# INTRODUCTION TO WEB DEVELOPMENT IN C++ WITH WT 4

https://www.webtoolkit.eu/wt

Roel Standaert
FOSDEM 2018

emweb

# CONTENTS

- What is Wt?
- A simple “Hello world”
- Creating a larger application, with:
  - Templates
  - Style sheets
  - i18n
- Being ~~lazy~~ efficient with Bootstrap
- Where to go from here?

# WHAT IS WT?

- A server side web framework written in C++
- Made for single page applications
  - REST is possible, API currently verbose
  - Websites also possible: e.g. webtoolkit.eu
- Inspired by Qt → widget based
- Abstraction of web technologies, adapts to what's available. No JavaScript required.

# WHY C++?

"I know we sometimes give C++ a bit of a bad rap for having funny defaults and weird, odd edge cases, but just take a look at JavaScript, really."

– Matt Godbolt at CppCon 2017, youtu.be/bSkpMdDe4g4?t=2870

# WHY C++?

- Low level vulnerabilities?
  - XSS, CSRF and SQL injection are larger threats
  - Use best practices and tools (compiler warnings, Valgrind, asan,...)
- Small footprint: no garbage, no runtime, create small statically linked binaries
- Familiar to Qt developers
- Ecosystem: Wt for web, any C or C++ library for everything else

# WT 4

- Released in September 2017
- C++11: it's about time

# HELLO WORLD!


Click "Greet me!" (or press Enter):

# HELLO WORLD!

```
#include <Wt/WApplication.h>
#include <Wt/WServer.h>

int main(int argc, char *argv[])
{
    return Wt::WRun(argc, argv, [](const Wt::WEnvironment &env){
        auto app = std::make_unique<Wt::WApplication>(env);

        // ...

        return app;
    });
}
```

- WRun: creates **WServer** instance, and accepts connections

# HELLO WORLD!

```
#include <Wt/WApplication.h>
#include <Wt/WServer.h>

int main(int argc, char *argv[])
{
    return Wt::WRun(argc, argv, [](const Wt::WEnvironment &env){
        auto app = std::make_unique<Wt::WApplication>(env);

        // ...

        return app;
    });
}
```

- **WRun**: creates **WServer** instance, and accepts connections
- New connection → callback is called

# HELLO WORLD!

```
#include <Wt/WApplication.h>
#include <Wt/WServer.h>

int main(int argc, char *argv[])
{
    return Wt::WRun(argc, argv, [](const Wt::WEnvironment &env){
        auto app = std::make_unique<Wt::WApplication>(env);

        // ...

        return app;
    });
}
```

- WRun: creates WServer instance, and accepts connections
- New connection → callback is called
- WEnvironment describes “environment”, e.g. user agent information

# HELLO WORLD!

```
#include <Wt/WApplication.h>
#include <Wt/WServer.h>

int main(int argc, char *argv[])
{
    return Wt::WRun(argc, argv, [](const Wt::WEnvironment &env){
        auto app = std::make_unique<Wt::WApplication>(env);

        // ...

        return app;
    });
}
```

- WRun: creates WServer instance, and accepts connections
- New connection → callback is called
- WEnvironment describes “environment”, e.g. user agent information
- Callback returns `unique_ptr` to WApplication.

# HELLO WORLD!

```
#include <Wt/WApplication.h>
#include <Wt/WServer.h>

int main(int argc, char *argv[])
{
    return Wt::WRun(argc, argv, [](const Wt::WEnvironment &env){
        auto app = std::make_unique<Wt::WApplication>(env);

        // ...

        return app;
    });
}
```

- **WRun**: creates **WServer** instance, and accepts connections
- New connection → callback is called
- **WEnvironment** describes “environment”, e.g. user agent information
- Callback returns `unique_ptr` to **WApplication**.

→ One instance of WApplication per session

# HELLO WORLD! (CONTINUED)

```
auto app = ...
auto root = app->root(); // WContainerWidget *root (<div>)
```

# HELLO WORLD! (CONTINUED)

```
auto app = ...
auto root = app->root();

// add a new WText (<span>)
root->addWidget(std::make_unique<Wt::WText>("Your name, please? "));
```

# HELLO WORLD! (CONTINUED)

```
auto app = ...
auto root = app->root();

// add a new WText, using the addNew shorthand
root->addNew<Wt::WText>("Your name, please? ");
```

# HELLO WORLD! (CONTINUED)

```
auto app = ...
auto root = app->root();

root->addNew<Wt::WText>("Your name, please? ");

// add a WLineEdit (<input type="text">)
Wt::WLineEdit* edit = root->addNew<Wt::WLineEdit>();
```

