

Programming with Wt

A C++ library for developing stateful and highly interactive web applications

ARMIN SOBHANI (ASOBHANI@SHARCNET.CA)

SHARCNET

UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

JUNE 24, 2015

Outline

- Library Overview
- A typical web application developed using Wt
- Installation
- Hello World!
- Signals and Slots
- Interactive Hello World
- Widget Gallery
- Wt Variants

Library Overview

What is Wt ('witty')

- ❑ C++ library and application server for developing web applications
- ❑ Free and open source (<http://www.webtoolkit.eu/>)
- ❑ Widget-centric rather than page-centric web toolkit
- ❑ Brings desktop programming model to web development
- ❑ Similar in concept to Qt and wxWindows
- ❑ Supports fallback mechanism for maximum browser compatibility
- ❑ Mature, robust, feature reach, well documented and well supported

Wt Features at a Glance

- **Core Library**
- Event Handling
- Native Painting System
- Built-in Security
- Object Relational Mapping Library
- Unit Testing
- Deployment

Core Library

- ✓ Supports major browsers (Firefox/Gecko, Internet Explorer, Safari, Chrome, Konqueror, Opera and web crawlers)
- ✓ Develop and deploy on Unix/Linux/Mac or Microsoft Windows (Visual Studio) environments
- ✓ Equal behavior with or without support for JavaScript or Ajax
- ✓ Efficient rendering and (sub-) millisecond latency
- ✓ Integrated Unicode support and pervasive localization
- ✓ Support for browser history navigation (back/forward buttons and bookmarks)

Wt Features at a Glance

- Core Library
- **Event Handling**
- Native Painting System
- Built-in Security
- Object Relational Mapping Library
- Unit Testing
- Deployment

Event Handling

- ✓ Type safe signal/slot API for responding to events
- ✓ Listens for keyboard, mouse, focus, scroll or drag'n'drop events
- ✓ Automatically synchronizes form field data from browser to server and tracks server-side changes to be rendered in browser
- ✓ Integrate with JavaScript libraries
- ✓ Timed events and server-initiated updates ("server push")
- ✓ Uses plain HTML CGI, Ajax or WebSockets

Wt Features at a Glance

- Core Library
- Event Handling
- **Native Painting System**
- Built-in Security
- Object Relational Mapping Library
- Unit Testing
- Deployment

Native Painting System

- ✓ Unified 2D painting API which uses the browsers native (vector) graphics support (inline VML, inline SVG, or HTML5 canvas), or renders to common image formats (PNG, GIF, ...) or vector formats (SVG, PDF)
- ✓ Unified GL-based 3D painting API which uses WebGL in the browser or server-side OpenGL (fallback)
- ✓ Rich set of 2D and 3D charting widgets

Wt Features at a Glance

- Core Library
- Event Handling
- Native Painting System
- **Built-in Security**
- Object Relational Mapping Library
- Unit Testing
- Deployment

Built-in Security

- ✓ Kernel-level memory protection
- ✓ Supports encryption and server authentication using Secure Sockets Layer (SSL) or Transport Layer Security (TLS) through HTTPS
- ✓ Enables continuous use of HTTPS through low bandwidth requirements (fine-grained Ajax)
- ✓ Built-in Cross-Site Scripting (XSS) prevention
- ✓ Not vulnerable to Cross-site Request Forgery (CSRF)
- ✓ Not vulnerable to breaking the application logic by skipping to a particular URL
- ✓ Session hijacking mitigation and risk prevention
- ✓ DoS mitigation
- ✓ A built-in authentication module implements best practices for authentication, and supports third party identity providers using OAuth 2.0, and (later) OpenID Connect

Wt Features at a Glance

- Core Library
- Event Handling
- Native Painting System
- Built-in Security
- **Object Relational Mapping Library**
- Unit Testing
- Deployment

Object Relational Mapping Library (Wt::Dbo)

- ✓ Maps Many-to-One and Many-to-Many relations to STL-compatible collections
- ✓ Provides schema generation (aka DDL: data definition language) and CRUD operations (aka DML: data manipulation language)
- ✓ Each session tracks dirty objects and provides a first-level cache
- ✓ Flexible querying which can query individual fields, objects, or tuples of any of these (using Boost.Tuple)
- ✓ Comes with Sqlite3, Firebird, MariaDB/MySQL and PostgreSQL backends

