

```

cmake_minimum_required(VERSION 2.8.9)

# General Advice
#
# For selecting between DEBUG / RELEASE, use -DCMAKE_BUILD_TYPE=DEBUG or =RELEASE
# debug builds include source level debug info and extra logging

set(LWS_WITH_BUNDLED_ZLIB_DEFAULT OFF)
if(WIN32)
    set(LWS_WITH_BUNDLED_ZLIB_DEFAULT ON)
endif()

set(LWS_ROLE_RAW 1)
set(LWS_WITH_POLL 1)

#
# Select features recommended for PC distro packaging
#
option(LWS_WITH_DISTRO_RECOMMENDED "Enable features recommended for distro
packaging" OFF)
option(LWS_FOR_GITOWASHI "Enable features recommended for use with gitowashi" OFF)

#
# Major individual features
#
option(LWS_WITH_NETWORK "Compile with network-related code" ON)
option(LWS_ROLE_H1 "Compile with support for http/1 (needed for ws)" ON)
option(LWS_ROLE_WS "Compile with support for websockets" ON)
option(LWS_ROLE_DBUS "Compile with support for DBUS" OFF)
option(LWS_ROLE_RAW_PROXY "Raw packet proxy" OFF)
option(LWS_WITH_HTTP2 "Compile with server support for HTTP/2" ON)
option(LWS_WITH_LWSWS "Libwebsockets Webserver" OFF)
option(LWS_WITH_CGI "Include CGI (spawn process with network-connected
stdin/out/err) APIs" OFF)
option(LWS_IPV6 "Compile with support for ipv6" OFF)
option(LWS_UNIX_SOCKET "Compile with support for UNIX domain socket" OFF)
option(LWS_WITH_PLUGINS "Support plugins for protocols and extensions" OFF)
option(LWS_WITH_HTTP_PROXY "Support for HTTP proxying" OFF)
option(LWS_WITH_ZIP_FOPS "Support serving pre-zipped files" OFF)
option(LWS_WITH SOCKS5 "Allow use of SOCKS5 proxy on client connections" OFF)
option(LWS_WITH_GENERIC_SESSIONS "With the Generic Sessions plugin" OFF)
option(LWS_WITH_PEER_LIMITS "Track peers and restrict resources a single peer can
allocate" OFF)
option(LWS_WITH_ACCESS_LOG "Support generating Apache-compatible access logs" OFF)
option(LWS_WITH_RANGES "Support http ranges (RFC7233)" OFF)
option(LWS_WITH_SERVER_STATUS "Support json + jscript server monitoring" OFF)
option(LWS_WITH_THREADPOOL "Managed worker thread pool support (relies on
pthreads)" OFF)
option(LWS_WITH_HTTP_STREAM_COMPRESSION "Support HTTP stream compression" OFF)
option(LWS_WITH_HTTP_BROTLI "Also offer brotli http stream compression (requires
LWS_WITH_HTTP_STREAM_COMPRESSION)" OFF)
option(LWS_WITH_ACME "Enable support for ACME automatic cert acquisition +
maintenance (letsencrypt etc)" OFF)
option(LWS_WITH_HUBBUB "Enable libhubbub rewriting support" OFF)
option(LWS_WITH_FTS "Full Text Search support" OFF)

#
# TLS library options... all except mbedTLS are basically OpenSSL variants.
#
option(LWS_WITH_SSL "Include SSL support (defaults to OpenSSL or similar, mbedTLS

```

```

if LWS_WITH_MBEDTLS is set)" ON)
option(LWS_WITH_MBEDTLS "Use mbedTLS (>=2.0) replacement for OpenSSL. When setting
this, you also may need to specify LWS_MBEDTLS_LIBRARIES and
LWS_MBEDTLS_INCLUDE_DIRS" OFF)
option(LWS_WITH_BORINGSSL "Use BoringSSL replacement for OpenSSL" OFF)
option(LWS_WITH_CYASSL "Use CyaSSL replacement for OpenSSL. When setting this, you
also need to specify LWS_CYASSL_LIBRARIES and LWS_CYASSL_INCLUDE_DIRS" OFF)
option(LWS_WITH_WOLFSSL "Use wolfSSL replacement for OpenSSL. When setting this,
you also need to specify LWS_WOLFSSL_LIBRARIES and LWS_WOLFSSL_INCLUDE_DIRS" OFF)
option(LWS_SSL_CLIENT_USE_OS_CA_CERTS "SSL support should make use of the OS-
installed CA root certs" ON)
#
# Event library options (may select multiple, or none for default poll())
#
option(LWS_WITH_LIBEV "Compile with support for libev" OFF)
option(LWS_WITH_LIBUV "Compile with support for libuv" OFF)
option(LWS_WITH_LIBEVENT "Compile with support for libevent" OFF)
#
# Static / Dynamic build options
#
option(LWS_WITH_STATIC "Build the static version of the library" ON)
option(LWS_WITH_SHARED "Build the shared version of the library" ON)
option(LWS_LINK_TESTAPPS_DYNAMIC "Link the test apps to the shared version of the
library. Default is to link statically" OFF)
option(LWS_STATIC_PIC "Build the static version of the library with position-
independent code" OFF)
#
# Specific platforms
#
option(LWS_WITH_ESP32 "Build for ESP32" OFF)
option(LWS_WITH_ESP32_HELPER "Build ESP32 helper" OFF)
option(LWS_PLAT_OPTEE "Build for OPTEE" OFF)
#
# Client / Server / Test Apps build control
#
option(LWS_WITHOUT_CLIENT "Don't build the client part of the library" OFF)
option(LWS_WITHOUT_SERVER "Don't build the server part of the library" OFF)
option(LWS_WITHOUT_TESTAPPS "Don't build the libwebsocket-test-apps" OFF)
option(LWS_WITHOUT_TEST_SERVER "Don't build the test server" OFF)
option(LWS_WITHOUT_TEST_SERVER_EXTPOLL "Don't build the test server version that
uses external poll" OFF)
option(LWS_WITHOUT_TEST_PING "Don't build the ping test application" OFF)
option(LWS_WITHOUT_TEST_CLIENT "Don't build the client test application" OFF)
#
# Extensions (permessage-deflate)
#
option(LWS_WITHOUT_EXTENSIONS "Don't compile with extensions" ON)
#
# Helpers + misc
#
option(LWS_WITHOUT_BUILTIN_GETIFADDRS "Don't use the BSD getifaddrs implementation
from libwebsockets if it is missing (this will result in a compilation error) ...
The default is to assume that your libc provides it. On some systems such as uclibc
it doesn't exist." OFF)
option(LWS_FALLBACK_GETHOSTBYNAME "Also try to do dns resolution using
gethostbyname if getaddrinfo fails" OFF)
option(LWS_WITHOUT_BUILTIN_SHA1 "Don't build the lws sha-1 (eg, because openssl
will provide it" OFF)
option(LWS_WITH_LATENCY "Build latency measuring code into the library" OFF)

```

```

option(LWS_WITHOUT_DAEMONIZE "Don't build the daemonization api" ON)
option(LWS_SSL_SERVER_WITH_ECDH_CERT "Include SSL server use ECDH certificate" OFF)
option(LWS_WITH_LEJP "With the Lightweight JSON Parser" ON)
option(LWS_WITH_SQLITE3 "Require SQLITE3 support" OFF)
option(LWS_WITH_STRUCT_JSON "Generic struct serialization to and from JSON" ON)
option(LWS_WITH_STRUCT_SQLITE3 "Generic struct serialization to and from SQLITE3"
OFF)
option(LWS_WITH_SMTP "Provide SMTP support" OFF)
if (WIN32 OR LWS_WITH_ESP32)
option(LWS_WITH_DIR "Directory scanning api support" OFF)
option(LWS_WITH_LEJP_CONF "With LEJP configuration parser as used by lws" OFF)
else()
option(LWS_WITH_DIR "Directory scanning api support" ON)
option(LWS_WITH_LEJP_CONF "With LEJP configuration parser as used by lws" ON)
endif()
option(LWS_WITH_NO_LOGS "Disable all logging from being compiled in" OFF)
option(LWS_AVOID_SIGPIPE_IGNORE "Android 7+ reportedly needs this" OFF)
option(LWS_WITH_STATS "Keep statistics of lws internal operations" OFF)
option(LWS_WITH_JOSE "JSON Web Signature / Encryption / Keys (RFC7515/6/) API" OFF)
option(LWS_WITH_GENCRYPTO "Enable support for Generic Crypto apis independent of
TLS backend" OFF)
option(LWS_WITH_SELFTESTS "Selftests run at context creation" OFF)
option(LWS_WITH_GCOV "Build with gcc gcov coverage instrumentation" OFF)
option(LWS_WITH_EXPORT_LWSTARGETS "Export libwebsockets CMake targets. Disable if
they conflict with an outer cmake project." ON)
option(LWS_REPRODUCIBLE "Build libwebsockets reproducible. It removes the build
user and hostname from the build" ON)
option(LWS_WITH_MINIMAL_EXAMPLES "Also build the normally standalone minimal
examples, for QA" OFF)
option(LWS_WITH_LWSAC "lwsac Chunk Allocation api" ON)
option(LWS_WITH_CUSTOM_HEADERS "Store and allow querying custom HTTP headers (H1
only)" ON)
option(LWS_WITH_DISKCACHE "Hashed cache directory with lazy LRU deletion to size
limit" OFF)
option(LWS_WITH_ASAN "Build with gcc runtime sanitizer options enabled (needs
libasan)" OFF)
option(LWS_WITH_DIR "Directory scanning api support" OFF)
option(LWS_WITH_LEJP_CONF "With LEJP configuration parser as used by lws" OFF)
option(LWS_WITH_ZLIB "Include zlib support (required for extensions)" OFF)
option(LWS_WITH_BUNDLED_ZLIB "Use bundled zlib version (Windows only)" $
{LWS_WITH_BUNDLED_ZLIB_DEFAULT})
option(LWS_WITH_MINIZ "Use miniz instead of zlib" OFF)
option(LWS_WITH_DEPRECATED_LWS_DLL "Migrate to lws_dll2 instead ASAP" OFF)
option(LWS_WITH_SEQUENCER "lws_seq_t support" ON)
option(LWS_WITH_EXTERNAL_POLL "Support external POLL integration using callback
messages (not recommended)" OFF)
option(LWS_WITH_LWS_DSH "Support lws_dsh_t Disordered Shared Heap" OFF)
#
# to use miniz, enable both LWS_WITH_ZLIB and LWS_WITH_MINIZ
#
# End of user settings
#

# Workaround for ESP-IDF
# Detect ESP_PLATFORM environment flag, if exist, set LWS_WITH_ESP32.
# Otherwise the user may not be able to run configuration ESP-IDF in the first
time.
if(ESP_PLATFORM)
    message(STATUS "ESP-IDF enabled")

```

```

        set(LWS_WITH_ESP32 ON)
else()
    set(LWS_WITH_ESP32_HELPER OFF)
endif()

if (WIN32 OR LWS_WITH_ESP32)
    message(STATUS "No LWS_WITH_DIR and LWS_WITH_DIR")
    set(LWS_WITH_DIR OFF)
    set(LWS_WITH_LEJP_CONF OFF)
    message("LWS_WITH_DIR ${LWS_WITH_DIR}")
else()
    message(STATUS "Compiled with LWS_WITH_DIR and LWS_WITH_DIR")
    set(LWS_WITH_DIR ON)
    set(LWS_WITH_LEJP_CONF ON)
endif()

if (LWS_FOR_GITOHASHI)
    set(LWS_WITH_THREADPOOL 1)
    set(LWS_WITH_HTTP2 1)
    set(LWS_UNIX_SOCKET 1)
    set(LWS_WITH_HTTP_PROXY 1)
    set(LWS_WITH_FTS 1)
    set(LWS_WITH_DISKCACHE 1)
    set(LWS_WITH_LWSAC 1)
    set(LWS_WITH_LEJP_CONF 1)
endif()

if(LWS_WITH_DISTRO_RECOMMENDED)
    set(LWS_WITH_HTTP2 1)
    set(LWS_WITH_LWSWS 1)
    set(LWS_WITH_CGI 1)
    set(LWS_IPV6 1)
    set(LWS_WITH_ZIP_FOPS 1)
    set(LWS_WITH_SOCKS5 1)
    set(LWS_WITH_RANGES 1)
    set(LWS_WITH_ACME 1)
    set(LWS_WITH_SERVER_STATUS 1)
    set(LWS_WITH_LIBUV 1)
    set(LWS_WITH_LIBEV 1)
    # libev + libevent cannot coexist at build-time
    set(LWS_WITH_LIBEVENT 0)
    set(LWS_WITHOUT_EXTENSIONS 0)
    set(LWS_ROLE_DBUS 1)
    set(LWS_WITH_FTS 1)
    set(LWS_WITH_THREADPOOL 1)
    set(LWS_UNIX_SOCKET 1)
    set(LWS_WITH_HTTP_PROXY 1)
    set(LWS_WITH_DISKCACHE 1)
    set(LWS_WITH_LWSAC 1)
    set(LWS_WITH_LEJP_CONF 1)
    set(LWS_WITH_PLUGINS 1)
    set(LWS_ROLE_RAW_PROXY 1)
    set(LWS_WITH_GENCRYPTO 1)
    set(LWS_WITH_JOSE 1)
endif()

if (NOT LWS_WITH_NETWORK)
    set(LWS_ROLE_H1 0)
    set(LWS_ROLE_WS 0)

```

```

    set(LWS_ROLE_RAW 0)
    set(LWS_WITHOUT_EXTENSIONS 1)
    set(LWS_WITHOUT_SERVER 1)
    set(LWS_WITHOUT_CLIENT 1)
    set(LWS_WITH_HTTP2 0)
    set(LWS_WITH_SOCKS5 0)
    set(LWS_UNIX_SOCKET 0)
    set(LWS_WITH_HTTP_PROXY 0)
    set(LWS_WITH_PLUGINS 0)
    set(LWS_WITH_LWSWS 0)
    set(LWS_WITH_CGI 0)
    set(LWS_ROLE_RAW_PROXY 0)
    set(LWS_WITH_PEER_LIMITS 0)
    set(LWS_WITH_GENERIC_SESSIONS 0)
    set(LWS_WITH_HTTP_STREAM_COMPRESSION 0)
    set(LWS_WITH_HTTP_BROTLI 0)
    set(LWS_WITH_POLL 0)
    set(LWS_WITH_SEQUENCER 0)
    set(LWS_ROLE_DBUS 0)
    set(LWS_WITH_LWS_DSH 0)
endif()

if (LWS_WITH_STRUCT_SQLITE3)
    set(LWS_WITH_SQLITE3 1)
endif()

# do you care about this? Then send me a patch where it disables it on travis
# but allows it on APPLE
if (APPLE)
    set(LWS_ROLE_DBUS 0)
endif()

if(NOT DEFINED CMAKE_BUILD_TYPE)
    set(CMAKE_BUILD_TYPE Release CACHE STRING "Build type")
endif()

# microsoft... that's why you can't have nice things

if (WIN32 OR LWS_WITH_ESP32)
    set(LWS_UNIX_SOCKET 0)
endif()

if (LWS_WITH_ESP32)
    set(LWS_WITH_LWSAC 0)
    set(LWS_WITH_FTS 0)
endif()

project(libwebsockets C)

set(PACKAGE "libwebsockets")
set(CPACK_PACKAGE_NAME "${PACKAGE}")
set(CPACK_PACKAGE_VERSION_MAJOR "3")
set(CPACK_PACKAGE_VERSION_MINOR "2")
set(CPACK_PACKAGE_VERSION_PATCH "0")
set(CPACK_PACKAGE_RELEASE 1)
set(CPACK_GENERATOR "RPM")
set(CPACK_PACKAGE_VERSION "${CPACK_PACKAGE_VERSION_MAJOR}.${CPACK_PACKAGE_VERSION_MINOR}.${CPACK_PACKAGE_VERSION_PATCH}")
set(CPACK_PACKAGE_VENDOR "andy@warmcat.com")