# HELLO WORLD! (CONTINUED)

```
auto app = ...
auto root = app->root();

root->addNew<Wt::WText>("Your name, please? ");

Wt::WLineEdit* edit = root->addNew<Wt::WLineEdit>();
// add a WPushButton (<button>)
auto btn = root->addNew<Wt::WPushButton>("Greet me!");
```

# HELLO WORLD! (CONTINUED)

```
auto app = ...
auto root = app->root();

root->addNew<Wt::WText>("Your name, please? ");

Wt::WLineEdit* edit = root->addNew<Wt::WLineEdit>();
auto btn = root->addNew<Wt::WPushButton>("Greet me!");
// add a WBreak (<br>)
root->addNew<Wt::WBreak>();
```

# HELLO WORLD! (CONTINUED)

```
auto app = ...
auto root = app->root();

root->addNew<Wt::WText>("Your name, please? ");

Wt::WLineEdit* edit = root->addNew<Wt::WLineEdit>();
auto btn = root->addNew<Wt::WPushButton>("Greet me!");
root->addNew<Wt::WBreak>();
// empty WText in Plain format (default is filtered XHTML)
auto result = root->addNew<Wt::WText>();
result->setTextFormat(Wt::TextFormat::Plain);
```

# HELLO WORLD! (CONTINUED)

```
auto app = ...
auto root = app->root();

root->addNew<Wt::WText>("Your name, please? ");

Wt::WLineEdit* edit = root->addNew<Wt::WLineEdit>();
auto btn = root->addNew<Wt::WPushButton>("Greet me!");
root->addNew<Wt::WBreak>();
auto result = root->addNew<Wt::WText>();
result->setTextFormat(Wt::TextFormat::Plain);

// create a lambda function
auto showGreeting = [edit,result]{
    result->setText(Wt::WString("Hello, {1}!").arg(edit->text()));
};
```

# HELLO WORLD! (CONTINUED)

```
auto app = ...
auto root = app->root();

root->addNew<Wt::WText>("Your name, please? ");

Wt::WLineEdit* edit = root->addNew<Wt::WLineEdit>();
auto btn = root->addNew<Wt::WPushButton>("Greet me!");
root->addNew<Wt::WBreak>();
auto result = root->addNew<Wt::WText>();
result->setTextFormat(Wt::TextFormat::Plain);

auto showGreeting = [edit,result]{
    result->setText(Wt::WString("Hello, {1}!").arg(edit->text()));
};

// connect signals
edit->enterPressed().connect(showGreeting);
btn->clicked().connect(showGreeting);
```

# HELLO WORLD! (FINAL CODE)

```cpp
#include <Wt/WApplication.h>
#include <Wt/WBreak.h>
#include <Wt/WContainerWidget.h>
#include <Wt/WLineEdit.h>
#include <Wt/WPushButton.h>
#include <Wt/WServer.h>
#include <Wt/WString.h>
#include <Wt/WText.h>

int main(int argc, char *argv[])
{
  return Wt::WRun(argc, argv, [](const Wt::WEnvironment &env){
    auto app = std::make_unique<Wt::WApplication>(env);
    auto root = app->root();

    root->addWidget(std::make_unique<Wt::WText>("Your name, please? "));
    auto edit = root->addNew<Wt::WLineEdit>();
    auto btn = root->addNew<Wt::WPushButton>("Greet me!");
    root->addNew<Wt::WBreak>();
    auto result = root->addNew<Wt::WText>();
    result->setTextFormat(Wt::TextFormat::Plain);

    auto showGreeting = [edit,result]{
      result->setText(Wt::WString("Hello, {1}!").arg(edit->text()));
    };

    edit->enterPressed().connect(showGreeting);
    btn->clicked().connect(showGreeting);

    return app;
  });
}
```

# BUILD AND RUN

## Release:

```
c++ -std=c++14 -O3 -DNDEBUG -o hello.wt hello.cpp \
  -I/path/to/wt/include -L/path/to/wt/lib \
  -lwt -lwthttp
```

## Debug:

```
c++ -std=c++14 -O0 -g3 -o hello.wt hello.cpp \
  -I/path/to/wt/include -L/path/to/wt/lib \
  -lwtd -lwthttpd
```

