OSDEVLAB

How to install websocket POCO C++ Library on Raspberry Pi

How to install websocket POCO C++ Library on Raspberry

In this tutorial, we will use POCO C++ library to create a websocket. Then we will use a browser to test the webserver and connectivity.
We will need

- POCO C++ Library
- CMAKE

1. Download and extract POCO C++ Library Files

We will first download files from POCO website. You can check the latest file from here.

> *Note: At the time of writing, the latest version is `1.6.1` . You can just adjust version accordingly.*

> *Note: We will be installing the `Basic Edition` (No external dependencies) here. If you need Complete Edition, then you can install from the above download link also.*

```
$ cd ~
$ mkdir PocoFiles
$ cd PocoFiles
$ wget http://pocoproject.org/releases/poco-1.6.1/poco-1.6.1.tar.gz
$ gunzip poco-1.6.1.tar.gz
$ tar -xf poco-1.6.1.tar
```

2. Install POCO C++ Library

Now we will install POCO C++ library. You can check full detail installation from here. Note that this process use `Cmake` . If your system doesn't have CMake yet, you can use this link to install latest CMake.

```
$ cd poco-1.6.1
$ ./configure --no-tests --no-samples
$ make -j4
$ sudo make install
```

> *Note: These arguments `--no-test --no-samples` and `-j4` are to speed up the build time.*

Once the installation is finished, you can check installed files from here. You can see the files started with `libPoco` .

```
$ cd /usr/local/lib
$ ls
```

3. Write C++ program

Blog
Dec
Sep
Aug
Ma
Ma
Feb
Jan
Dec
Jun
Jan
Dec
Nov
Abo
Vie
Labe

We will create a main.cpp file. Copy the following text and save it.

```
$ cd ~
$ mkdir PoCoWebSocketTest
$ cd PoCoWebSocketTest
$ gedit main.cpp
```

```
//
// WebSocketServer.cpp
//
// $Id: //poco/1.4/Net/samples/WebSocketServer/src/WebSocketServer.cpp#1 $
//
// This sample demonstrates the WebSocket class.
//
// Copyright (c) 2012, Applied Informatics Software Engineering GmbH.
// and Contributors.
//
// SPDX-License-Identifier: BSL-1.0
//



#include "Poco/Net/HTTPServer.h"
#include "Poco/Net/HTTPRequestHandler.h"
#include "Poco/Net/HTTPRequestHandlerFactory.h"
#include "Poco/Net/HTTPServerParams.h"
#include "Poco/Net/HTTPServerRequest.h"
#include "Poco/Net/HTTPServerResponse.h"
#include "Poco/Net/HTTPServerParams.h"
#include "Poco/Net/ServerSocket.h"
#include "Poco/Net/WebSocket.h"
#include "Poco/Net/NetException.h"
#include "Poco/Util/ServerApplication.h"
#include "Poco/Util/Option.h"
#include "Poco/Util/OptionSet.h"
#include "Poco/Util/HelpFormatter.h"
#include "Poco/Format.h"
#include <iostream>



using Poco::Net::ServerSocket;
using Poco::Net::WebSocket;
using Poco::Net::WebSocketException;
using Poco::Net::HTTPRequestHandler;
using Poco::Net::HTTPRequestHandlerFactory;
using Poco::Net::HTTPServer;
using Poco::Net::HTTPServerRequest;
using Poco::Net::HTTPResponse;
using Poco::Net::HTTPServerResponse;
using Poco::Net::HTTPServerParams;
using Poco::Timestamp;
using Poco::ThreadPool;
```

