Compilation

Jump to bottom

akallabeth edited this page on Feb 20 · 55 revisions

Table of Contents

Options

here are the `CMake option` values available to adjust compilation (along with default values) The list is not updated regularily, so check the `CMakeLists.txt` files for `option` and `cmake_dependent_option` values

- `WITH_DEBUG_URBDRC` Dump data send/received in URBDRC channel `OFF`
- `BUILD_SHARED_LIBS` Build shared libraries `ON`
- `EXPORT_ALL_SYMBOLS` Export all symbols form library `OFF`
- `BUILD_TESTING` Build library unit tests `ON`
- `WITH_LIBRARY_VERSIONING` Use library version triplet `ON`
- `WITH_SHADOW` Compile with shadow server `ON`
- `WITH_PROXY` Compile with proxy server `ON`
- `WITH_PLATFORM_SERVER` Compile with platform server `ON`

- `WITH_PROXY_EMULATE_SMARTCARD` Compile proxy smartcard emulation `OFF`
- `WITH_PROXY_APP` Compile proxy application `ON`
- `WITH_PROXY_MODULES` Compile proxy modules `ON`
- `WITH_SMARTCARD_PCSC` Enable smartcard PCSC backend `ON`
- `WITH_OPENSC_PKCS11_LINKED` Directly link opensc-pkcs11 `OFF`
- `WITH_DEBUG_SCHANNEL` Compile support for SCHANNEL debug `OFF`
- `WITH_KRB5` Compile support for kerberos authentication. ${KRB5_DEFAULT}`
- `WITH_KRB5_NO_NTLM_FALLBACK` Do not fall back to NTLM if no kerberos ticket available OFF `WITH_KRB5` OFF`
- `USE_UNWIND` Use unwind.h to generate backtraces `ON`
- `WITH_LODEPNG` build WinPR with PNG support `OFF`
- `WITH_LIBRARY_VERSIONING` Use library version triplet `ON`
- `BUILD_SHARED_LIBS` Build shared libraries `ON`
- `EXPORT_ALL_SYMBOLS` Export all symbols form library `OFF`
- `WITH_VERBOSE_WINPR_ASSERT` Compile with verbose WINPR_ASSERT. `ON`
- `WITH_WINPR_TOOLS` Build WinPR helper binaries `ON`
- `WITH_WINPR_DEPRECATED` Build WinPR deprecated symbols `OFF`
- `WITH_DEBUG_THREADS` Print thread debug messages, enables handle dump ${DEFAULT_DEBUG_OPTION}`
- `WITH_DEBUG_EVENTS` Print event debug messages, enables handle dump ${DEFAULT_DEBUG_OPTION}`
- `WITH_DEBUG_SYMBOLS` Pack debug symbols to installer `OFF`
- `WITH_NATIVE_SSPI` Use native SSPI modules ${NATIVE_SSPI}`
- `WITH_SMARTCARD_INSPECT` Enable SmartCard API Inspector `OFF`
- `WITH_DEBUG_MUTEX` Print mutex debug messages ${DEFAULT_DEBUG_OPTION}`
- `WITH_INTERNAL_RC4` Use compiled in rc4 functions instead of OpenSSL/MBedTLS `OFF`
- `WITH_INTERNAL_MD4` Use compiled in md4 hash functions instead of OpenSSL/MBedTLS `OFF`
- `WITH_INTERNAL_MD5` Use compiled in md5 hash functions instead of OpenSSL/MBedTLS `OFF`
- `WITH_UNICODE_BUILTIN` Use built-in Unicode conversion (don't use system-provided libraries OFF`
- `SSPI_DLL` Define and export SSPI API symbols for usage as a Windows SSPI DLL replacement `OFF`
- `WITH_DEBUG_NTLM` Print NTLM debug messages ${DEFAULT_DEBUG_OPTION}`
- `WITH_DEBUG_NLA` Print authentication related debug messages. ${DEFAULT_DEBUG_OPTION}`
- `WITH_POLL` Check for and include poll.h `ON`

