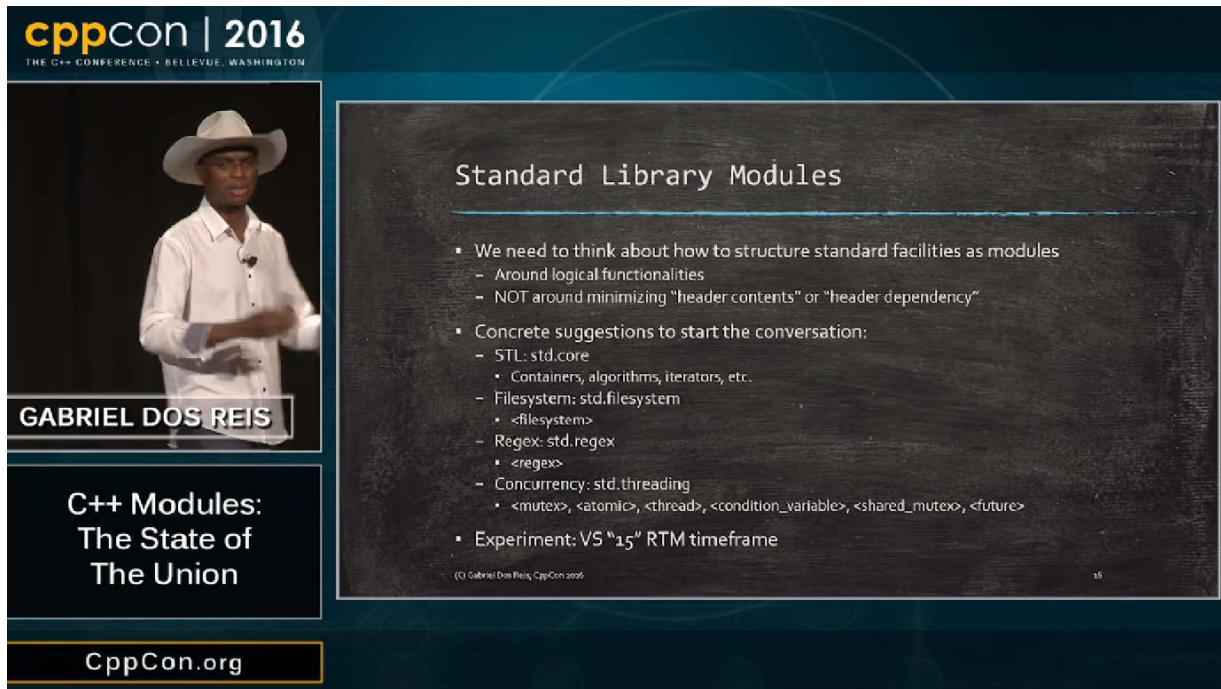


Names for the standard modules

1. Introduction



This proposal intend to bring possibles good name names for the [future] Standard C++ Modules. As Gabriel dos Reis invited in "Modules: The State of the Union" (CppCon 2016), here I am with names to use in the future modules. At the end of this, that are some proposals to the modular system.

2. Technical Specification

I. The Input/Output Streams

```
std.io.stream //<iostream>  
  
std.io.filestream //<fstream>  
  
std.io.stringstream //<sstream>  
  
std.io.manip //<iomanip>  
  
std.locale //<locale>
```

As `std::string` is necessary in all that modules, the `std.string` need to be imported in each one.