## Run:

```
./hello.wt --docroot . --http-listen localhost:8080
```

--docroot: Where to serve static content from

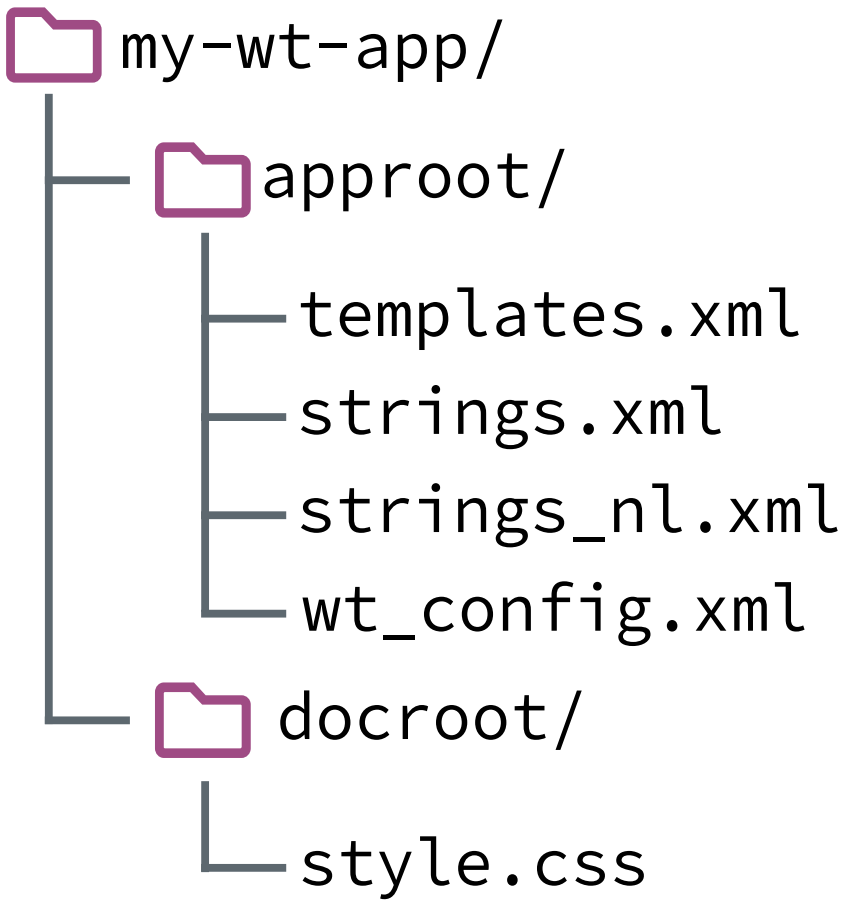
--http-listen: host and port to listen on
(0.0.0.0 for any IPv4, [0::0] for any IPv6)

# DEMO

# CREATING LARGER WT APPLICATIONS

- Classic: using layouts:
  - WBoxLayout, WGridLayout, WBorderLayout
- Using templates:
  - HTML and CSS (if CSS isn't your forte, there's Bootstrap)

# TYPICAL PROJECT STRUCTURE




Configuration, templates, SQLite database, i18n,...

Templates

English strings

Dutch strings

Wt configuration

Static web content

CSS stylesheet

# TEMPLATE.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<messages>
    <message id="tpl.template">
        ${tr:str.prompt} ${edit} ${btn} <br />
        ${result}
    </message>
</messages>
```

${variable}

${function:args}

# TEMPLATE.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<messages>
    <message id="tpl.template">
        ${tr:str.prompt} ${edit} ${btn} <br />
        ${result}
    </message>
</messages>
```

${variable}

${function:args}

## strings.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<messages>
    <message id="str.prompt">Please enter your name:</message>
    <message id="str.greet-me">Greet me</message>
    <message id="str.result">Hello, {1}!</message>
</messages>
```

## strings_nl.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<messages>
    <message id="str.prompt">Voer jouw naam in:</message>
    <message id="str.greet-me">Begroet me</message>
    <message id="str.result">Hallo, {1}!</message>
</messages>
```

## STYLE.CSS

```
body {
   font-family: sans-serif;
}
```

# LOADING TEMPLATES, STRINGS & STYLESHEETS

```
// load approot/template.xml
app->messageResourceBundle().use(
    app->appRoot() + "template");


// load approot/strings.xml
// (or approot/strings_nl.xml)
app->messageResourceBundle().use(
    app->appRoot() + "strings");


// add stylesheet
app->useStyleSheet("style.css");
```