```

```

set(CPACK_PACKAGE_CONTACT "andy@warmcat.com")
set(CPACK_PACKAGE_DESCRIPTION_SUMMARY "${PACKAGE} ${PACKAGE_VERSION}")
set(SOVERSION "15")
if(NOT CPACK_GENERATOR)
    if(UNIX)
        set(CPACK_GENERATOR "TGZ")
    else()
        set(CPACK_GENERATOR "ZIP")
    endif()
endif()
set(CPACK_SOURCE_GENERATOR "TGZ")
set(CPACK_SOURCE_PACKAGE_FILE_NAME "${CPACK_PACKAGE_NAME}-${CPACK_PACKAGE_VERSION}")
set(VERSION "${CPACK_PACKAGE_VERSION}")

set(LWS_LIBRARY_VERSION ${CPACK_PACKAGE_VERSION})
set(LWS_LIBRARY_VERSION_MAJOR ${CPACK_PACKAGE_VERSION_MAJOR})
set(LWS_LIBRARY_VERSION_MINOR ${CPACK_PACKAGE_VERSION_MINOR})
set(LWS_LIBRARY_VERSION_PATCH ${CPACK_PACKAGE_VERSION_PATCH})

set(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} "${PROJECT_SOURCE_DIR}/cmake/")

message(STATUS "CMAKE_TOOLCHAIN_FILE='${CMAKE_TOOLCHAIN_FILE}'")

if(WIN32)
    configure_file(${CMAKE_CURRENT_SOURCE_DIR}/win32port/version.rc.in $
{CMAKE_CURRENT_BINARY_DIR}/win32port/version.rc @ONLY)
    set(RESOURCES ${CMAKE_CURRENT_BINARY_DIR}/win32port/version.rc)
endif()

# Try to find the current Git hash.
find_package(Git)
if(GIT_EXECUTABLE)
    execute_process(
        WORKING_DIRECTORY "${CMAKE_CURRENT_SOURCE_DIR}"
        COMMAND "${GIT_EXECUTABLE}" describe --tags
        OUTPUT_VARIABLE GIT_HASH
        OUTPUT_STRIP_TRAILING_WHITESPACE
    )
    set(LWS_BUILD_HASH ${GIT_HASH})

    # append the build user and hostname
    if(NOT LWS_REPRODUCIBLE)
        execute_process(
            WORKING_DIRECTORY "${CMAKE_CURRENT_SOURCE_DIR}"
            COMMAND "whoami"
            OUTPUT_VARIABLE GIT_USER
            OUTPUT_STRIP_TRAILING_WHITESPACE
        )
        execute_process(
            WORKING_DIRECTORY "${CMAKE_CURRENT_SOURCE_DIR}"
            COMMAND "hostname"
            OUTPUT_VARIABLE GIT_HOST
            OUTPUT_STRIP_TRAILING_WHITESPACE
        )
        string(REGEX REPLACE "([\\`])" "\\1\\\\" GIT_USER $
{GIT_USER})
        set(LWS_BUILD_HASH ${GIT_USER}@${GIT_HOST}-${GIT_HASH})
    endif()
endif()

```

```

endif()

message("Git commit hash: ${LWS_BUILD_HASH}")
endif()

# translate old functionality enables to set up ROLE enables so nothing changes
if (LWS_WITH_HTTP2 AND LWS_WITHOUT_SERVER)
    set(LWS_WITH_HTTP2 0)
    message("HTTP2 disabled due to LWS_WITHOUT_SERVER")
endif()

if (LWS_WITH_HTTP2)
    set(LWS_ROLE_H2 1)
endif()
if (LWS_WITH_CGI)
    set(LWS_ROLE_CGI 1)
endif()

if (NOT LWS_ROLE_WS)
    set(LWS_WITHOUT_EXTENSIONS 1)
endif()

include_directories(include plugins)

if (LWS_WITH_LWSWS)
    message(STATUS "LWS_WITH_LWSWS --> Enabling LWS_WITH_PLUGINS and LWS_WITH_LIBUV")
    set(LWS_WITH_PLUGINS 1)
    set(LWS_WITH_LIBUV 1)
    set(LWS_WITH_ACCESS_LOG 1)
    set(LWS_WITH_SERVER_STATUS 1)
    set(LWS_WITH_LEJP 1)
    set(LWS_WITH_LEJP_CONF 1)
    set(LWS_WITH_PEER_LIMITS 1)
    set(LWS_ROLE_RAW_PROXY 1)
endif()

# sshd plugin
if (LWS_WITH_PLUGINS)
    set(LWS_WITH_GENCRYPTO 1)
endif()

if (LWS_ROLE_RAW_PROXY)
    set (LWS_WITHOUT_CLIENT 0)
    set (LWS_WITHOUT_SERVER 0)
endif()

if (LWS_WITH_ACME)
    set (LWS_WITHOUT_CLIENT 0)
    set (LWS_WITHOUT_SERVER 0)
    set (LWS_WITH_JOSE 1)
endif()

if (LWS_WITH_JOSE)
    set(LWS_WITH_LEJP 1)
    set(LWS_WITH_GENCRYPTO 1)
endif()

if (LWS_WITH_PLUGINS AND NOT LWS_WITH_LIBUV)
    message(STATUS "LWS_WITH_PLUGINS --> Enabling LWS_WITH_LIBUV")

```

```

    set(LWS_WITH_LIBUV 1)
endif()

if (LWS_WITH_PLUGINS OR LWS_WITH_CGI)
    # sshd plugin
    set(LWS_WITH_GENCRYPTO 1)
endif()

if (LWS_WITH_GENERIC_SESSIONS)
    set(LWS_WITH_SQLITE3 1)
    set(LWS_WITH_SMTP 1)
    set(LWS_WITH_STRUCT_SQLITE3 1)
endif()

if (LWS_WITH_ESP32)
    set(LWS_WITH_SHARED OFF)
    set(LWS_WITH_MBEDTLS ON)
    # set(LWS_WITHOUT_CLIENT ON)
    set(LWS_WITHOUT_TESTAPPS ON)
    set(LWS_WITHOUT_EXTENSIONS ON)
    set(LWS_WITH_PLUGINS OFF)
    set(LWS_WITH_RANGES ON)
    # this implies no pthreads in the lib
    set(LWS_MAX_SMP 1)
    set(LWS_HAVE_MALLOC 1)
    set(LWS_HAVE_REALLOC 1)
    set(LWS_HAVE_GETIFADDRS 1)
    set(LWS_WITH_ZIP_FOPS 1)
    set(LWS_WITH_CUSTOM_HEADERS 0)
endif()

if (WIN32)
    set(LWS_MAX_SMP 1)
    set(LWS_WITH_THREADPOOL 0)
endif()

if (LWS_WITHOUT_SERVER)
    set(LWS_WITH_LWSWS OFF)
endif()

if (LWS_WITH_LEJP_CONF)
    set(LWS_WITH_DIR 1)
endif()

# confirm H1 relationships

if (NOT LWS_ROLE_H1 AND LWS_ROLE_H2)
    message(FATAL_ERROR "H2 requires LWS_ROLE_H1")
endif()

if (NOT LWS_ROLE_H1 AND LWS_ROLE_WS)
    message(FATAL_ERROR "WS requires LWS_ROLE_H1")
endif()

if (NOT LWS_ROLE_H1 AND LWS_ROLE_CGI)
    message(FATAL_ERROR "CGI requires LWS_ROLE_H1")
endif()

```



```

# confirm HTTP relationships

if (NOT LWS_ROLE_H1 AND NOT LWS_ROLE_H2 AND LWS_WITH_HTTP_PROXY)
    message(FATAL_ERROR "LWS_WITH_LWSWS requires LWS_ROLE_H1")
endif()

if (NOT LWS_ROLE_H1 AND NOT LWS_ROLE_H2 AND LWS_WITH_HTTP_PROXY)
    message(FATAL_ERROR "LWS_WITH_HTTP_PROXY requires LWS_ROLE_H1")
endif()

if (NOT LWS_ROLE_H1 AND NOT LWS_ROLE_H2 AND LWS_WITH_RANGES)
    message(FATAL_ERROR "LWS_WITH_RANGES requires LWS_ROLE_H1")
endif()

if (NOT LWS_ROLE_H1 AND NOT LWS_ROLE_H2 AND LWS_WITH_ACCESS_LOG)
    message(FATAL_ERROR "LWS_WITH_ACCESS_LOG requires LWS_ROLE_H1")
endif()

if (LWS_WITH_HTTP_PROXY AND (LWS_WITHOUT_CLIENT OR LWS_WITHOUT_SERVER))
    message("You have to enable both client and server for http proxy")
    set(LWS_WITH_HTTP_PROXY 0)
endif()

# Allow the user to override installation directories.
set(LWS_INSTALL_LIB_DIR      lib CACHE PATH "Installation directory for
libraries")
set(LWS_INSTALL_BIN_DIR      bin CACHE PATH "Installation directory for
executables")
set(LWS_INSTALL_INCLUDE_DIR  include CACHE PATH "Installation directory for header
files")
set(LWS_INSTALL_EXAMPLES_DIR bin CACHE PATH "Installation directory for example
files")

# Allow the user to use the old CyaSSL options/library in stead of wolfSSL
if (LWS_WITH_CYASSL AND LWS_WITH_WOLFSSL)
    message(FATAL_ERROR "LWS_WITH_CYASSL and LWS_WITH_WOLFSSL are mutually
exclusive!")
endif()
if (LWS_WITH_CYASSL)
    # Copy CyaSSL options to the wolfSSL options
    set(LWS_WITH_WOLFSSL ${LWS_WITH_CYASSL} CACHE BOOL "Use wolfSSL/CyaSSL
instead of OpenSSL" FORCE)
    set(LWS_WOLFSSL_LIBRARIES ${LWS_CYASSL_LIBRARIES} CACHE PATH "Path to
wolfSSL/CyaSSL libraries" FORCE)
    set(LWS_WOLFSSL_INCLUDE_DIRS ${LWS_CYASSL_INCLUDE_DIRS} CACHE PATH "Path to
wolfSSL/CyaSSL header files" FORCE)
endif()

if (NOT (LWS_WITH_STATIC OR LWS_WITH_SHARED))
    message(FATAL_ERROR "Makes no sense to compile with neither static nor shared
libraries.")
endif()

if (NOT LWS_WITHOUT_EXTENSIONS OR LWS_WITH_ZIP_FOPS)
    set(LWS_WITH_ZLIB 1)
endif()

# if you gave LWS_WITH_MINIZ, point to MINIZ here if not found

```

```
# automatically
```

```
set(LWS_ZLIB_LIBRARIES CACHE PATH "Path to the zlib/miniz library")
set(LWS_ZLIB_INCLUDE_DIRS CACHE PATH "Path to the zlib/miniz include directory")
set(LWS_OPENSSL_LIBRARIES CACHE PATH "Path to the OpenSSL library")
set(LWS_OPENSSL_INCLUDE_DIRS CACHE PATH "Path to the OpenSSL include directory")
set(LWS_WOLFSSL_LIBRARIES CACHE PATH "Path to the wolfSSL library")
set(LWS_WOLFSSL_INCLUDE_DIRS CACHE PATH "Path to the wolfSSL include directory")
set(LWS_LIBEV_LIBRARIES CACHE PATH "Path to the libev library")
set(LWS_LIBEV_INCLUDE_DIRS CACHE PATH "Path to the libev include directory")
set(LWS_LIBUV_LIBRARIES CACHE PATH "Path to the libuv library")
set(LWS_LIBUV_INCLUDE_DIRS CACHE PATH "Path to the libuv include directory")
set(LWS_SQLITE3_LIBRARIES CACHE PATH "Path to the sqlite3 library")
set(LWS_SQLITE3_INCLUDE_DIRS CACHE PATH "Path to the sqlite3 include directory")
set(LWS_LIBEVENT_INCLUDE_DIRS CACHE PATH "Path to the libevent include directory")
set(LWS_LIBEVENT_LIBRARIES CACHE PATH "Path to the libevent library")
```

```
if (NOT LWS_WITH_SSL)
    set(LWS_WITHOUT_BUILTIN_SHA1 OFF)
endif()
```

```
if (LWS_WITH_BORINGSSL)
    # boringssl deprecated EVP_PKEY
    set (LWS_WITH_GENHASH OFF)
endif()
```

```
if (LWS_WITH_SSL AND NOT LWS_WITH_WOLFSSL AND NOT LWS_WITH_MBEDTLS)
    if ("${LWS_OPENSSL_LIBRARIES}" STREQUAL "" OR "${LWS_OPENSSL_INCLUDE_DIRS}"
STREQUAL "")
        else()
            if (NOT LWS_WITH_ESP32)
                set(OPENSSL_LIBRARIES ${LWS_OPENSSL_LIBRARIES})
            endif()
            set(OPENSSL_INCLUDE_DIRS ${LWS_OPENSSL_INCLUDE_DIRS})
            set(OPENSSL_FOUND 1)
        endif()
    endif()
endif()
```

```
if (LWS_WITH_SSL AND LWS_WITH_WOLFSSL)
    if ("${LWS_WOLFSSL_LIBRARIES}" STREQUAL "" OR "${LWS_WOLFSSL_INCLUDE_DIRS}"
STREQUAL "")
        if (NOT WOLFSSL_FOUND)
            if (LWS_WITH_CYASSL)
                message(FATAL_ERROR "You must set LWS_CYASSL_LIBRARIES and
LWS_CYASSL_INCLUDE_DIRS when LWS_WITH_CYASSL is turned on.")
            else()
                message(FATAL_ERROR "You must set LWS_WOLFSSL_LIBRARIES and
LWS_WOLFSSL_INCLUDE_DIRS when LWS_WITH_WOLFSSL is turned on.")
            endif()
        endif()
    else()
        set(WOLFSSL_LIBRARIES ${LWS_WOLFSSL_LIBRARIES})
        set(WOLFSSL_INCLUDE_DIRS ${LWS_WOLFSSL_INCLUDE_DIRS})
        set(WOLFSSL_FOUND 1)
    endif()
    set(USE_WOLFSSL 1)
    set(LWS_WITH_TLS 1)
    if (LWS_WITH_CYASSL)
```

```

        set(USE_OLD_CYASSL 1)
    endif()
endif()

if (LWS_WITH_SSL AND LWS_WITH_MBEDTLS)
    if ("${LWS_MBEDTLS_LIBRARIES}" STREQUAL "" OR "${LWS_MBEDTLS_INCLUDE_DIRS}"
STREQUAL "" AND NOT LWS_WITH_ESP32)

        find_path(LWS_MBEDTLS_INCLUDE_DIRS mbedtls/ssl.h)

        find_library(MBEDTLS_LIBRARY mbedtls)
        find_library(MBEDX509_LIBRARY mbedx509)
        find_library(MBEDCRYPTO_LIBRARY mbedcrypto)

        set(LWS_MBEDTLS_LIBRARIES "${MBEDTLS_LIBRARY}" "${MBEDX509_LIBRARY}" "${
MBEDCRYPTO_LIBRARY}")

        include(FindPackageHandleStandardArgs)
        find_package_handle_standard_args(MBEDTLS DEFAULT_MSG
            LWS_MBEDTLS_INCLUDE_DIRS MBEDTLS_LIBRARY MBEDX509_LIBRARY
MBEDCRYPTO_LIBRARY)

        mark_as_advanced(LWS_MBEDTLS_INCLUDE_DIRS MBEDTLS_LIBRARY
MBEDX509_LIBRARY MBEDCRYPTO_LIBRARY)

        if ("${LWS_MBEDTLS_LIBRARIES}" STREQUAL "" OR "${
LWS_MBEDTLS_INCLUDE_DIRS}" STREQUAL "")
            message(FATAL_ERROR "You must set LWS_MBEDTLS_LIBRARIES and
LWS_MBEDTLS_INCLUDE_DIRS when LWS_WITH_MBEDTLS is turned on.")
        endif()
    endif()
    set(MBEDTLS_LIBRARIES ${LWS_MBEDTLS_LIBRARIES})
    set(MBEDTLS_INCLUDE_DIRS ${LWS_MBEDTLS_INCLUDE_DIRS})
    set(MBEDTLS_FOUND 1)
    set(USE_MBEDTLS 1)
endif()

if (LWS_WITH_HTTP_STREAM_COMPRESSION)
    set(LWS_WITH_ZLIB 1)
endif()

if (LWS_WITH_ZLIB AND NOT LWS_WITH_BUNDLED_ZLIB)
    if ("${LWS_ZLIB_LIBRARIES}" STREQUAL "" OR "${LWS_ZLIB_INCLUDE_DIRS}"
STREQUAL "")
    else()
        set(ZLIB_LIBRARIES ${LWS_ZLIB_LIBRARIES})
        set(ZLIB_INCLUDE_DIRS ${LWS_ZLIB_INCLUDE_DIRS})
        set(ZLIB_FOUND 1)
    endif()
endif()

if (LWS_WITH_LIBEV)
    if ("${LWS_LIBEV_LIBRARIES}" STREQUAL "" OR "${LWS_LIBEV_INCLUDE_DIRS}"
STREQUAL "")
    else()
        set(LIBEV_LIBRARIES ${LWS_LIBEV_LIBRARIES})
        set(LIBEV_INCLUDE_DIRS ${LWS_LIBEV_INCLUDE_DIRS})
        set(LIBEV_FOUND 1)
    endif()
endif()