Wt Features at a Glance

- Core Library
- Event Handling
- Native Painting System
- Built-in Security
- Object Relational Mapping Library
- **Unit Testing**
- Deployment

Unit Testing

- ✓ Event handling code constructs and manipulates a widget tree, which can easily be inspected by test code
- ✓ `Wt::Test::WTestEnvironment` allows application to be instantiated and events to be simulated in absence of a browser

Wt Features at a Glance

- Core Library
- Event Handling
- Native Painting System
- Built-in Security
- Object Relational Mapping Library
- Unit Testing
- **Deployment**

Deployment

- ✓ Built-in httpd (libwthttp)
 - Simple, high-performance web application server (multi-threaded, asynchronous I/O) based on the C++ asio library
 - Supports the HTTP(S) and WebSocket(S) protocols
 - Supports response chunking and compression
 - Single process (convenient for development and debugging), and embeddable in an existing application
 - Available for both UNIX and Win32 platforms
- ✓ FastCGI (libfcgi)
 - Integrates with most common web servers (apache, lighttpd)
 - Different session-to-process mapping strategies
 - Available only for UNIX platforms
- ✓ ISAPI
 - Integrates with Microsoft IIS server
 - Uses the ISAPI asynchronous API for maximum performance
 - Available for the Win32 platform

Wt Licensing

OPEN SOURCE

GNU General Public License (GPL)

The source code must be available to anyone who you give the application to install the application on its own server.

COMMERCIAL

	Dbo	Wt + Dbo
Widget library		+
Application server		+
HTTP + WebSockets server		+
Charting Module		+
XHTML Rendering Module		+
C++ ORM	+	+
Sqlite3 driver	+	+
PostgreSQL driver	+	+
Firebird driver	+	+
MariaDB/MySQL driver	+	+
Oracle driver		Contact us
License		
GNU General Public License	free	free
Commercial License	€175 (license text)	€599 (license text)

About

Help

News

Support

Terms of Use

Tools

Examples

Submit

Register

Login



New Job Submission

1. Job Settings (Optional)

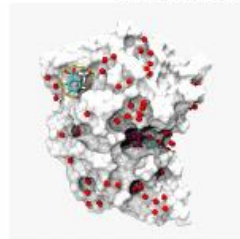
Number of CPUs:

Wall clock (in hours, max: 24):

2. Select Script

Ready-made script:

Description:



Unconstrained ligand exploration an binding site search

This script explores the entire space and locates different binding sites for the ligand.

Ligand must be present in the uploaded PDB file as HETATM records. The **ligand chain** and **residue IDs** and **Number of steps** must be set below.

Sample Public Web Application

Developed using Wt

<http://pele.bsc.es>

Installation

Installation

- **System Requirements**
 - **Mandatory**
 - Optional
 - Connectors
- Unix
- Windows

System Requirements (Mandatory)

- ✓ CMake cross-platform make utility
(≥ 2.6 is preferred)
- ✓ boost C++ library (≥ 1.41 is preferred)
 - ☐ Boost.Date_Time
 - ☐ Boost.Regex
 - ☐ Boost.Program_options
 - ☐ Boost.Signals
 - ☐ Boost.Random
 - ☐ Boost.System
 - ☐ Boost.Thread

Installation

- **System Requirements**
 - Mandatory
 - **Optional**
 - Connectors
- Unix
- Windows

System Requirements (Optional)

- ✓ **OpenSSL**
For HTTPS protocol by the web client (Http::Client) and web server (wthttp connector)
- ✓ **Libharu**
Rendering support for PDF documents
- ✓ **GraphicsMagick**
For outputs to raster images like PNG or GIF
- ✓ **Pango**
For text rendering of TrueType fonts in WPdfImage and WRasterImage
- ✓ **PostgreSQL, MySQL, and Firebird**
For the ORM library (Wt::Dbo)

Installation

- **System Requirements**
 - Mandatory
 - Optional
 - **Connectors**
- Unix
- Windows

System Requirements (Connectors)

- ✓ FastCGI (Unix only)
 - Apache 1 or 2, or any web server which supports the FastCGI protocol
 - Apache mod_fastcgi
- ✓ Built-in http daemon, wthttpd
 - libz (for compression-over-HTTP)
 - OpenSSL (for HTTPS support)
- ✓ ISAPI (Win32 only)

Installation

- System Requirements
 - Mandatory
 - Optional
 - Connectors
- **Unix**
- Windows

Unix

✓ Ubuntu

```
$ sudo apt-get install witty witty-dev witty-doc witty-dbg  
witty-examples
```

✓ Others (Linux, MacOS)

```
$ git clone git://github.com/kdeforche/wt.git  
$ cd wt  
$ mkdir build  
$ cd build  
$ cmake ../  
$ cmake .  
$ make  
$ sudo make install
```

Installation

- System Requirements
 - Mandatory
 - Optional
 - Connectors
- Unix
- **Windows**

Windows

- ✓ Binary builds:

<http://sourceforge.net/projects/witty/files/wt/>

Hello World!

