calling mapper ” << mapper.name;

  #define MBILOG_ENABLE_DEBUG

  you can add that #define at (the most) top of your .cpp to temporarily enable debug
  messages for that single file.

  or enable it for all files through the Cmake flag ‘MBILOG_ENABLE_DEBUG_MESSAGES’

  if MBILOG_ENABLE_DEBUG is not defined before #include <mbilog.h>,
  you can be sure that the LOG_DEBUG statement will be compiled to ZERO code,
  but its still checked for correct syntax and semantics
}
```

Conditionals

The () Operator also supports a bool as argument

The Log message will only be sent, when all given arguments are true;

```
LOG_DEBUG( x > 1000 ) << “x is too large”;

LOG_INFO( mitk::isVerboseLogEnabled() ) << “verbose message”;
```

you may also define a custom macros:

```
#define VERBOSE_INFO LOG_INFO( mitk::useVerboseLogging() )
```

“signature” is as with categories:

```
LOG_INFO( bool )( bool )…
```

How to use

```
#include “mbilog.h”
```

in your .cpp file

but it is often already included through mitkCommon.h