- `WITH_PKCS11` encryption, certificate validation, hashing functions ${PKCS11_DEFAULT}`
- `BUILD_SHARED_LIBS` Build shared libraries `ON`
- `EXPORT_ALL_SYMBOLS` Export all symbols form library `OFF`
- `BUILD_TESTING` Build library unit tests `ON`
- `WITH_LIBRARY_VERSIONING` Use library version triplet `ON`
- `UWAC_HAVE_PIXMAN_REGION` Use PIXMAN or FreeRDP for region calculations `NOT FREERDP_UNIFIED_BUILD`
- `BUILD_TESTING` Build automated tests. `ON`
- `WITH_WEBVIEW` Build with WebView support for AAD login popup browser `OFF`
- `WITH_WEBVIEW_QT` Build with QtWebEngine support for AAD login broweser popup `OFF`
- `CMAKE_COLOR_MAKEFILE` colorful CMake makefile `ON`
- `CMAKE_VERBOSE_MAKEFILE` verbose CMake makefile `ON`
- `CMAKE_POSITION_INDEPENDENT_CODE` build with position independent code (-fPIC or -fPIE `ON`
- `WITH_DEBUG_SDL_EVENTS` [dangerous, not for release builds!] Debug SDL events `OFF`
- `WITH_DEBUG_SDL_KBD_EVENTS` [dangerous, not for release builds!] Debug SDL keyboard events `OFF`
- `WITH_WIN_CONSOLE` Build ${PROJECT_NAME} with console support `ON`
- `WITH_WINDOWS_CERT_STORE` Build ${MODULE_NAME} with additional certificate validation against windows certificate store `ON`
- `WITH_WIN_CONSOLE` Build ${MODULE_NAME} with console support `OFF`
- `WITH_PROGRESS_BAR` Build ${MODULE_NAME} with connect progress bar (Windows 7+ or 2008 R2+ `ON`
- `CMAKE_COLOR_MAKEFILE` colorful CMake makefile `ON`
- `CMAKE_VERBOSE_MAKEFILE` verbose CMake makefile `ON`
- `CMAKE_POSITION_INDEPENDENT_CODE` build with position independent code (-fPIC or -fPIE `ON`
- `CMAKE_COLOR_MAKEFILE` colorful CMake makefile `ON`
- `CMAKE_VERBOSE_MAKEFILE` verbose CMake makefile `ON`
- `CMAKE_POSITION_INDEPENDENT_CODE` build with position independent code (-fPIC or -fPIE `ON`
- `WITH_XINERAMA` [X11] enable xinerama `ON`
- `WITH_XEXT` [X11] enable Xext `ON`
- `WITH_XCURSOR` [X11] enalbe Xcursor `ON`
- `WITH_XV` [X11] enable Xv `ON`
- `WITH_XI` [X11] enalbe Xi `ON`
- `WITH_XRENDER` [X11] enable XRender `ON`
- `WITH_XRANDR` [X11] enable XRandR `ON`

- `WITH_XFIXES` [X11] enable Xfixes `ON`
- `WITH_CLIENT_INTERFACE` Build clients as a library with an interface `OFF`
- `WITH_CLIENT_MAC` Build native mac client `ON`
- `WITH_FUSE` Build clipboard with FUSE file copy support ${OPT_FUSE_DEFAULT}`
- `CMAKE_COLOR_MAKEFILE` colorful CMake makefile `ON`
- `CMAKE_VERBOSE_MAKEFILE` verbose CMake makefile `ON`
- `CMAKE_POSITION_INDEPENDENT_CODE` build with position independent code (-fPIC or -fPIE ON`
- `WITH_X11` build X11 client/server ${OPT_DEFAULT_VAL}`
- `CMAKE_INTERPROCEDURAL_OPTIMIZATION` Enable LTO linking TRUE`
- `WITH_SMARTCARD_EMULATE` Emulate smartcards instead of redirecting readers `OFF`
- `WITH_FREERDP_DEPRECATED` Build FreeRDP deprecated symbols `OFF`
- `WITH_FREERDP_DEPRECATED_COMMANDLINE` Build FreeRDP deprecated command line options `OFF`
- `BUILD_SHARED_LIBS` Build shared libraries ${LIB_DEFAULT}`
- `EXPORT_ALL_SYMBOLS` Export all symbols form library `OFF`
- `WITH_AAD` Compile with support for Azure AD authentication `ON`
- `WITH_OPENH264_LOADING` Use LoadLibrary to load openh264 at runtime `OFF`
- `WITH_VERBOSE_WINPR_ASSERT` Compile with verbose WINPR_ASSERT. `ON`
- `WITH_RDTK` build rdtk toolkit `ON`