Qt

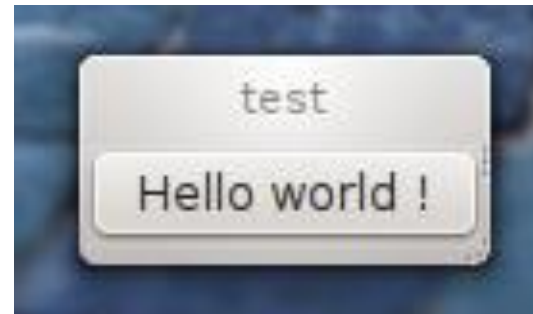
Hello World!

```
#include <QApplication>
#include <QPushButton>

int main(int argc, char **argv)
{
    QApplication app(argc, argv);

    QPushButton button("Hello world!");
    button.show();

    return app.exec();
}
```



Wt

- **hello.cpp**
- Compiling / Linking
- Program Options
- Running

hello.cpp

```
#include <Wt/WApplication>
#include <Wt/WPushButton>

using namespace Wt;

class HelloApplication : public WApplication
{
public:
    HelloApplication(const WEnvironment& env);
};

HelloApplication::HelloApplication(const WEnvironment& env)
:    WApplication(env)
{
    root()->addWidget(new WPushButton("Hello World!"));
}

WApplication* createApplication(const WEnvironment& env)
{
    return new HelloApplication(env);
}

int main(int argc, char** argv)
{
    return WRun(argc, argv, &createApplication);
}
```

Compiling / Linking

```
$ g++ hello.cpp -o hello -lwt -lwthttp
```

Wt

- hello.cpp
- **Compiling / Linking**
- Program Options
- Running

Program Options

```
$ ./hello --help
```

Wt

- hello.cpp
- Compiling / Linking
- **Program Options**
- Running

Wt

- hello.cpp
- Compiling / Linking
- Program Options
- **Running**

Running

```
$ ./hello --docroot . --http-address  
0.0.0.0 --http-port 8080
```

Interactive Hello World

Signals and Slots

SIGNAL

Message that an object can send

Some QPushButton signals:

clicked

doubleClicked

checked

unChecked

SLOT

Function used to respond to a signal

It can be a/an:

Ordinary function

Member function

C++11's Lambda

Connecting Signals to Slots

In order to respond to a signal, a slot must be *connected* to a signal:

- Simple Wt-way:

```
button->clicked().connect(this, &HelloApplication::greet);
```

- Using boost::bind():

```
nameEdit_->enterPressed().connect  
    (boost::bind(&HelloApplication::greet, this));
```

- Using C++11 Lambda:

```
button->clicked().connect(std::bind( [=]  
    { greeting->setText("Hello there, " + nameEdit->text()); } ));
```

Wt

Signals/Slots

- **hello2.cpp**
 - Additional Headers
 - New Members
 - Wt Connect
 - boost:bind()
 - Compiling / Linking
- **hello3.cpp**
 - C++11 Lambda 101
 - Additional Headers
 - Lambda in Action
 - Compiling / Linking

```
#include <Wt/WApplication>
#include <Wt/WPushButton>
#include <Wt/WLineEdit>
#include <Wt/WBreak>
#include <Wt/WText>

using namespace Wt;

class HelloApplication : public WApplication
{
public:
    HelloApplication(const WEnvironment& env);

private:
    WLineEdit* nameEdit_;
    WText* greeting_;
    void greet();
};

WApplication* createApplication(const WEnvironment& env)
{
    return new HelloApplication(env);
}

int main(int argc, char** argv)
{
    return WRun(argc, argv, &createApplication);
}
```

Wt Signals/Slots

- **hello2.cpp**
 - Additional Headers
 - New Members
 - **Wt Connect**
 - **boost::bind()**
 - Compiling / Linking
- **hello3.cpp**
 - C++11 Lambda 101
 - Additional Headers
 - Lambda in Action
 - Compiling / Linking