```

```

endif()

if (LWS_WITH_LIBUV)
    if ("${LWS_LIBUV_LIBRARIES}" STREQUAL "" OR "${LWS_LIBUV_INCLUDE_DIRS}"
STREQUAL "")
        else()
            set(LIBUV_LIBRARIES ${LWS_LIBUV_LIBRARIES})
            set(LIBUV_INCLUDE_DIRS ${LWS_LIBUV_INCLUDE_DIRS})
            set(LIBUV_FOUND 1)
        endif()
    endif()

endif()

if (LWS_WITH_LIBEVENT)
    if ("${LWS_LIBEVENT_LIBRARIES}" STREQUAL "" OR "${LWS_LIBEVENT_INCLUDE_DIRS}"
STREQUAL "")
        else()
            set(LIBEVENT_LIBRARIES ${LWS_LIBEVENT_LIBRARIES})
            set(LIBEVENT_INCLUDE_DIRS ${LWS_LIBEVENT_INCLUDE_DIRS})
            set(LIBEVENT_FOUND 1)
        endif()
    endif()

endif()

if (LWS_WITH_SQLITE3)
    if ("${LWS_SQLITE3_LIBRARIES}" STREQUAL "" OR "${LWS_SQLITE3_INCLUDE_DIRS}"
STREQUAL "")
        else()
            set(SQLITE3_LIBRARIES ${LWS_SQLITE3_LIBRARIES})
            set(SQLITE3_INCLUDE_DIRS ${LWS_SQLITE3_INCLUDE_DIRS})
            set(SQLITE3_FOUND 1)
        endif()
    endif()

endif()

if (LWS_WITH_LIBEV AND LWS_WITH_LIBEVENT)
    message(FATAL_ERROR "Sorry libev and libevent conflict with each others'
namespace, you can only have one or the other")
endif()

# The base dir where the test-apps look for the SSL certs.
set(LWS_OPENSSL_CLIENT_CERTS ../share CACHE PATH "Server SSL certificate
directory")
if (WIN32)
    set(LWS_OPENSSL_CLIENT_CERTS . CACHE PATH "Client SSL certificate directory")

    if (LWS_UNIX_SOCKET)
        set(LWS_UNIX_SOCKET OFF)
        message(WARNING "Windows does not support UNIX domain sockets")
    endif()
else()
    set(LWS_OPENSSL_CLIENT_CERTS /etc/pki/tls/certs/ CACHE PATH "Client SSL
certificate directory")
endif()

# LWS_OPENSSL_SUPPORT deprecated... use LWS_WITH_TLS
if (LWS_WITH_SSL OR LWS_WITH_MBEDTLS)
    set(LWS_OPENSSL_SUPPORT 1)
    set(LWS_WITH_TLS 1)
endif()

```

```

if (LWS_SSL_CLIENT_USE_OS_CA_CERTS)
    set(LWS_SSL_CLIENT_USE_OS_CA_CERTS 1)
endif()

if (LWS_WITH_LATENCY)
    set(LWS_LATENCY 1)
endif()

if (LWS_WITHOUT_DAEMONIZE OR WIN32)
    set(LWS_NO_DAEMONIZE 1)
endif()

if (LWS_WITHOUT_SERVER)
    set(LWS_NO_SERVER 1)
endif()

if (LWS_WITHOUT_CLIENT)
    set(LWS_NO_CLIENT 1)
endif()

if (LWS_WITH_LIBEV)
    set(LWS_WITH_LIBEV 1)
endif()

if (LWS_WITH_LIBUV)
    set(LWS_WITH_LIBUV 1)
endif()

if (LWS_WITH_LIBEVENT)
    set(LWS_WITH_LIBEVENT 1)
endif()

if (LWS_IPV6)
    set(LWS_WITH_IPV6 1)
endif()

if (LWS_UNIX_SOCK)
    set(LWS_WITH_UNIX_SOCK 1)
endif()

if (LWS_WITH_HTTP2)
    set(LWS_WITH_HTTP2 1)
endif()

if ("${LWS_MAX_SMP}" STREQUAL "")
    set(LWS_MAX_SMP 1)
endif()

# using any abstract protocol enables LWS_WITH_ABSTRACT

if (LWS_WITH_SMTP)
    set(LWS_WITH_ABSTRACT 1)
endif()

if (MINGW)
    set(LWS_MINGW_SUPPORT 1)
    set(CMAKE_C_FLAGS "-D__USE_MINGW_ANSI_STDIO ${CMAKE_C_FLAGS}")

```

```

        add_definitions(-DWINVER=0x0601 -D_WIN32_WINNT=0x0601)
endif()

if (LWS_SSL_SERVER_WITH_ECDH_CERT)
    set(LWS_SSL_SERVER_WITH_ECDH_CERT 1)
endif()

include_directories("${PROJECT_BINARY_DIR}")

include(CheckCSourceCompiles)

# Check for different inline keyword versions.
foreach(KEYWORD "inline" "__inline__" "__inline")
    set(CMAKE_REQUIRED_DEFINITIONS "-DKEYWORD=${KEYWORD}")
    CHECK_C_SOURCE_COMPILES(
        "
        #include <stdio.h>
        static KEYWORD void a() {}
        int main(int argc, char **argv) { a(); return 0; }
        " LWS_HAVE_${KEYWORD})
endforeach()

if (NOT LWS_HAVE_inline)
    if (LWS_HAVE__inline__)
        set(inline __inline__)
    elseif(LWS_HAVE__inline)
        set(inline __inline)
    endif()
endif()

# Put the libraries and binaries that get built into directories at the
# top of the build tree rather than in hard-to-find leaf directories.
SET(CMAKE_RUNTIME_OUTPUT_DIRECTORY "${PROJECT_BINARY_DIR}/bin")
SET(CMAKE_LIBRARY_OUTPUT_DIRECTORY "${PROJECT_BINARY_DIR}/lib")
SET(CMAKE_ARCHIVE_OUTPUT_DIRECTORY "${PROJECT_BINARY_DIR}/lib")

SET(LWS_INSTALL_PATH "${CMAKE_INSTALL_PREFIX}")

# Put absolute path of dynamic libraries into the object code. Some
# architectures, notably Mac OS X, need this.
SET(CMAKE_INSTALL_NAME_DIR "${CMAKE_INSTALL_PREFIX}/${LWS_INSTALL_LIB_DIR}${LIB_SUFFIX}")

include(CheckFunctionExists)
include(CheckSymbolExists)
include(CheckIncludeFile)
include(CheckIncludeFiles)
include(CheckLibraryExists)
include(CheckTypeSize)
include(CheckCSourceCompiles)

if (LWS_WITHOUT_BUILTIN_SHA1)
    set(LWS_SHA1_USE_OPENSSL_NAME 1)
endif()

if (HAIKU)
    set(CMAKE_REQUIRED_LIBRARIES network)
endif()

```

```

CHECK_C_SOURCE_COMPILES(
    "#include <malloc.h>
    int main(int argc, char **argv) { return malloc_trim(0); }
    " LWS_HAVE_MALLOC_TRIM)
CHECK_C_SOURCE_COMPILES(
    "#include <malloc.h>
    int main(int argc, char **argv) { return (int)malloc_usable_size((void
*)0); }
    " LWS_HAVE_MALLOC_USABLE_SIZE)

CHECK_FUNCTION_EXISTS(fork LWS_HAVE_FORK)
CHECK_FUNCTION_EXISTS(getenv LWS_HAVE_GETENV)
CHECK_FUNCTION_EXISTS(malloc LWS_HAVE_MALLOC)
CHECK_FUNCTION_EXISTS(memset LWS_HAVE_MEMSET)
CHECK_FUNCTION_EXISTS(realloc LWS_HAVE_REALLOC)
CHECK_FUNCTION_EXISTS(socket LWS_HAVE_SOCKET)
CHECK_FUNCTION_EXISTS(strerror LWS_HAVE_STRERROR)
CHECK_FUNCTION_EXISTS(vfork LWS_HAVE_VFORK)
CHECK_FUNCTION_EXISTS(execvpe LWS_HAVE_EXECPVE)
CHECK_FUNCTION_EXISTS(getifaddrs LWS_HAVE_GETIFADDRS)
CHECK_FUNCTION_EXISTS(snprintf LWS_HAVE_SNPRINTF)
CHECK_FUNCTION_EXISTS(_snprintf LWS_HAVE__SNPRINTF)
CHECK_FUNCTION_EXISTS(_vsnprintf LWS_HAVE__VSNPRINTF)
CHECK_FUNCTION_EXISTS(getloadavg LWS_HAVE_GETLOADAVG)
CHECK_FUNCTION_EXISTS(atoll LWS_HAVE_ATOLL)
CHECK_FUNCTION_EXISTS(_atoi64 LWS_HAVE__ATOI64)
CHECK_FUNCTION_EXISTS(_stat32i64 LWS_HAVE__STAT32I64)
CHECK_FUNCTION_EXISTS(clock_gettime LWS_HAVE_CLOCK_GETTIME)

if (NOT LWS_HAVE_GETIFADDRS)
    if (LWS_WITHOUT_BUILTIN_GETIFADDRS)
        message(FATAL_ERROR "No getifaddrs was found on the system. Turn off
the LWS_WITHOUT_BUILTIN_GETIFADDRS compile option to use the supplied BSD
version.")
    endif()
    set(LWS_BUILTIN_GETIFADDRS 1)
endif()

CHECK_INCLUDE_FILE(dlfcn.h LWS_HAVE_DLFCN_H)
CHECK_INCLUDE_FILE(fcntl.h LWS_HAVE_FCNTL_H)
CHECK_INCLUDE_FILE(in6addr.h LWS_HAVE_IN6ADDR_H)
CHECK_INCLUDE_FILE(memory.h LWS_HAVE_MEMORY_H)
CHECK_INCLUDE_FILE(netinet/in.h LWS_HAVE_NETINET_IN_H)
CHECK_INCLUDE_FILE(stdint.h LWS_HAVE_STDINT_H)
CHECK_INCLUDE_FILE(stdlib.h LWS_HAVE_STDLIB_H)
CHECK_INCLUDE_FILE(strings.h LWS_HAVE_STRINGS_H)
CHECK_INCLUDE_FILE(string.h LWS_HAVE_STRING_H)
CHECK_INCLUDE_FILE(sys/prctl.h LWS_HAVE_SYS_PRCTL_H)
CHECK_INCLUDE_FILE(sys/socket.h LWS_HAVE_SYS_SOCKET_H)
CHECK_INCLUDE_FILE(sys/sockio.h LWS_HAVE_SYS_SOCKIO_H)
CHECK_INCLUDE_FILE(sys/stat.h LWS_HAVE_SYS_STAT_H)
CHECK_INCLUDE_FILE(sys/types.h LWS_HAVE_SYS_TYPES_H)
CHECK_INCLUDE_FILE(unistd.h LWS_HAVE_UNISTD_H)
CHECK_INCLUDE_FILE(vfork.h LWS_HAVE_VFORK_H)
CHECK_INCLUDE_FILE(sys/capability.h LWS_HAVE_SYS_CAPABILITY_H)
CHECK_INCLUDE_FILE(malloc.h LWS_HAVE_MALLOC_H)
CHECK_INCLUDE_FILE(pthread.h LWS_HAVE_PTHREAD_H)
CHECK_INCLUDE_FILE(inttypes.h LWS_HAVE_INTTYPES_H)

```

```

CHECK_LIBRARY_EXISTS(cap cap_set_flag "" LWS_HAVE_LIBCAP)

if (LWS_ROLE_DBUS)

    if (NOT LWS_DBUS_LIB)
        set(LWS_DBUS_LIB "dbus-1")
    endif()

    CHECK_LIBRARY_EXISTS(${LWS_DBUS_LIB} dbus_connection_set_watch_functions ""
LWS_HAVE_LIBDBUS)
    if (NOT LWS_HAVE_LIBDBUS)
        message(FATAL_ERROR "Install dbus-devel, or libdbus-1-dev etc")
    endif()

    if (NOT LWS_DBUS_INCLUDE1)
        # look in fedora and debian / ubuntu place
        if (EXISTS "/usr/include/dbus-1.0")
            set(LWS_DBUS_INCLUDE1 "/usr/include/dbus-1.0")
        else()
            message(FATAL_ERROR "Set LWS_DBUS_INCLUDE1 to /usr/include/dbus-
1.0 or wherever the main dbus includes are")
        endif()
    endif()

    if (NOT LWS_DBUS_INCLUDE2)
        # look in fedora... debian / ubuntu has the ARCH in the path...
        if (EXISTS "/usr/lib64/dbus-1.0/include")
            set(LWS_DBUS_INCLUDE2 "/usr/lib64/dbus-1.0/include")
        else()
            message(FATAL_ERROR "Set LWS_DBUS_INCLUDE2 to /usr/lib/ARCH-
linux-gnu/dbus-1.0/include or wherever dbus-arch-deps.h is on your system")
        endif()
    endif()

    set(CMAKE_REQUIRED_INCLUDES ${CMAKE_REQUIRED_INCLUDES};${LWS_DBUS_INCLUDE1};$
{LWS_DBUS_INCLUDE2})

    CHECK_C_SOURCE_COMPILES("#include <dbus/dbus.h>
int main(void) {
    return 0;
}" LWS_DBUS_CHECK_OK)
endif()

if (LWS_WITH_LIBUV)
CHECK_INCLUDE_FILE(uv-version.h LWS_HAVE_UV_VERSION_H)
# libuv changed the location in 1.21.0. Retain both
# checks temporarily to ensure a smooth transition.
if (NOT LWS_HAVE_UV_VERSION_H)
    CHECK_INCLUDE_FILE(uv/version.h LWS_HAVE_NEW_UV_VERSION_H)
endif()
endif()

if (LWS_WITH_ZLIB AND NOT LWS_WITH_BUNDLED_ZLIB)
    if (LWS_WITH_MINIZ)
        CHECK_INCLUDE_FILE(miniz.h LWS_HAVE_ZLIB_H)
    else()
        CHECK_INCLUDE_FILE(zlib.h LWS_HAVE_ZLIB_H)
    endif()
endif()

```



```

endif()

# TODO: These can also be tested to see whether they actually work...
set(LWS_HAVE_WORKING_FORK LWS_HAVE_FORK)
set(LWS_HAVE_WORKING_VFORK LWS_HAVE_VFORK)

CHECK_INCLUDE_FILES("stdlib.h;stdarg.h;string.h;float.h" STDC_HEADERS)

CHECK_C_SOURCE_COMPILES("#include <stdint.h>
    int main(void) {
        intptr_t test = 1;
        return 0;
    }" LWS_HAS_INTPTR_T)

set(CMAKE_REQUIRED_FLAGS "-pthread")
CHECK_C_SOURCE_COMPILES("#define _GNU_SOURCE
    #include <pthread.h>
    int main(void) {
        pthread_t th = 0;
        pthread_setname_np(th, NULL);
        return 0;
    }" LWS_HAS_PTHREAD_SETNAME_NP)

CHECK_C_SOURCE_COMPILES("#include <stddef.h>
    #include <getopt.h>
    int main(void) {
        void *p = (void *)getopt_long;
        return p != NULL;
    }" LWS_HAS_GETOPT_LONG)

if (NOT PID_T_SIZE)
    set(pid_t int)
endif()

if (NOT SIZE_T_SIZE)
    set(size_t "unsigned int")
endif()

if (NOT LWS_HAVE_MALLOC)
    set(malloc rpl_malloc)
endif()

if (NOT LWS_HAVE_REALLOC)
    set(realloc rpl_realloc)
endif()

if (UNIX)
    execute_process(COMMAND uname -n OUTPUT_VARIABLE NODENAME)
    # Need to chomp the \n at end of output.
    string(REGEX REPLACE "[\n]+" "" NODENAME "${NODENAME}")

    if( NODENAME STREQUAL "smartos" )
        add_definitions( "-D__smartos__" )
        set(SMARTOS 1)
    endif()
endif()

if (MSVC)