Page

```
using Poco::Util::Application;

using Poco::Util::Option;

using Poco::Util::OptionSet;

using Poco::Util::HelpFormatter;



class PageRequestHandler: public HTTPRequestHandler

    /// Return a HTML document with some JavaScript creating

    /// a WebSocket connection.

{

public:

    void handleRequest(HTTPServerRequest& request, HTTPServerResponse& response)

    {

        response.setChunkedTransferEncoding(true);

        response.setContentType("text/html");

        std::ostream& ostr = response.send();

        ostr << "<html>";

        ostr << "<head>";

        ostr << "<title>WebSocketServer</title>";

        ostr << "<script type=\"text/javascript\">";

        ostr << "function WebSocketTest()";

        ostr << "{";

        ostr << "  if (\"WebSocket\" in window)";

        ostr << "  {";

        ostr << "    var ws = new WebSocket(\"ws://" << request.serverAddress().toString() << "/ws\");";

        ostr << "    ws.onopen = function()";

        ostr << "      {";

        ostr << "        ws.send(\"Hello, world!\");";

        ostr << "      };";

        ostr << "    ws.onmessage = function(evt)";

        ostr << "      { ";

        ostr << "        var msg = evt.data;";

        ostr << "        alert(\"Message received: \" + msg);";

        ostr << "        ws.close();";

        ostr << "      };";

        ostr << "    ws.onclose = function()";

        ostr << "      { ";

        ostr << "        alert(\"WebSocket closed.\");";

        ostr << "      };";

        ostr << "  }";

        ostr << "  else";

        ostr << "  {";

        ostr << "     alert(\"This browser does not support WebSockets.\");";

        ostr << "  }";

        ostr << "}";

        ostr << "</script>";

        ostr << "</head>";

        ostr << "<body>";

        ostr << "  <h1>WebSocket Server</h1>";

        ostr << "  <p><a href=\"javascript:WebSocketTest()\">Run WebSocket Script</a></p>";

        ostr << "</body>";
    }
};


class WebSocketRequestHandler: public HTTPRequestHandler
    /// Handle a WebSocket connection.
{
public:
    void handleRequest(HTTPServerRequest& request, HTTPServerResponse& response)
    {
        Application& app = Application::instance();
        try
        {
            WebSocket ws(request, response);
            app.logger().information("WebSocket connection established.");
            char buffer[1024];
            int flags;
            int n;
            do
            {
                n = ws.receiveFrame(buffer, sizeof(buffer), flags);
                app.logger().information(Poco::format("Frame received (length=%d, flags=0x%x).", n, unsigned(flags)));
                ws.sendFrame(buffer, n, flags);
            }
            while (n > 0 || (flags & WebSocket::FRAME_OP_BITMASK) != WebSocket::FRAME_OP_CLOSE);
            app.logger().information("WebSocket connection closed.");
        }
        catch (WebSocketException& exc)
        {
            app.logger().log(exc);
            switch (exc.code())
            {
            case WebSocket::WS_ERR_HANDSHAKE_UNSUPPORTED_VERSION:
                response.set("Sec-WebSocket-Version", WebSocket::WEBSOCKET_VERSION);
                // fallthrough
            case WebSocket::WS_ERR_NO_HANDSHAKE:
            case WebSocket::WS_ERR_HANDSHAKE_NO_VERSION:
            case WebSocket::WS_ERR_HANDSHAKE_NO_KEY:
                response.setStatusAndReason(HTTPResponse::HTTP_BAD_REQUEST);
                response.setContentLength(0);
                response.send();
                break;
            }
        }
    }
};


class RequestHandlerFactory: public HTTPRequestHandlerFactory
{
public:
    {
        Application& app = Application::instance();
        app.logger().information("Request from "
            + request.clientAddress().toString()
            + ": "
            + request.getMethod()
            + " "
            + request.getURI()
            + " "
            + request.getVersion());

        for (HTTPServerRequest::ConstIterator it = request.begin(); it != request.end(); ++it)
        {
            app.logger().information(it->first + ": " + it->second);
        }

        if(request.find("Upgrade") != request.end() && Poco::icompare(request["Upgrade"], "websocket") == 0)
            return new WebSocketRequestHandler;
        else
            return new PageRequestHandler;
    }
};



class WebSocketServer: public Poco::Util::ServerApplication
    /// The main application class.
    ///
    /// This class handles command-line arguments and
    /// configuration files.
    /// Start the WebSocketServer executable with the help
    /// option (/help on Windows, --help on Unix) for
    /// the available command line options.
    ///
    /// To use the sample configuration file (WebSocketServer.properties),
    /// copy the file to the directory where the WebSocketServer executable
    /// resides. If you start the debug version of the WebSocketServer
    /// (WebSocketServerd[.exe]), you must also create a copy of the configuration
    /// file named WebSocketServerd.properties. In the configuration file, you
    /// can specify the port on which the server is listening (default
    /// 9980) and the format of the date/time string sent back to the client.
    ///
    /// To test the WebSocketServer you can use any web browser (http://localhost:9980/).
{
public:
    WebSocketServer(): _helpRequested(false)
    {
    }

    ~WebSocketServer()
    {
    }
```

```
protected:
    void initialize(Application& self)
    {
        loadConfiguration(); // load default configuration files, if present
        ServerApplication::initialize(self);
    }

    void uninitialize()
    {
        ServerApplication::uninitialize();
    }

    void defineOptions(OptionSet& options)
    {
        ServerApplication::defineOptions(options);

        options.addOption(
            Option("help", "h", "display help information on command line arguments")
                .required(false)
                .repeatable(false));
    }

    void handleOption(const std::string& name, const std::string& value)
    {
        ServerApplication::handleOption(name, value);

        if (name == "help")
            _helpRequested = true;
    }

    void displayHelp()
    {
        HelpFormatter helpFormatter(options());
        helpFormatter.setCommand(commandName());
        helpFormatter.setUsage("OPTIONS");
        helpFormatter.setHeader("A sample HTTP server supporting the WebSocket protocol.");
        helpFormatter.format(std::cout);
    }

    int main(const std::vector<std::string>& args)
    {
        if (_helpRequested)
        {
            displayHelp();
        }
        else
        {
            // get parameters from configuration file
            unsigned short port = (unsigned short) config().getInt("WebSocketServer.port", 9980);

            // set-up a server socket
            // set-up a HTTPServer instance
            HTTPServer srv(new RequestHandlerFactory, svs, new HTTPServerParams);
            // start the HTTPServer
            srv.start();
            // wait for CTRL-C or kill
            waitForTerminationRequest();
            // Stop the HTTPServer
            srv.stop();
        }
        return Application::EXIT_OK;
    }

private:
    bool _helpRequested;
};



POCO_SERVER_MAIN(WebSocketServer)
```