II. The String classes and utilities

```
std.string //<string>  
  
std.regex //<regex>
```

III. The Math and numbers utilities

```
std.math //<cmath>
std.complex //<complex>
std.bitset //<bitset>
std.numeric //<numeric>
std.limits //<limits>
```

A good thing may be have a namespace for the standard math facilities: double val = math::sqrt(16);

IV. Standard Template Library

```
std.stl.algo //<algorithm>
std.stl.iterator //<iterator>
std.stl.containers //all containers imported here.
std.stl.vector //<vector>
std.stl.list //<list>
std.stl.map //<map>
... each container with its own module!
```

A good thing may be have a namespace for the standard algorithm facilities :

```
std::vector<int> v {0, 1, 2, 3, 4};
auto result1 = parallel::find(v.begin(), v.end(), -1); // Parallel TS
auto result2 = algo::find(v.begin(), v.end(), 3);
```

V. Standard Threading Library

```
std.concurrency //all related modules here
std.thread //<thread>
std.mutex //<mutex> <shared_mutex>
std.condition //<condition_variable>
std.future //<future>
std.atomic //<atomic>
```

VI. Technical Specifications (example)

```
std.ts.parallel //<experimental/parallel>
```

VII. Others

```
std.chrono //<chrono>
std.random //<random>
std.memory //<memory>
...
```

VIII. The Standard C Library

```
std.c.io //<cstdio>
std.c.lib//<cstdlib>
std.c.time //<ctime>
... std.c. [Name]
```

3. Other proposal: Solve the necessity of forward declarations insine a module interface

```
import gui.button;
import gui.eventhandler;
import core.time;
import std.io.stream;

module my.button;

namespace gui
{
    export class TheButton : public Button
    {
        public:
            TheButton(std::string name) : Button(name)
            {
                installEventHandler(&handler);
            }

        private:
            ButtonHandler handler;
    };

    class ButtonHandler : public EventHandler
    {
        public:
            void clickEvent(Widget *context) override
            {
                auto time = Time::currentTime().toString("hh:mm:ss");

                std::cout.printf("[%s] Button %s was clicked!", time, context.name());
            }
    };
}
```

This gives to programmers flexibility and add a little piece of code abstraction and solve the problem of forward declarations, as modules are self contained unites. I think: If the two related functions/classes are on the same module interface, why that's needed?

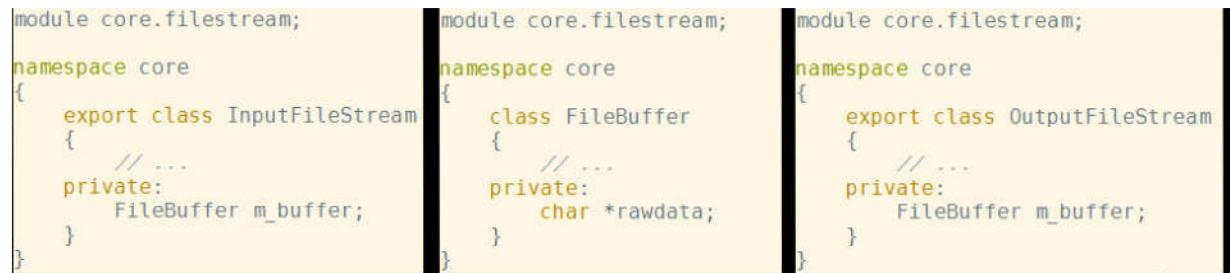
4. Other proposal: Allow a module interface to expand multiple files

This is good for large module interfaces. As for example, std.random (<random>) that have alot of class, free functions, etc. Having multiple interface header files, the module can be splitted in in several files, but only one .ifc is generated.

- std.random.random_device.ixx // The random_device class and all related stuffs
- std.random.mersenne_twister_engine.ixx // The mersenne_twister_engine template class, specifications and all related stuffs
- ... and so on

Making it, we can simplify correction of bugs, revisions, etc.

A example:



```
module core.filestream;
namespace core
{
    export class InputFileStream
    {
        // ...
        private:
            FileBuffer m_buffer;
    }
}

module core.filestream;
namespace core
{
    class FileBuffer
    {
        // ...
        private:
            char *rawdata;
    }
}

module core.filestream;
namespace core
{
    export class OutputFileStream
    {
        // ...
        private:
            FileBuffer m_buffer;
    }
}
```

5. Conclusion

I thank Gabriel dos Reis and all the C++ Team for the good work that is made to the modules standardization.