```

```

        # Turn off stupid microsoft security warnings.
        add_definitions(-D_CRT_SECURE_NO_DEPRECATED -D_CRT_NONSTDC_NO_DEPRECATED)
endif(MSVC)

include_directories("${PROJECT_SOURCE_DIR}/lib")

# Group headers and sources.
# Some IDEs use this for nicer file structure.
set(HDR_PRIVATE
    lib/core/private.h)

set(HDR_PUBLIC
    "${PROJECT_SOURCE_DIR}/include/libwebsockets.h"
    "${PROJECT_BINARY_DIR}/lws_config.h"
    "${PROJECT_SOURCE_DIR}/plugins/ssh-base/include/lws-plugin-ssh.h"
)

set(SOURCES
    lib/core/alloc.c
    lib/core/buflist.c
    lib/core/context.c
    lib/core/lws_dll2.c
    lib/core/libwebsockets.c
    lib/core/logs.c
    lib/misc/base64-decode.c
    lib/core/vfs.c
    lib/misc/lws-ring.c
)

if (LWS_WITH_DEPRECATED_LWS_DLL)
    list(APPEND SOURCES
        lib/core/lws_dll.c)
endif()

if (LWS_WITH_NETWORK)
    list(APPEND SOURCES
        lib/core-net/dummy-callback.c
        lib/core-net/output.c
        lib/core-net/close.c
        lib/core-net/network.c
        lib/core-net/vhost.c
        lib/core-net/pollfd.c
        lib/core-net/service.c
        lib/core-net/sorted-usec-list.c
        lib/core-net/stats.c
        lib/core-net/wsi.c
        lib/core-net/wsi-timeout.c
        lib/core-net/adopt.c
        lib/roles/pipe/ops-pipe.c
    )

    if (LWS_WITH_LWS_DSH)
        list(APPEND SOURCES
            lib/core-net/lws-dsh.c)
    endif()

    if (LWS_WITH_SEQUENCER)
        list(APPEND SOURCES
            lib/core-net/sequencer.c)
    endif()

```

```

endif()

if (LWS_WITH_ABSTRACT)
    list(APPEND SOURCES
        lib/abstract/abstract.c
    )
    if (LWS_WITH_SEQUENCER)
        list(APPEND SOURCES
            lib/abstract/test-sequencer.c)
    endif()
endif()

if (LWS_WITH_STATS)
    list(APPEND SOURCES
        lib/core-net/stats.c
    )
endif()
endif()

if (LWS_WITH_DIR)
    list(APPEND SOURCES lib/misc/dir.c)
endif()

if (LWS_WITH_THREADPOOL AND UNIX AND LWS_HAVE_PTHREAD_H)
    list(APPEND SOURCES lib/misc/threadpool/threadpool.c)
endif()

if (LWS_ROLE_H1 OR LWS_ROLE_H2)
    list(APPEND SOURCES
        lib/roles/http/header.c
        lib/roles/http/server/parsers.c)
    if (LWS_WITH_HTTP_STREAM_COMPRESSION)
        list(APPEND SOURCES
            lib/roles/http/compression/stream.c
            lib/roles/http/compression/deflate/deflate.c)
        if (LWS_WITH_HTTP_BROTLI)
            list(APPEND SOURCES
                lib/roles/http/compression/brotli/brotli.c)
        endif()
    endif()
endif()

if (LWS_ROLE_H1)
    list(APPEND SOURCES
        lib/roles/h1/ops-h1.c)
endif()

if (LWS_ROLE_WS)
    list(APPEND SOURCES
        lib/roles/ws/ops-ws.c)
    if (NOT LWS_WITHOUT_CLIENT)
        list(APPEND SOURCES
            lib/roles/ws/client-ws.c
            lib/roles/ws/client-parser-ws.c)
    endif()
    if (NOT LWS_WITHOUT_SERVER)
        list(APPEND SOURCES
            lib/roles/ws/server-ws.c)
    endif()
endif()

```

```
endif()

if (LWS_ROLE_RAW)
    list(APPEND SOURCES
        lib/roles/raw-skt/ops-raw-skt.c
        lib/roles/raw-file/ops-raw-file.c)

    if (LWS_WITH_ABSTRACT)
        list(APPEND SOURCES
            lib/abstract/transport/raw-skt.c)
    endif()
endif()

if (LWS_ROLE_RAW_PROXY)
    list(APPEND SOURCES
        lib/roles/raw-proxy/ops-raw-proxy.c)
endif()

if (LWS_ROLE_CGI)
    list(APPEND SOURCES
        lib/roles/cgi/cgi-server.c
        lib/roles/cgi/ops-cgi.c)
endif()

if (LWS_ROLE_DBUS)
    list(APPEND SOURCES
        lib/roles/dbus/dbus.c)
endif()

if (LWS_WITH_ACCESS_LOG)
    list(APPEND SOURCES
        lib/roles/http/server/access-log.c)
endif()

if (LWS_WITH_PEER_LIMITS)
    list(APPEND SOURCES
        lib/misc/peer-limits.c)
endif()

if (LWS_WITH_LWSAC)
    list(APPEND SOURCES
        lib/misc/lwsac/lwsac.c
        lib/misc/lwsac/checked-file.c)
endif()

if (LWS_WITH_FTS)
    list(APPEND SOURCES
        lib/misc/fts/trie.c
        lib/misc/fts/trie-fd.c)
endif()

if (LWS_WITH_DISKCACHE)
    list(APPEND SOURCES
        lib/misc/diskcache.c)
endif()

if (LWS_WITH_STRUCT_JSON)
    list(APPEND SOURCES
        lib/misc/lws-struct-lejp.c)
```

```

endif()

if (LWS_WITH_STRUCT_SQLITE3)
    list(APPEND SOURCES
        lib/misc/lws-struct-sqlite.c)
endif()

if (NOT LWS_WITHOUT_CLIENT)
    list(APPEND SOURCES
        lib/core-net/connect.c
        lib/core-net/client.c
        lib/roles/http/client/client.c
        lib/roles/http/client/client-handshake.c)
endif()

if (NOT LWS_WITHOUT_SERVER)
    list(APPEND SOURCES
        lib/core-net/server.c
        lib/roles/listen/ops-listen.c)
endif()

if (LWS_WITH_MBEDTLS)
    set(LWS_WITH_SSL ON)

    include_directories(lib/tls/mbedtls/wrapper/include)
    include_directories(lib/tls/mbedtls/wrapper/include/platform)
    include_directories(lib/tls/mbedtls/wrapper/include/internal)
    include_directories(lib/tls/mbedtls/wrapper/include/openssl)

    if (LWS_WITH_NETWORK)
        list(APPEND HDR_PRIVATE
            lib/tls/mbedtls/wrapper/include/internal/ssl3.h
            lib/tls/mbedtls/wrapper/include/internal/ssl_cert.h
            lib/tls/mbedtls/wrapper/include/internal/ssl_code.h
            lib/tls/mbedtls/wrapper/include/internal/ssl_dbg.h
            lib/tls/mbedtls/wrapper/include/internal/ssl_lib.h
            lib/tls/mbedtls/wrapper/include/internal/ssl_methods.h
            lib/tls/mbedtls/wrapper/include/internal/ssl_pkey.h
            lib/tls/mbedtls/wrapper/include/internal/ssl_stack.h
            lib/tls/mbedtls/wrapper/include/internal/ssl_types.h
            lib/tls/mbedtls/wrapper/include/internal/ssl_x509.h
            lib/tls/mbedtls/wrapper/include/internal/tls1.h
            lib/tls/mbedtls/wrapper/include/internal/x509_vfy.h)

        list(APPEND HDR_PRIVATE
            lib/tls/mbedtls/wrapper/include/openssl/ssl.h)

        list(APPEND HDR_PRIVATE
            lib/tls/mbedtls/wrapper/include/platform/ssl_pm.h
            lib/tls/mbedtls/wrapper/include/platform/ssl_port.h)

        list(APPEND SOURCES
            lib/tls/mbedtls/wrapper/library/ssl_cert.c
            lib/tls/mbedtls/wrapper/library/ssl_lib.c
            lib/tls/mbedtls/wrapper/library/ssl_methods.c
            lib/tls/mbedtls/wrapper/library/ssl_pkey.c
            lib/tls/mbedtls/wrapper/library/ssl_stack.c
            lib/tls/mbedtls/wrapper/library/ssl_x509.c)
    endif()
endif()

```

```

        list(APPEND SOURCES
            lib/tls/mbedtls/wrapper/platform/ssl_pm.c
            lib/tls/mbedtls/wrapper/platform/ssl_port.c)
    endif()
endif()

if (LWS_WITH_SSL)
    list(APPEND SOURCES
        lib/tls/tls.c
    )
    if (LWS_WITH_NETWORK)
        list(APPEND SOURCES
            lib/tls/tls-network.c
        )
    endif()

    if (LWS_WITH_MBEDTLS)
        list(APPEND SOURCES
            lib/tls/mbedtls/tls.c
            lib/tls/mbedtls/x509.c
        )
        if (LWS_WITH_NETWORK)
            list(APPEND SOURCES
                lib/tls/mbedtls/ssl.c
            )
        endif()
        if (LWS_WITH_GENCRYPTO)
            list(APPEND SOURCES
                lib/tls/mbedtls/lws-genhash.c
                lib/tls/mbedtls/lws-genrsa.c
                lib/tls/mbedtls/lws-genaes.c
                lib/tls/lws-genec-common.c
                lib/tls/mbedtls/lws-genec.c
                lib/tls/mbedtls/lws-gencrypto.c
            )
        endif()
    else()
        list(APPEND SOURCES
            lib/tls/openssl/tls.c
            lib/tls/openssl/x509.c
        )
        if (LWS_WITH_NETWORK)
            list(APPEND SOURCES
                lib/tls/openssl/ssl.c
            )
        endif()
        if (LWS_WITH_GENCRYPTO)
            list(APPEND SOURCES
                lib/tls/openssl/lws-genhash.c
                lib/tls/openssl/lws-genrsa.c
                lib/tls/openssl/lws-genaes.c
                lib/tls/lws-genec-common.c
                lib/tls/openssl/lws-genec.c
                lib/tls/openssl/lws-gencrypto.c
            )
        endif()
    endif()
endif()

if (NOT LWS_WITHOUT_SERVER)

```

```

        list(APPEND SOURCES
            lib/tls/tls-server.c)
    if (LWS_WITH_MBEDTLS)
        list(APPEND SOURCES
            lib/tls/mbedtls/mbedtls-server.c)
    else()
        list(APPEND SOURCES
            lib/tls/openssl/openssl-server.c)
    endif()
endif()
if (NOT LWS_WITHOUT_CLIENT)
    list(APPEND SOURCES
        lib/tls/tls-client.c)
    if (LWS_WITH_MBEDTLS)
        list(APPEND SOURCES
            lib/tls/mbedtls/mbedtls-client.c)
    else()
        list(APPEND SOURCES
            lib/tls/openssl/openssl-client.c)
    endif()
endif()

endif()
endif()

if (NOT LWS_WITHOUT_BUILTIN_SHA1)
    list(APPEND SOURCES
        lib/misc/sha-1.c)
endif()

if (LWS_WITH_HTTP2 AND NOT LWS_WITHOUT_SERVER)
    list(APPEND SOURCES
        lib/roles/h2/http2.c
        lib/roles/h2/hpack.c
        lib/roles/h2/ops-h2.c)
endif()
# select the active platform files

if (WIN32)
    list(APPEND SOURCES
        lib/plat/windows/windows-fds.c
        lib/plat/windows/windows-file.c
        lib/plat/windows/windows-init.c
        lib/plat/windows/windows-misc.c
        lib/plat/windows/windows-pipe.c
        lib/plat/windows/windows-plugins.c
        lib/plat/windows/windows-service.c
        lib/plat/windows/windows-sockets.c
    )
else()

    if (LWS_PLAT_OPTEE)
        list(APPEND SOURCES
            lib/plat/optee/lws-plat-optee.c
        )
        if (LWS_WITH_NETWORK)
            list(APPEND SOURCES
                lib/plat/optee/network.c
            )
        endif()
    endif()
endif()

```

```

else()
    if (LWS_WITH_ESP32)
        list(APPEND SOURCES
            lib/plat/esp32/esp32-fds.c
            lib/plat/esp32/esp32-file.c
            lib/plat/esp32/esp32-init.c
            lib/plat/esp32/esp32-misc.c
            lib/plat/esp32/esp32-pipe.c
            lib/plat/esp32/esp32-service.c
            lib/plat/esp32/esp32-sockets.c
            lib/misc/romfs.c)
        if(LWS_WITH_ESP32_HELPER)
            list(APPEND SOURCES lib/plat/esp32/esp32-helpers.c)
        endif()
    else()
        set(LWS_PLAT_UNIX 1)
        list(APPEND SOURCES
            lib/plat/unix/unix-caps.c
            lib/plat/unix/unix-file.c
            lib/plat/unix/unix-misc.c
            lib/plat/unix/unix-init.c
        )
        if (LWS_WITH_NETWORK)
            list(APPEND SOURCES
                lib/plat/unix/unix-pipe.c
                lib/plat/unix/unix-service.c
                lib/plat/unix/unix-sockets.c
                lib/plat/unix/unix-fds.c
            )
        endif()

        if (LWS_WITH_PLUGINS AND LWS_WITH_LIBUV)
            list(APPEND SOURCES lib/plat/unix/unix-plugins.c)
        endif()
    endif()
endif()

if ((LWS_ROLE_H1 OR LWS_ROLE_H2) AND NOT LWS_WITHOUT_SERVER)
    list(APPEND SOURCES
        lib/roles/http/server/server.c
        lib/roles/http/server/lws-spa.c)
endif()

if (LWS_ROLE_WS AND NOT LWS_WITHOUT_EXTENSIONS)
    list(APPEND HDR_PRIVATE
        lib/roles/ws/ext/extension-permessage-deflate.h)
    list(APPEND SOURCES
        lib/roles/ws/ext/extension.c
        lib/roles/ws/ext/extension-permessage-deflate.c)
endif()

if (LWS_WITH_HTTP_PROXY)
    list(APPEND SOURCES
        lib/roles/http/server/rewrite.c)
endif()

if (LWS_WITH_POLL AND LWS_WITH_NETWORK)
    list(APPEND SOURCES

```



```

        lib/event-libs/poll/poll.c)
endif()

if (LWS_WITH_LIBUV AND LWS_WITH_NETWORK)
    list(APPEND SOURCES
        lib/event-libs/libuv/libuv.c)
endif()

if (LWS_WITH_LIBEVENT AND LWS_WITH_NETWORK)
    list(APPEND SOURCES
        lib/event-libs/libevent/libevent.c)
endif()

if (LWS_WITH_LIBEV AND LWS_WITH_NETWORK)
    list(APPEND SOURCES
        lib/event-libs/libev/libev.c)
endif()

if (LWS_WITH_LEJP)
    list(APPEND SOURCES
        lib/misc/lejp.c)
endif()
if (LWS_WITH_LEJP_CONF AND LWS_WITH_NETWORK AND NOT LWS_PLAT_OPTEE)
    list(APPEND SOURCES
        "lib/roles/http/server/lejp-conf.c"
    )
endif()

if (LWS_WITH_ABSTRACT)
    list(APPEND SOURCES
        lib/abstract/transport/unit-test.c)
endif()

if (LWS_WITH_SMTP)
    list(APPEND SOURCES
        lib/abstract/protocols/smtp/smtp.c)
endif()

if (LWS_WITH_RANGES)
    list(APPEND SOURCES
        lib/roles/http/server/ranges.c)
endif()

if (LWS_WITH_ZIP_FOPS)
    if (LWS_WITH_ZLIB)
        list(APPEND SOURCES
            lib/roles/http/server/fops-zip.c)
    else()
        message(FATAL_ERROR "Pre-zipped file support (LWS_WITH_ZIP_FOPS)
requires ZLIB (LWS_WITH_ZLIB)")
    endif()
endif()

if (LWS_WITH_JOSE)
    list(APPEND SOURCES
        lib/jose/jwk/jwk.c
        lib/jose/jws/jose.c
        lib/jose/jws/jws.c
        lib/jose/jwe/jwe.c)
endif()