Wt and boost::bind() Version

```

HelloApplication::HelloApplication(const WEnvironment& env)
:   WApplication(env)
{
    setTitle("Hello World");

    root()->addWidget(new WText("Your name, please ? "));
    nameEdit_ = new WLineEdit(root());
    nameEdit_->setFocus();

    WPushButton* button = new WPushButton("Greet me.", root());
    button->setMargin(5, Left);

    root()->addWidget(new WBreak());
    greeting_ = new WText(root());

    button->clicked().connect(this, &HelloApplication::greet);

    nameEdit_->enterPressed().connect
        (boost::bind(&HelloApplication::greet, this));
}

void HelloApplication::greet()
{
    greeting_->setText("Hello there, " + nameEdit_->text());
}

```

Wt Signals/Slots

- **hello2.cpp**
 - Additional Headers
 - New Members
 - Wt Connect
 - boost:bind()
 - **Compiling / Linking**
- hello3.cpp
 - C++11 Lambda 101
 - Additional Headers
 - Lambda in Action
 - Compiling / Linking

Compiling / Linking

```
$ g++ hello2.cpp -o hello2 -lwt -lwthttp  
-lboost_signals-mt
```

Wt Signals/Slots

- hello2.cpp
 - Additional Headers
 - New Members
 - Wt Connect
 - boost:bind()
 - Compiling / Linking
- **hello3.cpp**
 - **C++11 Lambda 101**
 - Additional Headers
 - Lambda in Action
 - Compiling / Linking

C++11 Lambda 101

- Lambdas allow *ad hoc* functions to be declared at block scope
- Lambda is a function object (functor)

- **Simplified syntax**

`[...] (...) ->retTypeopt {...}`

- **Captures**

- [=] outer scope is passed by value
- [&] outer scope is passed by reference

- **Examples**

- `[] { return 42; }`
- `[] () -> float { return 3.14f; }`

Wt

Signals/Slots

- hello2.cpp
 - Additional Headers
 - New Members
 - Wt Connect
 - boost:bind()
 - Compiling / Linking
- **hello3.cpp**
 - C++11 Lambda 101
 - **Additional Headers**
 - Lambda in Action
 - Compiling / Linking

```
#include <Wt/WApplication>
#include <Wt/WPushButton>
#include <Wt/WLineEdit>
#include <Wt/WBreak>
#include <Wt/WText>

using namespace Wt;

class HelloApplication : public WApplication
{
public:
    HelloApplication(const WEnvironment& env);
};

WApplication* createApplication(const WEnvironment& env)
{
    return new HelloApplication(env);
}

int main(int argc, char** argv)
{
    return WRun(argc, argv, &createApplication);
}
```

Wt Signals/Slots

- hello2.cpp
 - Additional Headers
 - New Members
 - Wt Connect
 - boost::bind()
 - Compiling / Linking
- **hello3.cpp**
 - C++11 Lambda 101
 - Additional Headers
 - **Lambda in Action**
 - Compiling / Linking

Lambda Version

```

HelloApplication::HelloApplication(const WEnvironment& env)
:   WApplication(env)
{
    setTitle("Hello World");

    root()->addWidget(new WText("Your name, please ? "));
    WLineEdit* nameEdit = new WLineEdit(root());
    nameEdit->setFocus();

    WPushButton* button = new WPushButton("Greet me.", root());
    button->setMargin(5, Left);

    root()->addWidget(new WBreak());
    WText* greeting = new WText(root());

    auto greet = [=]
    { greeting->setText("Hello there, " + nameEdit->text()); };

    nameEdit->enterPressed().connect(std::bind(greet));
    button->clicked().connect(std::bind(greet));
}

```

Wt Signals/Slots

- hello2.cpp
 - Additional Headers
 - New Members
 - Wt Connect
 - boost:bind()
 - Compiling / Linking
- **hello3.cpp**
 - C++11 Lambda 101
 - Additional Headers
 - Lambda in Action
 - **Compiling / Linking**

Compiling / Linking

```
$ g++ -std=c++11 hello3.cpp -o hello3 -lwt  
-lwthttp -lboost_signals-mt
```

Widget Gallery

Wt Variants

Native Variants / Bindings



JWt – a native Java version of Wt
<http://www.webtoolkit.eu/jwt>



WtRuby – Ruby bindings to Wt
<http://github.com/rdale/wtruby/tree/master>