# HELLOTEMPLATE

```
class HelloTemplate : public Wt::WTemplate {
public:
  HelloTemplate();
};
```

# HELLOTEMPLATE (CONTINUED)

```
HelloTemplate::HelloTemplate()
  : WTemplate{tr("tpl.template")}
{
    addFunction("tr", &WTemplate::Functions::tr);
    auto edit = bindWidget("edit", std::make_unique<Wt::WLineEdit>());
    auto btn = bindNew<Wt::WPushButton>("btn", tr("str.greet-me"));
    bindEmpty("result");
    auto showGreeting = [this]{
        auto edit = resolve<Wt::WLineEdit*>("edit");
        bindString("result",
                   tr("str.result").arg(edit->text()),
                   Wt::TextFormat::Plain);
    };
    edit->enterPressed().connect(showGreeting);
    btn->clicked().connect(showGreeting);
}
```

# NEW MAIN

```
int main(int argc, char *argv[])
{
  return Wt::WRun(argc, argv, [](const Wt::WEnvironment &env) {
    auto app = std::make_unique<Wt::WApplication>(env);

    app->messageResourceBundle().use(
          app->appRoot() + "template");
    app->messageResourceBundle().use(
          app->appRoot() + "strings");

    app->useStyleSheet("style.css");

    app->root()->addNew<HelloTemplate>();

    return app;
  });
}
```

# BUILD AND RUN

## Release:

```
c++ -std=c++14 -O3 -DNDEBUG -o template.wt template.cpp \
  -I/path/to/wt/include -L/path/to/wt/lib \
  -lwt -lwthttp
```

## Debug:

```
c++ -std=c++14 -O0 -g3 -o template.wt template.cpp \
  -I/path/to/wt/include -L/path/to/wt/lib \
  -lwtd -lwthttpd
```

## Run:

```
./template.wt --approot approot --docroot docroot \
  --http-listen localhost:8080
```

# DEMO

# USING BOOTSTRAP: BEING ~~LAZY~~ EFFICIENT

```
auto theme = std::make_shared<Wt::WBootstrapTheme>();
theme->setVersion(Wt::BootstrapVersion::v3);
app->setTheme(theme);
```

- Includes Bootstrap CSS
- Adds appropriate classes
- Wt's own widgets in Bootstrap style

# TEMPLATE.XML: FORM

```
<div class="form-horizontal">
    <div class="form-group">
        <label class="col-md-3 control-label" for="${id:edit}">
            ${tr:str.prompt}
        </label>
        <div class="col-md-6">
            ${edit}
        </div>
    </div>
    <div class="form-group">
        <div class="col-md-offset-3 col-md-6">
            ${btn class="btn-primary"}
        </div>
    </div>
</div>
```

# TEMPLATE.XML: RESULT

```
${<if-result>}
    <div class="row">
        <div class="alert alert-info text-center
                    col-md-offset-1 col-md-8">
            ${result}
        </div>
    </div>
${</if-result>}
```

```
${<condition>}
${</condition>}
```

# HELLOTEMPLATE (CONTINUED)

```
HelloTemplate::HelloTemplate()
  : WTemplate{tr("tpl.template")}
{
    addFunction("id", &WTemplate::Functions::id);
    addFunction("tr", &WTemplate::Functions::tr);
    auto edit = bindWidget("edit", std::make_unique<Wt::WLineEdit>());
    auto btn = bindNew<Wt::WPushButton>("btn", tr("str.greet-me"));
    bindEmpty("result");
    auto showGreeting = [this]{
        auto edit = resolve<Wt::WLineEdit*>("edit");
        setCondition("if-result", true);
        bindString("result",
                   tr("str.result").arg(edit->text()),
                   Wt::TextFormat::Plain);
    };
    edit->enterPressed().connect(showGreeting);
    btn->clicked().connect(showGreeting);
}
```

# BUILD AND RUN

## Release:

```
c++ -std=c++14 -O3 -DNDEBUG -o bootstrap.wt bootstrap.cpp \
  -I/path/to/wt/include -L/path/to/wt/lib \
  -lwt -lwthttp
```

## Debug:

```
c++ -std=c++14 -O0 -g3 -o bootstrap.wt bootstrap.cpp \
  -I/path/to/wt/include -L/path/to/wt/lib \
  -lwtd -lwthttpd
```

## Run:

```
./bootstrap.wt --approot approot --docroot docroot \
  --http-listen localhost:8080 \
  --resources-dir=/path/to/wt/share/Wt/resources
```

# DEMO

# WHERE TO GO FROM HERE?

- Tutorials: https://webtoolkit.eu/wt/documentation
  - A hands-on introduction to Wt
  - A hands-on introduction to Wt::Dbo
  - Introduction to Wt::Auth
  - Library overview
- Widget gallery: https://webtoolkit.eu/widgets
- GitHub: https://github.com/emweb/wt
- Examples: examples directory in Wt source tree

# THANK YOU!