```

```

        lib/jose/jwe/enc/aescbc.c
        lib/jose/jwe/enc/aesgcm.c
        lib/jose/jwe/enc/aeskw.c
        lib/jose/jwe/jwe-rsa-aescbc.c
        lib/jose/jwe/jwe-rsa-aesgcm.c
        lib/jose/jwe/jwe-ecdh-es-aeskw.c
    )
endif()

if (LWS_WITH_JOSE OR LWS_WITH_GENCRYPTO)
    list(APPEND SOURCES
        lib/tls/lws-gencrypto-common.c)
endif()

# Add helper files for Windows.
if (WIN32)
    set(WIN32_HELPERS_PATH win32port/win32helpers)
    include_directories(${WIN32_HELPERS_PATH})

    if (WIN32)
        list(APPEND SOURCES
            ${WIN32_HELPERS_PATH}/gettimeofday.c
        )

        list(APPEND HDR_PRIVATE
            ${WIN32_HELPERS_PATH}/gettimeofday.h
        )
    endif(WIN32)
endif()

else()
    # Unix.
    if (NOT LWS_WITHOUT_DAEMONIZE)
        list(APPEND SOURCES
            lib/misc/daemonize.c)
    endif()
endif()

if (UNIX)
    if (NOT LWS_HAVE_GETIFADDRS)
        list(APPEND HDR_PRIVATE lib/misc/getifaddrs.h)
        list(APPEND SOURCES lib/misc/getifaddrs.c)
    endif()
endif()

if ((CMAKE_C_COMPILER_ID MATCHES "Clang") OR (CMAKE_CXX_COMPILER_ID MATCHES
"Clang"))
    set(COMPILER_IS_CLANG ON)
endif()

if (CMAKE_COMPILER_IS_GNUCC OR CMAKE_COMPILER_IS_GNUCXX OR COMPILER_IS_CLANG)
    include (CheckCCompilerFlag)
    CHECK_C_COMPILER_FLAG(-fvisibility=hidden LWS_HAVE_VISIBILITY)
    if (LWS_HAVE_VISIBILITY)
        set(VISIBILITY_FLAG -fvisibility=hidden)
    endif()
    if (LWS_WITH_GCOV)
        set (GCOV_FLAGS "-fprofile-arcs -ftest-coverage ")
    endif()
endif()

```

```

    if (LWS_WITH_ASAN)
        set (ASAN_FLAGS "-fsanitize=address -fsanitize=undefined -fsanitize-
address-use-after-scope -fsanitize-undefined-trap-on-error")
        if (NOT COMPILER_IS_CLANG)
            set (ASAN_FLAGS "${ASAN_FLAGS} -fsanitize=pointer-compare -
fsanitize=pointer-subtract -fsanitize=leak")
        endif()
        message("Enabling ASAN")
    endif()

    check_c_compiler_flag("-Wignored-qualifiers" LWS_GCC_HAS_IGNORED_QUALIFIERS)
    check_c_compiler_flag("-Wtype-limits" LWS_GCC_HAS_TYPE_LIMITS)

    if (LWS_GCC_HAS_IGNORED_QUALIFIERS)
        set(CMAKE_C_FLAGS "-Wignored-qualifiers ${CMAKE_C_FLAGS}" )
    endif()

    if (LWS_GCC_HAS_TYPE_LIMITS)
        set(CMAKE_C_FLAGS "-Wtype-limits ${CMAKE_C_FLAGS}" )
    endif()

    if (UNIX AND NOT LWS_WITH_ESP32)
        set(CMAKE_C_FLAGS "-Wall -Wsign-compare -Wuninitialized -Werror $
{VISIBILITY_FLAG} -Wundef ${GCOV_FLAGS} ${CMAKE_C_FLAGS} ${ASAN_FLAGS}" )
    else()
        set(CMAKE_C_FLAGS "-Wall -Wsign-compare -Wuninitialized -Werror $
{VISIBILITY_FLAG} ${GCOV_FLAGS} ${CMAKE_C_FLAGS}" )
    endif()
endif ( )

if (LWS_PLAT_OPTEE)
    set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} --sysroot
../../../../lib/libutils/isoc/include -I../../../../lib/libutils/isoc/include -
I../../../../lib/libutils/ext/include" )
endif()

if ((CMAKE_COMPILER_IS_GNUCC OR CMAKE_COMPILER_IS_GNUCXX) AND NOT
LWS_WITHOUT_TESTAPPS)
    if (UNIX AND LWS_HAVE_PTHREAD_H)
        # jeez clang understands -pthread but dies if he sees it at link time!
        # http://stackoverflow.com/questions/2391194/what-is-gs-pthread-equiv-in-
clang
        set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -pthread" )
    endif()
endif()

if (COMPILER_IS_CLANG)

    # otherwise osx blows a bunch of openssl deprecated api errors
    set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -Wno-deprecated-declarations" )
    if (UNIX AND LWS_HAVE_PTHREAD_H)
        set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -pthread" )
    endif()
endif()

source_group("Headers Private" FILES ${HDR_PRIVATE})
source_group("Headers Public" FILES ${HDR_PUBLIC})
source_group("Sources" FILES ${SOURCES})
source_group("Resources" FILES ${RESOURCES})

```

```

#
# Create the lib.
#
set(LWS_LIBRARIES)

if (LWS_WITH_STATIC)
  if (LWS_STATIC_PIC)
    set(CMAKE_POSITION_INDEPENDENT_CODE ON)
  endif()
  add_library(websockets STATIC
              ${HDR_PRIVATE}
              ${HDR_PUBLIC}
              ${SOURCES})
  list(APPEND LWS_LIBRARIES websockets)

  if (WIN32)
    # Windows uses the same .lib ending for static libraries and shared
    # library linker files, so rename the static library.
    set_target_properties(websockets
                          PROPERTIES
                          OUTPUT_NAME websockets_static)
  endif()
  add_custom_command(
    TARGET websockets
    COMMAND ${CMAKE_COMMAND} -E copy
    ${CMAKE_CURRENT_SOURCE_DIR}/include/libwebsockets.h
    ${CMAKE_CURRENT_BINARY_DIR}/include/libwebsockets.h
    )

  add_custom_command(
    TARGET websockets
    COMMAND ${CMAKE_COMMAND} -E copy_directory $
    {CMAKE_CURRENT_SOURCE_DIR}/include/libwebsockets/

    ${CMAKE_CURRENT_BINARY_DIR}/include/libwebsockets
    )

  add_custom_command(
    TARGET websockets
    COMMAND ${CMAKE_COMMAND} -E copy
    ${CMAKE_CURRENT_BINARY_DIR}/lws_config.h
    ${CMAKE_CURRENT_BINARY_DIR}/include/lws_config.h
    )
endif()

if (LWS_WITH_SHARED)
  add_library(websockets_shared SHARED
              ${HDR_PRIVATE}
              ${HDR_PUBLIC}
              ${SOURCES}
              ${RESOURCES})
  list(APPEND LWS_LIBRARIES websockets_shared)

  # We want the shared lib to be named "libwebsockets"
  # not "libwebsocket_shared".

```

```

    set_target_properties(websockets_shared
        PROPERTIES
        OUTPUT_NAME websockets)

    if (WIN32)
        # Compile as DLL (export function declarations)
        set_property(
            TARGET websockets_shared
            PROPERTY COMPILE_DEFINITIONS
            LWS_DLL
            LWS_INTERNAL)
    endif()

    if (APPLE)
        set_property(TARGET websockets_shared PROPERTY MACOSX_RPATH YES)
    endif()

    add_custom_command(
        TARGET websockets_shared
        COMMAND ${CMAKE_COMMAND} -E copy
        ${CMAKE_CURRENT_SOURCE_DIR}/include/libwebsockets.h
        ${CMAKE_CURRENT_BINARY_DIR}/include/libwebsockets.h
    )

    add_custom_command(
        TARGET websockets
        COMMAND ${CMAKE_COMMAND} -E copy_directory $
        {CMAKE_CURRENT_SOURCE_DIR}/include/libwebsockets
        ${CMAKE_CURRENT_BINARY_DIR}/include/libwebsockets
    )

    add_custom_command(
        TARGET websockets_shared
        COMMAND ${CMAKE_COMMAND} -E copy
        ${CMAKE_CURRENT_BINARY_DIR}/lws_config.h
        ${CMAKE_CURRENT_BINARY_DIR}/include/lws_config.h
    )

endif()

# Set the so version of the lib.
# Equivalent to LDFLAGS=-version-info x:x:x
if(CMAKE_COMPILER_IS_GNUCC OR CMAKE_COMPILER_IS_GNUCXX OR COMPILER_IS_CLANG)
    foreach(lib ${LWS_LIBRARIES})
        set_target_properties(${lib}
            PROPERTIES
            SOVERSION ${SOVERSION})
    endforeach()
endif()

set(LIB_LIST)

#
# Find libraries.
#

```

```

#
# ZLIB (needed for deflate extension and if LWS_WITH_HTTP_STREAM_COMPRESSION)
#
if (LWS_WITH_ZLIB)
    if (LWS_WITH_BUNDLED_ZLIB)
        if (WIN32)
            set(WIN32_ZLIB_PATH "win32port/zlib")
            set(ZLIB_SRCS
                ${WIN32_ZLIB_PATH}/adler32.c
                ${WIN32_ZLIB_PATH}/compress.c
                ${WIN32_ZLIB_PATH}/crc32.c
                ${WIN32_ZLIB_PATH}/deflate.c
                ${WIN32_ZLIB_PATH}/gzlib.c
                ${WIN32_ZLIB_PATH}/gzread.c
                ${WIN32_ZLIB_PATH}/gzwrite.c
                ${WIN32_ZLIB_PATH}/inffast.c
                ${WIN32_ZLIB_PATH}/inftrees.c
                ${WIN32_ZLIB_PATH}/trees.c
                ${WIN32_ZLIB_PATH}/uncompr.c
                ${WIN32_ZLIB_PATH}/zutil.c)
            add_library(zlib_internal STATIC ${ZLIB_SRCS})
            set(ZLIB_INCLUDE_DIRS ${WIN32_ZLIB_PATH})
            get_property(ZLIB_LIBRARIES TARGET zlib_internal PROPERTY
LOCATION)
                set(ZLIB_FOUND 1)
                # Make sure zlib_internal is compiled before the libs.
                foreach (lib ${LWS_LIBRARIES})
                    add_dependencies(${lib} zlib_internal)
                endforeach()
            else()
                message(FATAL_ERROR "Don't have bundled zlib for that platform")
            endif()
        elseif (NOT ZLIB_FOUND)
            if (LWS_WITH_MINIZ)
                find_package(Miniz REQUIRED)
                set(ZLIB_INCLUDE_DIRS ${MINIZ_INCLUDE_DIRS})
                set(ZLIB_LIBRARIES ${MINIZ_LIBRARIES})
            else()
                find_package(ZLIB REQUIRED)
            endif()
        endif()
        message("zlib/miniz include dirs: ${ZLIB_INCLUDE_DIRS}")
        message("zlib/miniz libraries: ${ZLIB_LIBRARIES}")
        include_directories(${ZLIB_INCLUDE_DIRS})
        list(APPEND LIB_LIST ${ZLIB_LIBRARIES})
    endif()

    if (LWS_WITH_HTTP_BROTLI)
        list(APPEND LIB_LIST brotlienc brotlidec brotlidec)
    endif()

#
# OpenSSL
#
if (LWS_WITH_SSL)
    message("Compiling with SSL support")

```

```

set(chose_ssl 0)
if (LWS_WITH_WOLFSSL)
    # Use wolfSSL as OpenSSL replacement.
    # TODO: Add a find_package command for this also.
    message("wolfSSL include dir: ${WOLFSSL_INCLUDE_DIRS}")
    message("wolfSSL libraries: ${WOLFSSL_LIBRARIES}")

    # Additional to the root directory we need to include
    # the wolfssl/ subdirectory which contains the OpenSSL
    # compatibility layer headers.

    if (LWS_WITH_CYASSL)
        foreach(inc ${WOLFSSL_INCLUDE_DIRS})
            include_directories("${inc}" "${inc}/cyassl")
        endforeach()
    else()
        foreach(inc ${WOLFSSL_INCLUDE_DIRS})
            include_directories("${inc}" "${inc}/wolfssl")
        endforeach()
    endif()

    list(APPEND LIB_LIST "${WOLFSSL_LIBRARIES}")
    set(chose_ssl 1)
endif()

if (LWS_WITH_MBEDTLS)
    message("MBEDTLS include dir: ${MBEDTLS_INCLUDE_DIRS}")
    message("MBEDTLS libraries: ${MBEDTLS_LIBRARIES}")

    foreach(inc ${MBEDTLS_INCLUDE_DIRS})
        include_directories("${inc}" "${inc}/mbedtls")
    endforeach()

    list(APPEND LIB_LIST "${MBEDTLS_LIBRARIES}")
    set(chose_ssl 1)
endif()

if (NOT chose_ssl)
    if (NOT OPENSSL_FOUND AND NOT LWS_WITH_BORINGSSL)
        # TODO: Add support for STATIC also.
        if (NOT LWS_WITH_ESP32)
            find_package(OpenSSL REQUIRED)
        endif()
        set(OPENSSL_INCLUDE_DIRS "${OPENSSL_INCLUDE_DIR}")
    endif()

    message("OpenSSL include dir: ${OPENSSL_INCLUDE_DIRS}")
    if (NOT LWS_WITH_ESP32)
        message("OpenSSL libraries: ${OPENSSL_LIBRARIES}")
    endif()

    include_directories("${OPENSSL_INCLUDE_DIRS}")
    if (NOT LWS_WITH_ESP32)
        list(APPEND LIB_LIST ${OPENSSL_LIBRARIES})
    endif()
endif()

if (NOT LWS_WITH_MBEDTLS)
    # older (0.98) openssl lacks this
    set(CMAKE_REQUIRED_INCLUDES ${OPENSSL_INCLUDE_DIRS})

```

```

        check_include_file(openssl/ecdh.h LWS_HAVE_OPENSSL_ECDH_H)

        if (LWS_SSL_SERVER_WITH_ECDH_CERT AND NOT LWS_HAVE_OPENSSL_ECDH_H)
            message(FATAL_ERROR "Missing openssl/ecdh.h, so cannot use
LWS_SSL_SERVER_WITH_ECDH_CERT")
        endif()
    else()
        unset(LWS_HAVE_OPENSSL_ECDH_H)
    endif(NOT LWS_WITH_MBEDTLS)
endif()

endif(LWS_WITH_SSL)

if (LWS_WITH_LIBEV)
    if (NOT LIBEV_FOUND)
        find_path(LIBEV_INCLUDE_DIRS NAMES ev.h)
        find_library(LIBEV_LIBRARIES NAMES ev)
        if(LIBEV_INCLUDE_DIRS AND LIBEV_LIBRARIES)
            set(LIBEV_FOUND 1)
        endif()
    endif()
    message("libev include dir: ${LIBEV_INCLUDE_DIRS}")
    message("libev libraries: ${LIBEV_LIBRARIES}")
    include_directories("${LIBEV_INCLUDE_DIRS}")
    list(APPEND LIB_LIST ${LIBEV_LIBRARIES})
endif(LWS_WITH_LIBEV)

if (LWS_WITH_LIBUV)
    if (NOT LIBUV_FOUND)
        find_path(LIBUV_INCLUDE_DIRS NAMES uv.h)
        find_library(LIBUV_LIBRARIES NAMES uv)
        if(LIBUV_INCLUDE_DIRS AND LIBUV_LIBRARIES)
            set(LIBUV_FOUND 1)
        endif()
    endif()
    message("libuv include dir: ${LIBUV_INCLUDE_DIRS}")
    message("libuv libraries: ${LIBUV_LIBRARIES}")
    include_directories("${LIBUV_INCLUDE_DIRS}")
    list(APPEND LIB_LIST ${LIBUV_LIBRARIES})
endif()

if (LWS_WITH_LIBEVENT)
    if (NOT LIBEVENT_FOUND)
        find_path(LIBEVENT_INCLUDE_DIRS NAMES event2/event.h)
        find_library(LIBEVENT_LIBRARIES NAMES event)
        if(LIBEVENT_INCLUDE_DIRS AND LIBEVENT_LIBRARIES)
            set(LIBEVENT_FOUND 1)
        endif()
    endif()
    message("libevent include dir: ${LIBEVENT_INCLUDE_DIRS}")
    message("libevent libraries: ${LIBEVENT_LIBRARIES}")
    include_directories("${LIBEVENT_INCLUDE_DIRS}")
    list(APPEND LIB_LIST ${LIBEVENT_LIBRARIES})
endif(LWS_WITH_LIBEVENT)

if (LWS_WITH_SQLITE3)
    if (NOT SQLITE3_FOUND)
        find_path(SQLITE3_INCLUDE_DIRS NAMES sqlite3.h)
        find_library(SQLITE3_LIBRARIES NAMES sqlite3)
    endif()
endif()

