Roland Bock

rbock eudoxos de

https://github.com/rbock/sqlpp11

https://github.com/rbock/kiss-templates

CppCon, 2015-09-22

I need to generate

- HTML
- ATEX
- eMails
- C++ code

I want a text template system that

- has a simple, yet powerful language
- works for all kinds of text
- is statically compiled

Kiss Templates

Keep it simple, stupid!

Hello World!

```
$class Sample
%void render()
%{
   Hello ${data.name}!
%}
$endclass
```

Hello World!

```
$class Sample
%void render()
%{
   Hello ${data.name}!
%}
$endclass
```

Hello World!

```
$class Sample
%void render()
%{
   Hello ${data.name}!
%}
$endclass
```

```
%#include <functional>
%namespace test
у.Г
  $class Sample
  %void render()
  %{
    %for (auto&& name : data.names)
    %{
        Hello ${name} aka ${hash<std::string>{}(name)}!
    %}
  %}
  $endclass
%}
```

```
%#include <functional>
%namespace test
%{
  $class Sample
  %void render()
  %{
    %for (auto&& name : data.names)
    %{
        Hello ${name} aka ${hash<std::string>{}(name)}!
    %}
  %}
  $endclass
%}
```

```
%#include <functional>
%namespace test
у.Г
  $class Sample
  %void render()
  %{
    %for (auto&& name : data.names)
    %{
        Hello ${name} aka ${hash<std::string>{}(name)}!
    %}
  %}
  $endclass
%}
```

```
%#include <functional>
%namespace test
у.Г
  $class Sample
  %void render()
  %{
    %for (auto&& name : data.names)
    %{
        Hello ${name} aka ${hash<std::string>{}(name)}!
    %}
  %}
  $endclass
%}
```

Translating the template

\$ kiste2cpp hello_world.kiste > hello_world.h

```
#include <iostream>
#include <vector>
#include <string>
#include <hello_world.h>
#include <kiste/raw.h>
```

```
#include <iostream>
#include <vector>
#include <string>
#include <hello_world.h>
#include <kiste/raw.h>

struct Data
{
   std::vector<std::string> names;
};
```

```
#include <iostream>
#include <vector>
#include <string>
#include <hello_world.h>
#include <kiste/raw.h>
struct Data
  std::vector<std::string> names;
};
int main()
  const auto data = Data{{"World"}}:
  auto serializer = kiste::raw{std::cout};
  auto sample = test::Sample(data, serializer);
  sample.render();
```

```
#include <iostream>
#include <vector>
#include <string>
#include <hello_world.h>
#include <kiste/raw.h>
struct Data
  std::vector<std::string> names;
};
int main()
  const auto data = Data{{"World"}}:
  auto serializer = kiste::raw{std::cout};
  auto sample = test::Sample(data, serializer);
  sample.render();
```

Compile and run

```
$ clang++ -std=c++14 hello.cpp -I. -o hello
```

Compile and run

```
$ clang++ -std=c++14 hello.cpp -I. -o hello
```

\$./hello Hello World aka 18151272839730735401!

Compile and run

```
$ clang++ -std=c++14 hello.cpp -I. -o hello
```

\$./hello Hello World aka 18151272839730735401!

What else?

The syntax summary

• %<C++ code>

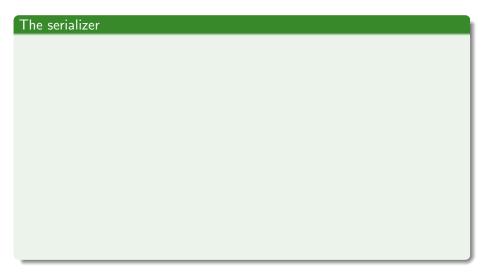
- %<C++ code>
- \$class <name>
- \$class <name> : <base>
- \$member <class> <name>
- \$endclass

- %<C++ code>
- \$class <name>
- \$class <name> : <base>
- \$member <class> <name>
- \$endclass
- \${<expression>}
- \$\ \\$\ \{\expression>\}
- \$call{<void expression>}

- %<C++ code>
- \$class <name>
- \$class <name> : <base>
- \$member <class> <name>
- \$endclass
- \${<expression>}
- \$\ \\$\ \\ \\ \expression> \}
- \$call{<void expression>}
- \$1

- %<C++ code>
- \$class <name>
- \$class <name> : <base>
- \$member <class> <name>
- \$endclass
- \${<expression>}
- \$\ \\$\ \{\expression>}
- \$call{<void expression>}
- \$1
- \$\$ and \$%

- %<C++ code>
- \$class <name>
- \$class <name> : <base>
- \$member <class> <name>
- \$endclass
- \${<expression>}
- \$\ \\$\ \{\expression>\}
- \$call{<void expression>}
- \$1
- \$\$ and \$%
- Anything else inside a function of a template class is text



```
struct Serializer
{
  void text(const char* text);
```

```
struct Serializer
{
  void text(const char* text);
  template<typename T>
  void escape(T&& t);
```

```
struct Serializer
{
  void text(const char* text);

  template<typename T>
  void escape(T&& t);

  // optionally
  template<typename T>
  void raw(T&& t);
```

```
struct Serializer
  void text(const char* text);
  template<typename T>
  void escape(T&& t);
  // optionally
  template<typename T>
  void raw(T&& t);
  void report_exception(long lineNo,
                        const std::string& expression,
                        std::exception_ptr e);
};
```

And if I make a mistake?

And if I make a mistake?

Summary

- A dozen syntax elements
- Full power of C++
- Fit for all text formats via specific serializers
- Statically compiled templates

Summary

- A dozen syntax elements
- Full power of C++
- Fit for all text formats via specific serializers
- Statically compiled templates

Kept it simple, but not stupid!

Feedback welcome!

https://github.com/rbock/kiss-templates

Thank You!