4. Write Cmake Text File

Again we will create CMake file. Copy the following text and save it.

```
$ gedit CMakeLists.txt
```

```
#Ref http://stackoverflow.com/questions/30114662/clion-cmake-and-poco
cmake_minimum_required(VERSION 3.3)

project(PoCoWebSocketTest)

# define the project
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -std=c++11")

set(SOURCE_FILES main.cpp)
add_executable(PoCoWebSocketTest ${SOURCE_FILES})

# set the POCO paths and libs
set(POCO_PREFIX "/usr/local") # the directory containing "include" and "lib"
set(POCO_INCLUDE_DIR "${POCO_PREFIX}/include")
set(POCO_LIB_DIR "${POCO_PREFIX}/lib")

set(POCO_LIBS
        "${POCO_LIB_DIR}/libPocoNet.so"
        "${POCO_LIB_DIR}/libPocoUtil.so"
        "${POCO_LIB_DIR}/libPocoFoundation.so")



# set the include path for the app
target_include_directories(PoCoWebSocketTest PRIVATE $(POCO_INCLUDE_DIR))
```

```
target_link_libraries(PoCoWebSocketTest "${POCO_LIBS}")
```

5. Complie and Run C++ program

Make sure we are still in `PoCoWebSocketTest` folder. We will use CMake to build the program.

```
$ mkdir build
$ cd build
$ cmake ..
$ make
$ ./PoCoWebSocketTest
```

There won't be any interesting output after we run the program. We will only see it once we use the web browser.

6. Testing Websocket with Web Browser

You can use any modern web-browser to test. Just type in the local IP adress on the browser if you are on Raspberry Pi.

```
http://localhost:9980/
```

If you have the PC that connected to same network as Raspberry Pi, you can just type in IP address of Raspberry Pi on your browser. Just like this.

```
http://192.168.1.104:9980/
```

You can even write a little html file and test it too. More detail from websocket.org itself. The server will just echo back whatever you send to it.

Labels: C++, POCO Library, Raspberry Pi, WebSocket

No comments:

Post a Comment