```



```

        if(SQLITE3_INCLUDE_DIRS AND SQLITE3_LIBRARIES)
            set(SQLITE3_FOUND 1)
        endif()
    endif()
    message("sqlite3 include dir: ${SQLITE3_INCLUDE_DIRS}")
    message("sqlite3 libraries: ${SQLITE3_LIBRARIES}")
    include_directories("${SQLITE3_INCLUDE_DIRS}")
    list(APPEND LIB_LIST ${SQLITE3_LIBRARIES})
endif()

if (LWS_WITH_HUBBUB)
    find_library(LIBHUBBUB_LIBRARIES NAMES hubbub)
    list(APPEND LIB_LIST ${LIBHUBBUB_LIBRARIES} )
endif()

if (LWS_ROLE_DBUS)
    message("dbus include dir 1: ${LWS_DBUS_INCLUDE1}")
    message("dbus include dir 2: ${LWS_DBUS_INCLUDE2}")
    include_directories("${LWS_DBUS_INCLUDE1}")
    include_directories("${LWS_DBUS_INCLUDE2}")
    list(APPEND LIB_LIST ${LWS_DBUS_LIB})
endif()

#
# Platform specific libs.
#
if (WINCE)
    list(APPEND LIB_LIST ws2.lib)
elseif (WIN32)
    list(APPEND LIB_LIST ws2_32.lib userenv.lib psapi.lib iphlpapi.lib)
endif()

if (${CMAKE_SYSTEM_NAME} MATCHES "QNX")
    list(APPEND LIB_LIST socket)
endif()

if (UNIX)
    list(APPEND LIB_LIST m)
endif()

if(SMARTOS)
    list(APPEND LIB_LIST socket)
endif()

if (HAIKU)
    list(APPEND LIB_LIST network)
endif()

if (LWS_HAVE_LIBCAP)
    list(APPEND LIB_LIST cap )
endif()

# Setup the linking for all libs.
foreach (lib ${LWS_LIBRARIES})
    target_link_libraries(${lib} ${LIB_LIST})
endforeach()

set (temp ${CMAKE_REQUIRED_LIBRARIES})

```

```

set(CMAKE_REQUIRED_LIBRARIES ${LIB_LIST})

if (LWS_WITH_ZLIB)
    if (LWS_WITH_BUNDLED_ZLIB)
        if (WIN32)
            # it's trying to delete internal zlib entry
            LIST(REMOVE_AT CMAKE_REQUIRED_LIBRARIES 0 )
        endif()
    endif()
endif()

CHECK_FUNCTION_EXISTS(SSL_CTX_set1_param LWS_HAVE_SSL_CTX_set1_param)
CHECK_FUNCTION_EXISTS(SSL_set_info_callback LWS_HAVE_SSL_SET_INFO_CALLBACK)
CHECK_FUNCTION_EXISTS(X509_VERIFY_PARAM_set1_host
LWS_HAVE_X509_VERIFY_PARAM_set1_host)
CHECK_FUNCTION_EXISTS(RSA_set0_key LWS_HAVE_RSA_SET0_KEY)
CHECK_FUNCTION_EXISTS(X509_get_key_usage LWS_HAVE_X509_get_key_usage)
CHECK_FUNCTION_EXISTS(EVP_PKEY_new_raw_private_key
LWS_HAVE_SSL_CTX_EVP_PKEY_new_raw_private_key)
CHECK_FUNCTION_EXISTS(SSL_CTX_get0_certificate LWS_HAVE_SSL_CTX_get0_certificate)
CHECK_FUNCTION_EXISTS(SSL_get0_alpn_selected LWS_HAVE_SSL_get0_alpn_selected)
CHECK_FUNCTION_EXISTS(SSL_set_alpn_protos LWS_HAVE_SSL_set_alpn_protos)
CHECK_FUNCTION_EXISTS(EVP_aes_128_cfb8 LWS_HAVE_EVP_aes_128_cfb8)
CHECK_FUNCTION_EXISTS(EVP_aes_128_cfb128 LWS_HAVE_EVP_aes_128_cfb128)
CHECK_FUNCTION_EXISTS(EVP_aes_192_cfb8 LWS_HAVE_EVP_aes_192_cfb8)
CHECK_FUNCTION_EXISTS(EVP_aes_192_cfb128 LWS_HAVE_EVP_aes_192_cfb128)
CHECK_FUNCTION_EXISTS(EVP_aes_256_cfb8 LWS_HAVE_EVP_aes_256_cfb8)
CHECK_FUNCTION_EXISTS(EVP_aes_256_cfb128 LWS_HAVE_EVP_aes_256_cfb128)
CHECK_FUNCTION_EXISTS(EVP_aes_128_xts LWS_HAVE_EVP_aes_128_xts)
CHECK_FUNCTION_EXISTS(RSA_verify_pss_mgf1 LWS_HAVE_RSA_verify_pss_mgf1)
CHECK_FUNCTION_EXISTS(HMAC_CTX_new LWS_HAVE_HMAC_CTX_new)
CHECK_FUNCTION_EXISTS(SSL_CTX_set_ciphersuites LWS_HAVE_SSL_CTX_set_ciphersuites)
if (LWS_WITH_SSL AND NOT LWS_WITH_MBEDTLS)
    if (UNIX)
        set(CMAKE_REQUIRED_LIBRARIES ${CMAKE_REQUIRED_LIBRARIES} dl)
    endif()
CHECK_C_SOURCE_COMPILES("#include <openssl/ssl.h>\nint main(void) { STACK_OF(X509)
*c = NULL; SSL_CTX *ctx = NULL; return (int)SSL_CTX_get_extra_chain_certs_only(ctx,
&c); }\n" LWS_HAVE_SSL_EXTRA_CHAIN_CERTS)
CHECK_C_SOURCE_COMPILES("#include <openssl/ssl.h>\nint main(void) { EVP_MD_CTX
*md_ctx = NULL; EVP_MD_CTX_free(md_ctx); return 0; }\n" LWS_HAVE_EVP_MD_CTX_free)
CHECK_FUNCTION_EXISTS(ECDSA_SIG_set0 LWS_HAVE_ECDSA_SIG_set0)
CHECK_FUNCTION_EXISTS(BN_bn2binpad LWS_HAVE_BN_bn2binpad)
CHECK_FUNCTION_EXISTS(EVP_aes_128_wrap LWS_HAVE_EVP_aes_128_wrap)
CHECK_FUNCTION_EXISTS(EC_POINT_get_affine_coordinates
LWS_HAVE_EC_POINT_get_affine_coordinates)
endif()
if (LWS_WITH_MBEDTLS)
    set(LWS_HAVE_TLS_CLIENT_METHOD 1)
    if (NOT LWS_WITH_ESP32)
        # not supported in esp-idf openssl wrapper yet, but is in our version
        set(LWS_HAVE_X509_VERIFY_PARAM_set1_host 1)
    endif()

    CHECK_FUNCTION_EXISTS(mbedtls_ssl_conf_alpn_protocols
LWS_HAVE_mbedtls_ssl_conf_alpn_protocols)
    CHECK_FUNCTION_EXISTS(mbedtls_ssl_get_alpn_protocol
LWS_HAVE_mbedtls_ssl_get_alpn_protocol)
    CHECK_FUNCTION_EXISTS(mbedtls_ssl_conf_sni LWS_HAVE_mbedtls_ssl_conf_sni)

```

```

        CHECK_FUNCTION_EXISTS(mbedtls_ssl_set_hs_ca_chain
LWS_HAVE_mbedtls_ssl_set_hs_ca_chain)
        CHECK_FUNCTION_EXISTS(mbedtls_ssl_set_hs_own_cert
LWS_HAVE_mbedtls_ssl_set_hs_own_cert)
        CHECK_FUNCTION_EXISTS(mbedtls_ssl_set_hs_authmode
LWS_HAVE_mbedtls_ssl_set_hs_authmode)
        CHECK_FUNCTION_EXISTS(mbedtls_net_init LWS_HAVE_mbedtls_net_init)

else()
CHECK_FUNCTION_EXISTS(TLS_client_method LWS_HAVE_TLS_CLIENT_METHOD)
CHECK_FUNCTION_EXISTS(TLSv1_2_client_method LWS_HAVE_TLSV1_2_CLIENT_METHOD)
endif()

# ideally we want to use pipe2()

CHECK_C_SOURCE_COMPILES("#define _GNU_SOURCE\n#include <unistd.h>\nint main(void)
{int fd[2];\n return pipe2(fd, 0);\n}\n" LWS_HAVE_PIPE2)

# tcp keepalive needs this on linux to work practically... but it only exists
# after kernel 2.6.37

CHECK_C_SOURCE_COMPILES("#include <netinet/tcp.h>\nint main(void) { return
TCP_USER_TIMEOUT; }\n" LWS_HAVE_TCP_USER_TIMEOUT)

set(CMAKE_REQUIRED_LIBRARIES ${temp})
# Generate the lws_config.h that includes all the public compilation settings.
configure_file(
    "${PROJECT_SOURCE_DIR}/cmake/lws_config.h.in"
    "${PROJECT_BINARY_DIR}/lws_config.h")

# Generate the lws_config.h that includes all the private compilation settings.
configure_file(
    "${PROJECT_SOURCE_DIR}/cmake/lws_config_private.h.in"
    "${PROJECT_BINARY_DIR}/lws_config_private.h")

# Generate self-signed SSL certs for the test-server.

if (LWS_WITH_SSL AND NOT LWS_WITH_WOLFSSL)
    message("Searching for OpenSSL executable and dlls")
    find_package(OpenSSLbins)
    message("OpenSSL executable: ${OPENSSL_EXECUTABLE}")
    if (OPENSSL_EXECUTABLE MATCHES "^$")
        set(OPENSSL_EXECUTABLE openssl)
    endif()
    if (NOT OPENSSL_EXECUTABLE)
        set(OPENSSL_EXECUTABLE openssl)
    endif()
endif()

set(GENCERTS 0)

if (LWS_WITH_SSL AND OPENSSL_EXECUTABLE AND NOT LWS_WITHOUT_TEST_SERVER AND NOT
LWS_WITHOUT_SERVER AND NOT LWS_WITHOUT_TESTAPPS)
    set(GENCERTS 1)
endif()
if (LWS_WITH_ESP32)
    set(GENCERTS 1)
endif()

```

```

message(" GENCERTS = ${GENCERTS}")
if (GENCERTS)
    message("Generating SSL Certificates for the test-server...")

    set(TEST_SERVER_SSL_KEY "${PROJECT_BINARY_DIR}/libwebsockets-test-
server.key.pem")
    set(TEST_SERVER_SSL_CERT "${PROJECT_BINARY_DIR}/libwebsockets-test-
server.pem")

    if (WIN32)
        if (MINGW)
            message("cmd = \"${OPENSSL_EXECUTABLE}\" req -new -newkey
rsa:1024 -days 10000 -nodes -x509 -subj \"/C=GB/ST=Erewhon/L=All
around/O=libwebsockets-test/CN=localhost\" -keyout \"${TEST_SERVER_SSL_KEY}\" -
out \"${TEST_SERVER_SSL_CERT}\"")
            execute_process(
                COMMAND "${OPENSSL_EXECUTABLE}" req -new -newkey rsa:1024 -
days 10000 -nodes -x509 -subj "/C=GB/ST=Erewhon/L=All
around/O=libwebsockets-test/CN=localhost" -keyout "${TEST_SERVER_SSL_KEY}" -out "${
TEST_SERVER_SSL_CERT}"
                RESULT_VARIABLE OPENSSL_RETURN_CODE)
        else()
            file(WRITE "${PROJECT_BINARY_DIR}/openssl_input.txt"
"GB\n"
"Erewhon\n"
"All around\n"
"libwebsockets-test\n"
"localhost\n"
"none@invalid.org\n\n"
)

            # The "type" command is a bit picky with paths.
            file(TO_NATIVE_PATH "${PROJECT_BINARY_DIR}/openssl_input.txt"
OPENSSL_INPUT_WIN_PATH)
            message("OPENSSL_INPUT_WIN_PATH = ${OPENSSL_INPUT_WIN_PATH}")
            message("cmd = \"${OPENSSL_EXECUTABLE}\" req -new -newkey
rsa:1024 -days 10000 -nodes -x509 -keyout \"${TEST_SERVER_SSL_KEY}\" -out \"${
TEST_SERVER_SSL_CERT}\"")

            execute_process(
                COMMAND cmd /c type "${OPENSSL_INPUT_WIN_PATH}"
                COMMAND "${OPENSSL_EXECUTABLE}" req -new -newkey rsa:1024 -
days 10000 -nodes -x509 -keyout "${TEST_SERVER_SSL_KEY}" -out "${
TEST_SERVER_SSL_CERT}"
                RESULT_VARIABLE OPENSSL_RETURN_CODE
                OUTPUT_QUIET ERROR_QUIET)

            message("\n")
        endif()

        if (OPENSSL_RETURN_CODE)
            message(WARNING "!!! Failed to generate SSL certificate for Test
Server using cmd.exe !!!:\nOpenSSL return code = ${OPENSSL_RETURN_CODE}")
        else()
            message("SUCCESFULLY generated SSL certificate")
        endif()
    else()
        # Unix.
        execute_process(

```

```

        COMMAND printf "GB\\nErewhon\\nAll around\\nlibwebsockets-test\\n\\nlocalhost\\nnone@invalid.org\\n"
        COMMAND "${OPENSSL_EXECUTABLE}"
            req -new -newkey rsa:1024 -days 10000 -nodes -x509 -keyout
"${TEST_SERVER_SSL_KEY}" -out "${TEST_SERVER_SSL_CERT}"
        RESULT_VARIABLE OPENSSL_RETURN_CODE
        # OUTPUT_QUIET ERROR_QUIET
    )

    if (OPENSSL_RETURN_CODE)
        message(WARNING "!!! Failed to generate SSL certificate for Test
Server!!!:\\nOpenSSL return code = ${OPENSSL_RETURN_CODE}")
    else()
        message("SUCCESSFULLY generated SSL certificate")
    endif()
endif()

list(APPEND TEST_SERVER_DATA
    "${TEST_SERVER_SSL_KEY}"
    "${TEST_SERVER_SSL_CERT}")
endif()

```

```

#
# Test applications
#
set(TEST_APP_LIST)
if ((LWS_ROLE_H1 OR LWS_ROLE_H2) AND NOT LWS_WITHOUT_TESTAPPS)
    #
    # Helper function for adding a test app.
    #
    macro(create_test_app TEST_NAME MAIN_SRC S2 S3 S4 S5 S6)

        set(TEST_SRCS ${MAIN_SRC})
        set(TEST_HDR)
        if ("${S2}" STREQUAL "")
        else()
            list(APPEND TEST_SRCS ${S2})
        endif()
        if ("${S3}" STREQUAL "")
        else()
            list(APPEND TEST_SRCS ${S3})
        endif()
        if ("${S4}" STREQUAL "")
        else()
            list(APPEND TEST_SRCS ${S4})
        endif()
        if ("${S5}" STREQUAL "")
        else()
            list(APPEND TEST_SRCS ${S5})
        endif()
        if ("${S6}" STREQUAL "")
        else()
            list(APPEND TEST_SRCS ${S6})
        endif()
        if (WIN32)
            list(APPEND TEST_SRCS

```

```

        ${WIN32_HELPERS_PATH}/getopt.c
        ${WIN32_HELPERS_PATH}/getopt_long.c
        ${WIN32_HELPERS_PATH}/gettimeofday.c
    )

    list(APPEND TEST_HDR
        ${WIN32_HELPERS_PATH}/getopt.h
        ${WIN32_HELPERS_PATH}/gettimeofday.h
    )
endif(WIN32)

source_group("Headers Private"   FILES ${TEST_HDR})
source_group("Sources"          FILES ${TEST_SRCS})
add_executable(${TEST_NAME} ${TEST_SRCS} ${TEST_HDR})

if (LWS_LINK_TESTAPPS_DYNAMIC)
    if (NOT LWS_WITH_SHARED)
        message(FATAL_ERROR "Build of the shared library is
disabled. LWS_LINK_TESTAPPS_DYNAMIC must be combined with LWS_WITH_SHARED.")
    endif()
    target_link_libraries(${TEST_NAME} websockets_shared)
    add_dependencies(${TEST_NAME} websockets_shared)
else()
    if (NOT LWS_WITH_STATIC)
        message(FATAL_ERROR "Build of the static library is
disabled. Disabled LWS_LINK_TESTAPPS_DYNAMIC must be combined with
LWS_WITH_STATIC.")
    endif()
    target_link_libraries(${TEST_NAME} websockets)
    add_dependencies(${TEST_NAME} websockets)
    if (UNIX AND LWS_WITH_SSL AND NOT LWS_WITH_MBEDTLS)
        target_link_libraries(${TEST_NAME} dl)
    endif()
endif()

if (LWS_WITH_HTTP_STREAM_COMPRESSION)
    target_link_libraries(${TEST_NAME} z)
endif()

# Set test app specific defines.
set_property(TARGET ${TEST_NAME}
    PROPERTY COMPILE_DEFINITIONS
        INSTALL_DATADIR="${CMAKE_INSTALL_PREFIX}/share"
)

# Prefix the binary names with libwebsockets.
set_target_properties(${TEST_NAME}
    PROPERTIES
        OUTPUT_NAME libwebsockets-${TEST_NAME})

# Add to the list of tests.
list(APPEND TEST_APP_LIST ${TEST_NAME})
endmacro()

if (UNIX AND LWS_WITH_PLUGINS)
    set(CMAKE_C_FLAGS "-fPIC ${CMAKE_C_FLAGS}")
    if(NOT((${CMAKE_SYSTEM_NAME} MATCHES "FreeBSD") OR ($
{CMAKE_SYSTEM_NAME} MATCHES "QNX")))
        target_link_libraries(websockets dl)
    endif()
endif()

