Kiss Templates

Roland Bock

rbock eudoxos de

https://github.com/rbock/sqlpp11

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

CppCon, 2015-09-22

Kiss Templates

I need to generate

- HTML
- LaTeX
- eMails
- C++ code

Kiss Templates

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!

Kiss Templates

Hello World!

```
$class Sample

%void render()
%{
  Hello ${data.name}!
%}

$endclass
```

Kiss Templates

Hello World!

```
$class Sample

%void render()
%{
  Hello ${data.name}!
%}

$endclass
```

Kiss Templates

Hello World!

```
$class Sample

%void render()
%{
  Hello ${data.name}!
%}

$endclass
```

Hello World! Extended edition

```
%#include <functional>
%namespace test
%{
  $class Sample

  %void render()
  %{
    %for (auto&& name : data.names)
    %{
        Hello ${name} aka ${hash<std::string>{}(name)}!
    %}
  %}

  $endclass
%}
```

Hello World! Extended edition

```
%#include <functional>
%namespace test
%{
  $class Sample

  %void render()
  %{
    %for (auto&& name : data.names)
    %{
        Hello ${name} aka ${hash<std::string>{}(name)}!
    %}
  %}

  $endclass
%}
```

Hello World! Extended edition

```
%#include <functional>
%namespace test
%{
  $class Sample

  %void render()
  %{
    %for (auto&& name : data.names)
    %{
        Hello ${name} aka ${hash<std::string>{}(name)}!
    %}
  %}

  $endclass
%}
```

Hello World! Extended edition

```
%#include <functional>
%namespace test
%{
  $class Sample

  %void render()
  %{
    %for (auto&& name : data.names)
    %{
        Hello ${name} aka ${hash<std::string>{}(name)}!
    %}
  %}

  $endclass
%}
```

Kiss Templates

Translating the template

```
$ kiste2cpp hello_world.kiste > hello_world.h
```

Using the Hello-World template

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

Using the Hello-World template

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

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

Using the Hello-World template

```
#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();
}
```

Using the Hello-World template

```
#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
```

Kiss Templates

Compile and run

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

$ ./hello
      Hello World aka 181512728397307354O1!
```

Kiss Templates

Compile and run

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

$ ./hello
      Hello World aka 181512728397307354O1!
```

What else?

The syntax summary

- `%<C++ code>`

Kiss Templates

The syntax summary

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

Kiss Templates

The syntax summary

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

The syntax summary

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

Kiss Templates

The syntax summary

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

Kiss Templates

The syntax summary

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

The serializer

The serializer

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

The serializer

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

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

The serializer

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

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

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

The serializer

```
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?

```
$ g++ -I. -std=c++11 test.cpp -o hello_world

In file included from test.cpp:2:0:
hello_world.kiste: In instantiation of  void test::Hello_t<DERIVED_T, DATA_T, SEP
test.cpp:17:16:   required from here
hello_world.kiste:7:48: error:  const _data_t  has no member named  Name
     Hello ${data.Name}!
```

Kiss Templates

Summary

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

Kiss 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!