```

```

        endif()
    endif()

    if (LWS_WITH_LIBEV)
        # libev generates a big mess of warnings with gcc, maintainer claims
gcc to blame
        set_source_files_properties( lib/event-libs/libev/libev.c PROPERTIES
COMPILE_FLAGS "-Wno-error" )
    endif()

    if (NOT LWS_WITHOUT_SERVER)
        #
        # test-server
        #
        if (NOT LWS_WITHOUT_TEST_SERVER)
            create_test_app(test-server "test-apps/test-server.c"
                ""
                ""
                ""
                ""
                "")

            if (LWS_WITH_CGI)
                create_test_app(test-sshd "test-apps/test-sshd.c"
                    ""
                    ""
                    ""
                    ""
                    "")
                target_include_directories(test-sshd PRIVATE "$
{PROJECT_SOURCE_DIR}/plugins/ssh-base/include")
            endif()
        endif()

    endif()

    #
    # test-server-extpoll
    #
    if (NOT LWS_WITHOUT_TEST_SERVER_EXTPOLL AND NOT WIN32)
        create_test_app(test-server-extpoll
            "test-apps/test-server.c"
            ""
            ""
            ""
            ""
            "")
        # Set defines for this executable only.
        set_property(
            TARGET test-server-extpoll
            PROPERTY COMPILE_DEFINITIONS
                EXTERNAL_POLL
                INSTALL_DATADIR="${CMAKE_INSTALL_PREFIX}/share"
        )

        # We need to link against winsock code.
        if (WIN32)
            target_link_libraries(test-server-extpoll ws2_32.lib)
        endif()
    endif()

```

```

        endif(WIN32)
    endif()

    if (LWS_WITH_LEJP)
        create_test_app(
            test-lejp
            "test-apps/test-lejp.c"
            ""
            ""
            ""
            ""
            "")
    endif()

    # Data files for running the test server.
    list(APPEND TEST_SERVER_DATA
        "${PROJECT_SOURCE_DIR}/test-apps/favicon.ico"
        "${PROJECT_SOURCE_DIR}/test-apps/leaf.jpg"
        "${PROJECT_SOURCE_DIR}/test-apps/candide.zip"
        "${PROJECT_SOURCE_DIR}/test-apps/libwebsockets.org-logo.svg"
        "${PROJECT_SOURCE_DIR}/test-apps/http2.png"
        "${PROJECT_SOURCE_DIR}/test-apps/wss-over-h2.png"
        "${PROJECT_SOURCE_DIR}/test-apps/lws-common.js"
        "${PROJECT_SOURCE_DIR}/test-apps/test.html"
        "${PROJECT_SOURCE_DIR}/test-apps/test.css"
        "${PROJECT_SOURCE_DIR}/test-apps/test.js")

    add_custom_command(TARGET test-server
                        POST_BUILD
                        COMMAND "${CMAKE_COMMAND}" -E make_directory
"$<TARGET_FILE_DIR:test-server>/../share/libwebsockets-test-server")

    # Copy the file needed to run the server so that the test apps can
    # reach them from their default output location
    foreach (TEST_FILE ${TEST_SERVER_DATA})
        if (EXISTS ${TEST_FILE})
            add_custom_command(TARGET test-server
                                POST_BUILD
                                COMMAND "${CMAKE_COMMAND}" -E copy "$
{TEST_FILE}" "$<TARGET_FILE_DIR:test-server>/../share/libwebsockets-test-server"
VERBATIM)
        endif()
    endforeach()
endif(NOT LWS_WITHOUT_SERVER)

if (NOT LWS_WITHOUT_CLIENT)
    #
    # test-client
    #
    if (NOT LWS_WITHOUT_TEST_CLIENT)
        create_test_app(test-client "test-apps/test-client.c" "" "" "" "")
    endif()
endif(NOT LWS_WITHOUT_CLIENT)

if (LWS_WITH_PLUGINS AND LWS_WITH_SHARED)
    macro(create_plugin PLUGIN_NAME PLUGIN_INCLUDE MAIN_SRC S2 S3)

```



```

set(PLUGIN_SRCS ${MAIN_SRC})

if ("${S2}" STREQUAL "")
else()
    list(APPEND PLUGIN_SRCS ${S2})
endif()
if ("${S3}" STREQUAL "")
else()
    list(APPEND PLUGIN_SRCS ${S3})
endif()

if (WIN32)
    list(APPEND PLUGIN_SRCS
        ${WIN32_HELPERS_PATH}/getopt.c
        ${WIN32_HELPERS_PATH}/getopt_long.c
        ${WIN32_HELPERS_PATH}/gettimeofday.c
    )

    list(APPEND PLUGIN_HDR
        ${WIN32_HELPERS_PATH}/getopt.h
        ${WIN32_HELPERS_PATH}/gettimeofday.h
    )
endif(WIN32)

source_group("Headers Private"    FILES ${PLUGIN_HDR})
source_group("Sources"           FILES ${PLUGIN_SRCS})
add_library(${PLUGIN_NAME} SHARED ${PLUGIN_SRCS} ${PLUGIN_HDR})

target_link_libraries(${PLUGIN_NAME} websockets_shared)
add_dependencies(${PLUGIN_NAME} websockets_shared)
include_directories(${PLUGIN_INCLUDE})

# Set test app specific defines.
set_property(TARGET ${PLUGIN_NAME}
    PROPERTY COMPILE_DEFINITIONS
    INSTALL_DATADIR="${CMAKE_INSTALL_PREFIX}/plugins"
)

SET_TARGET_PROPERTIES(${PLUGIN_NAME}
    PROPERTIES COMPILE_FLAGS ${CMAKE_C_FLAGS})

#
# set_target_properties(${PLUGIN_NAME}
#     PROPERTIES
#     OUTPUT_NAME ${PLUGIN_NAME})

list(APPEND PLUGINS_LIST ${PLUGIN_NAME})

endmacro()

if (LWS_ROLE_WS)
    create_plugin(protocol_dumb_increment ""
        "plugins/protocol_dumb_increment.c" "" "")
    create_plugin(protocol_lws_mirror ""
        "plugins/protocol_lws_mirror.c" "" "")
    create_plugin(protocol_lws_status ""
        "plugins/protocol_lws_status.c" "" "")
    create_plugin(protocol_lws_table_dirlisting ""
        "plugins/generic-table/protocol_table_dirlisting.c" "" "")

```

```

        if (NOT WIN32)
            create_plugin(protocol_lws_raw_test ""
                "plugins/protocol_lws_raw_test.c" "" "")

            create_plugin(protocol_deaddrop ""
                "plugins/deaddrop/protocol_lws_deaddrop.c" "" "")

        endif()

    if (LWS_WITH_SERVER_STATUS)
        create_plugin(protocol_lws_server_status ""
            "plugins/protocol_lws_server_status.c" "" "")
    endif()

    if (NOT LWS_WITHOUT_CLIENT)
        create_plugin(protocol_client_loopback_test ""
            "plugins/protocol_client_loopback_test.c" "" "")
    endif()

    endif()

        create_plugin(protocol_post_demo ""
            "plugins/protocol_post_demo.c" "" "")

    if (LWS_ROLE_RAW_PROXY)
        create_plugin(protocol_lws_raw_proxy ""
            "plugins/raw-proxy/protocol_lws_raw_proxy.c" "" "")
    endif()

    if (LWS_WITH_FTS)
        create_plugin(protocol_fulltext_demo ""
            "plugins/protocol_fulltext_demo.c" "" "")
    endif()

    if (LWS_WITH_SSL)
        create_plugin(protocol_lws_ssh_base "plugins/ssh-base/include"

"plugins/ssh-base/sshd.c;plugins/ssh-base/telnet.c;plugins/ssh-base/kex-25519.c"
"plugins/ssh-base/crypto/chacha.c;plugins/ssh-base/crypto/ed25519.c;plugins/ssh-
base/crypto/fe25519.c;plugins/ssh-base/crypto/ge25519.c;plugins/ssh-base/crypto/
poly1305.c;plugins/ssh-base/crypto/sc25519.c;plugins/ssh-base/crypto/
smult_curve25519_ref.c" "")
        create_plugin(protocol_lws_sshd_demo "plugins/ssh-base/include"
"plugins/protocol_lws_sshd_demo.c" "" "")

        include_directories("${PROJECT_SOURCE_DIR}/plugins/ssh-base/include")
    endif()

    if (LWS_WITH_ACME)
        create_plugin(protocol_lws_acme_client ""
            "plugins/acme-client/protocol_lws_acme_client.c" "" "")
    endif()

    if (LWS_WITH_GENERIC_SESSIONS AND LWS_ROLE_WS)
        create_plugin(protocol_generic_sessions ""
            "plugins/generic-sessions/protocol_generic_sessions.c"

```

```

        "plugins/generic-sessions/utils.c"
        "plugins/generic-sessions/handlers.c")

    if (WIN32)
        target_link_libraries(protocol_generic_sessions $
{LWS_SQLITE3_LIBRARIES})
    else()
        target_link_libraries(protocol_generic_sessions sqlite3 )
    endif(WIN32)

    create_plugin(protocol_lws_messageboard ""
        "plugins/generic-sessions/protocol_lws_messageboard.c" ""
    "")
    if (WIN32)
        target_link_libraries(protocol_lws_messageboard $
{LWS_SQLITE3_LIBRARIES})
    else()
        target_link_libraries(protocol_lws_messageboard sqlite3 )
    endif(WIN32)

endif(LWS_WITH_GENERIC_SESSIONS AND LWS_ROLE_WS)

endif(LWS_WITH_PLUGINS AND LWS_WITH_SHARED)

#
# Copy OpenSSL dlls to the output directory on Windows.
# (Otherwise we'll get an error when trying to run)
#
if (WIN32 AND LWS_WITH_SSL AND NOT LWS_WITH_WOLFSSL)
    if(OPENSSSL_BIN_FOUND)
        message("OpenSSL dlls found:")
        message("  Libeay: ${LIBEAY_BIN}")
        message("  SSLeay: ${SSLEAY_BIN}")

        foreach(TARGET_BIN ${TEST_APP_LIST})
            add_custom_command(TARGET ${TARGET_BIN}
                POST_BUILD
                COMMAND "${CMAKE_COMMAND}" -E copy "${LIBEAY_BIN}"
"$<TARGET_FILE_DIR:${TARGET_BIN}>" VERBATIM)
            add_custom_command(TARGET ${TARGET_BIN}
                POST_BUILD
                COMMAND "${CMAKE_COMMAND}" -E copy "${SSLEAY_BIN}"
"$<TARGET_FILE_DIR:${TARGET_BIN}>" VERBATIM)

            #
            # Win32: if we are using libuv, also need to copy it in the
output dir
            #
            if (WIN32 AND LWS_WITH_LIBUV)
                STRING(REPLACE ".lib" ".dll" LIBUV_BIN $
{LIBUV_LIBRARIES})

                add_custom_command(TARGET ${TARGET_BIN}
                    POST_BUILD
                    COMMAND "${CMAKE_COMMAND}" -E copy "$
{LIBUV_BIN}" "$<TARGET_FILE_DIR:${TARGET_BIN}>" VERBATIM)
            endif()
        endforeach()
    endif()
endif()

```

```

        endif()
    endif((LWS_ROLE_H1 OR LWS_ROLE_H2) AND NOT LWS_WITHOUT_TESTAPPS)

    if (LWS_WITH_LWSWS)
        list(APPEND LWSWS_SRCS
            "lwsws/main.c"
        )

        if (WIN32)
            list(APPEND LWSWS_SRCS
                ${WIN32_HELPERS_PATH}/getopt.c
                ${WIN32_HELPERS_PATH}/getopt_long.c
                ${WIN32_HELPERS_PATH}/gettimeofday.c
            )

            list(APPEND LWSWS_HDR
                ${WIN32_HELPERS_PATH}/getopt.h
                ${WIN32_HELPERS_PATH}/gettimeofday.h
            )
        endif(WIN32)

        source_group("Headers Private"    FILES ${LWSWS_HDR})
        source_group("Sources"            FILES ${LWSWS_SRCS})
        add_executable("lwsws" ${LWSWS_SRCS} ${LWSWS_HDR})

        target_link_libraries("lwsws" websockets_shared)
        add_dependencies("lwsws" websockets_shared)

        # Set test app specific defines.
        set_property(TARGET "lwsws"
            PROPERTY COMPILE_DEFINITIONS
            INSTALL_DATADIR="${CMAKE_INSTALL_PREFIX}/share"
        )
    endif (LWS_WITH_LWSWS)

    if (UNIX)

        # Generate and install pkgconfig.
        # (This is not indented, because the tabs will be part of the output)
        file(WRITE "${PROJECT_BINARY_DIR}/libwebsockets.pc"
            "prefix=\\\"${CMAKE_INSTALL_PREFIX}\\\"
            exec_prefix=\\${prefix}
            libdir=\\${exec_prefix}/lib${LIB_SUFFIX}
            includedir=\\${prefix}/include

            Name: libwebsockets
            Description: Websockets server and client library
            Version: ${CPACK_PACKAGE_VERSION_MAJOR}.${CPACK_PACKAGE_VERSION_MINOR}.${
            {CPACK_PACKAGE_VERSION_PATCH}

            Libs: -L\\${libdir} -lwebsockets
            Cflags: -I\\${includedir}"
        )

        install(FILES "${PROJECT_BINARY_DIR}/libwebsockets.pc"
            DESTINATION lib${LIB_SUFFIX}/pkgconfig)

        file(WRITE "${PROJECT_BINARY_DIR}/libwebsockets_static.pc"
            "prefix=\\\"${CMAKE_INSTALL_PREFIX}\\\"

```

```

exec_prefix=\${prefix}
libdir=\${exec_prefix}/lib${LIB_SUFFIX}
includedir=\${prefix}/include

Name: libwebsockets_static
Description: Websockets server and client static library
Version: ${CPACK_PACKAGE_VERSION_MAJOR}.${CPACK_PACKAGE_VERSION_MINOR}.${CPACK_PACKAGE_VERSION_PATCH}

Libs: -L\${libdir} -lwebsockets_static
Libs.private:
Cflags: -I\${includedir}"
)

    install(FILES "${PROJECT_BINARY_DIR}/libwebsockets_static.pc"
            DESTINATION lib${LIB_SUFFIX}/pkgconfig)

endif(UNIX)

#
# Installation preparations.
#

if(WIN32 AND NOT CYGWIN)
    set(DEF_INSTALL_CMAKE_DIR cmake)
else()
    set(DEF_INSTALL_CMAKE_DIR lib${LIB_SUFFIX}/cmake/libwebsockets)
endif()

set(LWS_INSTALL_CMAKE_DIR ${DEF_INSTALL_CMAKE_DIR} CACHE PATH "Installation
directory for CMake files")

# Export targets (This is used for other CMake projects to easily find the
libraries and include files).
if (LWS_WITH_EXPORT_LWSTARGETS)
    export(TARGETS ${LWS_LIBRARIES}
           FILE "${PROJECT_BINARY_DIR}/LibwebsocketsTargets.cmake")
endif()

export(PACKAGE libwebsockets)

# Generate the config file for the build-tree.
set(LWS__INCLUDE_DIRS
    "${PROJECT_SOURCE_DIR}/lib"
    "${PROJECT_BINARY_DIR}")
set(LIBWEBSOCKETS_INCLUDE_DIRS ${LWS__INCLUDE_DIRS} CACHE PATH "Libwebsockets
include directories")
configure_file(${PROJECT_SOURCE_DIR}/cmake/LibwebsocketsConfig.cmake.in
    ${PROJECT_BINARY_DIR}/LibwebsocketsConfig.cmake
    @ONLY)

# Generate the config file for the installation tree.
get_filename_component(LWS_ABSOLUTE_INSTALL_CMAKE_DIR ${LWS_INSTALL_CMAKE_DIR}
ABSOLUTE)
get_filename_component(LWS_ABSOLUTE_INSTALL_INCLUDE_DIR ${LWS_INSTALL_INCLUDE_DIR}
ABSOLUTE)
file(RELATIVE_PATH
    REL_INCLUDE_DIR

```

```

    "${LWS_ABSOLUTE_INSTALL_CMAKE_DIR}"
    "${LWS_ABSOLUTE_INSTALL_INCLUDE_DIR}") # Calculate the relative directory from
the cmake dir.

# Note the EVENT_CMAKE_DIR is defined in JanssonConfig.cmake.in,
# we escape it here so it's evaluated when it is included instead
# so that the include dirs are given relative to where the
# config file is located.
set(LWS__INCLUDE_DIRS
    "\${LWS_CMAKE_DIR}/${REL_INCLUDE_DIR}")
configure_file(${PROJECT_SOURCE_DIR}/cmake/LibwebsocketsConfig.cmake.in

${PROJECT_BINARY_DIR}${CMAKE_FILES_DIRECTORY}/LibwebsocketsConfig.cmake
    @ONLY)

# Generate version info for both build-tree and install-tree.
configure_file(${PROJECT_SOURCE_DIR}/cmake/LibwebsocketsConfigVersion.cmake.in
    ${PROJECT_BINARY_DIR}/LibwebsocketsConfigVersion.cmake
    @ONLY)

    set_target_properties(${LWS_LIBRARIES}
        PROPERTIES PUBLIC_HEADER "${HDR_PUBLIC}")

#
# Installation.
#

install(DIRECTORY include/libwebsockets
    DESTINATION "${LWS_INSTALL_INCLUDE_DIR}" COMPONENT dev_headers)

# Install libs and headers.
install(TARGETS ${LWS_LIBRARIES}
    EXPORT LibwebsocketsTargets
    LIBRARY DESTINATION "${LWS_INSTALL_LIB_DIR}${LIB_SUFFIX}" COMPONENT
libraries
    ARCHIVE DESTINATION "${LWS_INSTALL_LIB_DIR}${LIB_SUFFIX}" COMPONENT
libraries
    RUNTIME DESTINATION "${LWS_INSTALL_BIN_DIR}" COMPONENT libraries #
Windows DLLs
    PUBLIC_HEADER DESTINATION "${LWS_INSTALL_INCLUDE_DIR}" COMPONENT dev)

set(CPACK_COMPONENT_LIBRARIES_DISPLAY_NAME "Libraries")
set(CPACK_COMPONENT_DEV_DISPLAY_NAME "Development files")

# Install test apps.
if (NOT LWS_WITHOUT_TESTAPPS)
    install(TARGETS ${TEST_APP_LIST}
        RUNTIME DESTINATION ${LWS_INSTALL_EXAMPLES_DIR}
        COMPONENT examples)
    set(CPACK_COMPONENT_EXAMPLES_DISPLAY_NAME "Example files")
endif()

# lwsws
if (LWS_WITH_LWSWS)
    install(TARGETS lwsws
        RUNTIME DESTINATION "${LWS_INSTALL_BIN_DIR}" COMPONENT lwsws )
endif()

# Programs shared files used by the test-server.

```

```

if (NOT LWS_WITHOUT_TESTAPPS AND NOT LWS_WITHOUT_SERVER)
    install(FILES ${TEST_SERVER_DATA}
        DESTINATION share/libwebsockets-test-server
        COMPONENT examples)

    install(FILES "${PROJECT_SOURCE_DIR}/test-apps/private/index.html"
        DESTINATION share/libwebsockets-test-server/private
        COMPONENT examples)
if (LWS_WITH_CGI)
    set(CGI_TEST_SCRIPT "${PROJECT_SOURCE_DIR}/test-apps/lws-cgi-test.sh")
    install(FILES ${CGI_TEST_SCRIPT}
        PERMISSIONS OWNER_EXECUTE GROUP_EXECUTE WORLD_EXECUTE OWNER_READ
GROUP_READ WORLD_READ
        DESTINATION share/libwebsockets-test-server
        COMPONENT examples)
    endif()
endif()

if (NOT LWS_WITHOUT_TEST_SERVER AND NOT LWS_WITHOUT_SERVER AND NOT
LWS_WITHOUT_TESTAPPS)
    install(FILES test-apps/lws-ssh-test-keys;test-apps/lws-ssh-test-keys.pub
        DESTINATION share/libwebsockets-test-server
        COMPONENT examples)
endif()

# plugins

if (LWS_WITH_PLUGINS)
    install(TARGETS ${PLUGINS_LIST}
        PERMISSIONS OWNER_WRITE OWNER_EXECUTE GROUP_EXECUTE WORLD_EXECUTE
OWNER_READ GROUP_READ WORLD_READ
        DESTINATION share/libwebsockets-test-server/plugins
        COMPONENT plugins)

    if (NOT WIN32)
        install(FILES
plugins/deaddrop/assets/index.html;plugins/deaddrop/assets/deaddrop.js;plugins/
deaddrop/assets/deaddrop.css;plugins/deaddrop/assets/drop.svg
        DESTINATION share/libwebsockets-test-server/deaddrop
        COMPONENT plugins)
    endif()

if (LWS_WITH_SERVER_STATUS)
    install(FILES
plugins/server-status.html;plugins/server-status.js;plugins/server-
status.css;plugins/lwsws-logo.png
        DESTINATION share/libwebsockets-test-server/server-status
        COMPONENT examples)
endif()
if (LWS_WITH_GENERIC_SESSIONS)
    install(FILES
        plugins/generic-sessions/assets/lwsgs-logo.png
        plugins/generic-sessions/assets/seats.jpg
        plugins/generic-sessions/assets/failed-login.html
        plugins/generic-sessions/assets/lwsgs.js
        plugins/generic-sessions/assets/lwsgs.css
        plugins/generic-sessions/assets/post-register-fail.html

```

```

        plugins/generic-sessions/assets/post-register-ok.html
        plugins/generic-sessions/assets/post-verify-ok.html
        plugins/generic-sessions/assets/post-verify-fail.html
        plugins/generic-sessions/assets/sent-forgot-ok.html
        plugins/generic-sessions/assets/sent-forgot-fail.html
        plugins/generic-sessions/assets/post-forgot-ok.html
        plugins/generic-sessions/assets/post-forgot-fail.html
        plugins/generic-sessions/assets/index.html
    DESTINATION share/libwebsockets-test-server/generic-sessions
    COMPONENT examples)
install(FILEs plugins/generic-sessions/assets/successful-login.html
    DESTINATION share/libwebsockets-test-server/generic-sessions/needauth
    COMPONENT examples)
install(FILEs plugins/generic-sessions/assets/admin-login.html
    DESTINATION share/libwebsockets-test-server/generic-sessions/needadmin
    COMPONENT examples)
endif()

install(FILEs
    plugins/generic-table/assets/lwsgt.js
    plugins/generic-table/assets/index.html
    DESTINATION share/libwebsockets-test-server/generic-table
    COMPONENT examples)

endif()

# Install the LibwebsocketsConfig.cmake and LibwebsocketsConfigVersion.cmake
install(FILEs

"${PROJECT_BINARY_DIR}${CMAKE_FILES_DIRECTORY}/LibwebsocketsConfig.cmake"
    "${PROJECT_BINARY_DIR}/LibwebsocketsConfigVersion.cmake"
    DESTINATION "${LWS_INSTALL_CMAKE_DIR}" COMPONENT dev)

# Install exports for the install-tree.
if (LWS_WITH_EXPORT_LWSTARGETS)
    install(EXPORT LibwebsocketsTargets
        DESTINATION "${LWS_INSTALL_CMAKE_DIR}" COMPONENT dev)
endif()

# build subdir is not part of sources
set(CPACK_SOURCE_IGNORE_FILES ${CPACK_SOURCE_IGNORE_FILES} "/.git/" "/build/"
    "\\\\.tgz$" "\\\\.tar\\.\\.\\.\\.gz$")

# Most people are more used to "make dist" compared to "make package_source"
add_custom_target(dist COMMAND "${CMAKE_MAKE_PROGRAM}" package_source)

include(UserRPMTools)
if (RPMTools_FOUND)
    RPMTools_ADD_RPM_TARGETS(libwebsockets scripts/libwebsockets.spec)
endif()

message("-----")
message("  Settings:  (For more help do cmake -LH <srcpath>)")
message("-----")
message(" LWS_WITH_STATIC = ${LWS_WITH_STATIC}")
message(" LWS_WITH_SHARED = ${LWS_WITH_SHARED}")
message(" LWS_WITH_SSL = ${LWS_WITH_SSL} (SSL Support)")
message(" LWS_SSL_CLIENT_USE_OS_CA_CERTS = ${LWS_SSL_CLIENT_USE_OS_CA_CERTS}")
message(" LWS_WITH_WOLFSSL = ${LWS_WITH_WOLFSSL} (wolfSSL/CyaSSL replacement for

```



```

OpenSSL)")
if (LWS_WITH_WOLFSSL)
    message("    LWS_WOLFSSL_LIBRARIES = ${LWS_WOLFSSL_LIBRARIES}")
    message("    LWS_WOLFSSL_INCLUDE_DIRS = ${LWS_WOLFSSL_INCLUDE_DIRS}")
endif()
message(" LWS_WITH_MBEDTLS = ${LWS_WITH_MBEDTLS} (mbedtls replacement for
OpenSSL)")
message(" LWS_WITHOUT_BUILTIN_SHA1 = ${LWS_WITHOUT_BUILTIN_SHA1}")
message(" LWS_WITHOUT_BUILTIN_GETIFADDRS = ${LWS_WITHOUT_BUILTIN_GETIFADDRS}")
message(" LWS_WITHOUT_CLIENT = ${LWS_WITHOUT_CLIENT}")
message(" LWS_WITHOUT_SERVER = ${LWS_WITHOUT_SERVER}")
message(" LWS_LINK_TESTAPPS_DYNAMIC = ${LWS_LINK_TESTAPPS_DYNAMIC}")
message(" LWS_WITHOUT_TESTAPPS = ${LWS_WITHOUT_TESTAPPS}")
message(" LWS_WITHOUT_TEST_SERVER = ${LWS_WITHOUT_TEST_SERVER}")
message(" LWS_WITHOUT_TEST_SERVER_EXTPOLL = ${LWS_WITHOUT_TEST_SERVER_EXTPOLL}")
message(" LWS_WITHOUT_TEST_PING = ${LWS_WITHOUT_TEST_PING}")
message(" LWS_WITHOUT_TEST_CLIENT = ${LWS_WITHOUT_TEST_CLIENT}")
message(" LWS_WITHOUT_EXTENSIONS = ${LWS_WITHOUT_EXTENSIONS}")
message(" LWS_WITH_LATENCY = ${LWS_WITH_LATENCY}")
message(" LWS_WITHOUT_DAEMONIZE = ${LWS_WITHOUT_DAEMONIZE}")
message(" LWS_WITH_LIBEV = ${LWS_WITH_LIBEV}")
message(" LWS_WITH_LIBUV = ${LWS_WITH_LIBUV}")
message(" LWS_WITH_LIBEVENT = ${LWS_WITH_LIBEVENT}")
message(" LWS_IPV6 = ${LWS_IPV6}")
message(" LWS_UNIX_SOCKET = ${LWS_UNIX_SOCKET}")
message(" LWS_WITH_HTTP2 = ${LWS_WITH_HTTP2}")
message(" LWS_SSL_SERVER_WITH_ECDH_CERT = ${LWS_SSL_SERVER_WITH_ECDH_CERT}")
message(" LWS_MAX_SMP = ${LWS_MAX_SMP}")
message(" LWS_HAVE_PTHREAD_H = ${LWS_HAVE_PTHREAD_H}")
message(" LWS_WITH_CGI = ${LWS_WITH_CGI}")
message(" LWS_HAVE_OPENSSL_ECDH_H = ${LWS_HAVE_OPENSSL_ECDH_H}")
message(" LWS_HAVE_SSL_CTX_set1_param = ${LWS_HAVE_SSL_CTX_set1_param}")
message(" LWS_HAVE_RSA_SET0_KEY = ${LWS_HAVE_RSA_SET0_KEY}")
message(" LWS_WITH_HTTP_PROXY = ${LWS_WITH_HTTP_PROXY}")
message(" LIBHUBBUB_LIBRARIES = ${LIBHUBBUB_LIBRARIES}")
message(" PLUGINS = ${PLUGINS_LIST}")
message(" LWS_WITH_ACCESS_LOG = ${LWS_WITH_ACCESS_LOG}")
message(" LWS_WITH_SERVER_STATUS = ${LWS_WITH_SERVER_STATUS}")
message(" LWS_WITH_LEJP = ${LWS_WITH_LEJP}")
message(" LWS_WITH_LEJP_CONF = ${LWS_WITH_LEJP_CONF}")
message(" LWS_WITH_SMTP = ${LWS_WITH_SMTP}")
message(" LWS_WITH_GENERIC_SESSIONS = ${LWS_WITH_GENERIC_SESSIONS}")
message(" LWS_STATIC_PIC = ${LWS_STATIC_PIC}")
message(" LWS_WITH_RANGES = ${LWS_WITH_RANGES}")
message(" LWS_PLAT_OPTEE = ${LWS_PLAT_OPTEE}")
message(" LWS_WITH_ESP32 = ${LWS_WITH_ESP32}")
message(" LWS_WITH_ZIP_FOPS = ${LWS_WITH_ZIP_FOPS}")
message(" LWS_AVOID_SIGPIPE_IGN = ${LWS_AVOID_SIGPIPE_IGN}")
message(" LWS_WITH_STATS = ${LWS_WITH_STATS}")
message(" LWS_WITH_SOCKS5 = ${LWS_WITH_SOCKS5}")
message(" LWS_HAVE_SYS_CAPABILITY_H = ${LWS_HAVE_SYS_CAPABILITY_H}")
message(" LWS_HAVE_LIBCAP = ${LWS_HAVE_LIBCAP}")
message(" LWS_WITH_PEER_LIMITS = ${LWS_WITH_PEER_LIMITS}")
message(" LWS_HAVE_ATOLL = ${LWS_HAVE_ATOLL}")
message(" LWS_HAVE__ATOI64 = ${LWS_HAVE__ATOI64}")
message(" LWS_HAVE_STAT32I64 = ${LWS_HAVE_STAT32I64}")
message(" LWS_HAS_INTPTR_T = ${LWS_HAS_INTPTR_T}")
message(" LWS_WITH_EXPORT_LWSTARGETS = ${LWS_WITH_EXPORT_LWSTARGETS}")
message(" LWS_WITH_ABSTRACT = ${LWS_WITH_ABSTRACT}")

```

```

message("-----")

# These will be available to parent projects including libwebsockets using
add_subdirectory()
set(LIBWEBSOCKETS_LIBRARIES ${LWS_LIBRARIES} CACHE STRING "Libwebsocket libraries")
if (LWS_WITH_STATIC)
    set(LIBWEBSOCKETS_LIBRARIES_STATIC websockets CACHE STRING "Libwebsocket
static library")
endif()
if (LWS_WITH_SHARED)
    set(LIBWEBSOCKETS_LIBRARIES_SHARED websockets_shared CACHE STRING
"Libwebsocket shared library")
endif()

if (LWS_WITH_MINIMAL_EXAMPLES)
    MACRO(SUBDIRLIST result curdir)
        FILE(GLOB children RELATIVE ${curdir} ${curdir}/*)
        SET(dirlist "")

        FOREACH(child ${children})
            IF(IS_DIRECTORY ${curdir}/${child})
                LIST(APPEND dirlist ${child})
            ENDIF()
        ENDFOREACH()

        SET(${result} ${dirlist})
    ENDMACRO()

    SUBDIRLIST(SUBDIRS "${PROJECT_SOURCE_DIR}/minimal-examples")
    FOREACH(subdir ${SUBDIRS})

        SUBDIRLIST(SUBDIRS2 "${PROJECT_SOURCE_DIR}/minimal-examples/${subdir}")
        FOREACH(subdir2 ${SUBDIRS2})
            if (EXISTS "${PROJECT_SOURCE_DIR}/minimal-examples/${subdir}/${subdir2}/CMakeLists.txt")
                message("Processing ${PROJECT_SOURCE_DIR}/minimal-
examples/${subdir}/${subdir2}")
                add_subdirectory("${PROJECT_SOURCE_DIR}/minimal-examples/${subdir}/${subdir2}")
            endif()
        ENDFOREACH()
    ENDFOREACH()
ENDIF()

# This must always be last!
include(CPack)

```