Dependencies

Server and Client applications

- OpenSSL or MbedTLS for encryption/decryption

Client

- `uriparser` for parsing clipboard urls (optional, linux only)
- `libusb` for raw USB redirection
- Sound (microphone and playback) requires a encoder/decoder for various formats. That can be:
  - `FFMPEG` does all encoding/decoding
  - Use `gsm`, `faad2`, `faac` for `GSM` and `AAC` support
  - Use `soxr` for sound resampling

- Video decoders (For GFX H264 and `/video` redirection support):
  - `H264` can be supported though
    - `OpenH264` (working only with `1.8.0` until https://github.com/cisco/openh264/issues/3476 is resolved)
    - `FFMPEG`
- Image scalers (for `smart-sizing` and some HighDPI features of RDP)
  - `swscale`
  - `cairo`
- Old (Windows 7) style media redirection (deactivated by default) also requires:
  - GStreamer (0.1 or 1.0)
- Platform/feature dependent options:
  - Android MediaCodec (h264)
  - Windows MediaFramework (h264)
  - `ALSA` (sound)
  - `PULSE` (sound)
  - `OSS` (sound)
  - `CUPS` (printing)
  - `FUSE` (file clipboard)
  - `ICU` (unicode)
  - `OPENSLES` (sound)
  - `PCSC` (smartcard)
  - `cJSON` (AVD/AAD auth)
  - `libsdl2-ttf`, `libsdl2-image` and `libsdl2` (SDL client)

Legacy issues (`rc4`, `md4`, `md5`)

`RDP` uses a couple of legacy algorithms deep within the protocol which might lead to problems at runtime. Ensure, that

1. `rc4`, `md4` and `md5` are supported by your `SSL` library

- Also ensure you have the correct runtime configuration that allows these hashes

1. Compile in `rc4`, `md4` or `md5` with `WITH_INTERNAL_RC4`, `WITH_INTERNAL_MD4` or `WITH_INTERNAL_MD5`

Compilation

First, you'll need to get the dependencies and sources. Follow the next instructions step by step (unless the dependency is already installed on your system)

Common instructions

See options list of compilation options for `FreeRDP`.

Build OpenSSL

1. to build the library static add `no-shared` to the `Configure` options
2. basic steps are similar on all OS, but the detailed `./Configure` options might vary, check https://wiki.openssl.org/index.php/Compilation_and_Installation for further details

```
git clone -b openssl-3.1.1 https://github.com/openssl/openssl.git
cd openssl
./Configure mingw64 --cross-compile-prefix=x86_64-w64-mingw32- --prefix=<your install
path> --libdir=<lib|lib64>
make -j build_sw
make -j install_sw
```

LibreSSL

As an alternative to `OpenSSL` `LibreSSL` can be used. Add `-DWITH_LIBRESSL=ON` to your `FreeRDP` build to force detection of `LibreSSL`

Build static or shared lib, set `-DBUILD_SHARED_LIBS=ON|OFF` accordingly

```
git clone -b v3.8.2 https://github.com/libressl/portable.git libressl
cmake -GNinja \
    -DCMAKE_TOOLCHAIN_FILE=<full path to toolchain.cmake> \
    -B libressl-build \
    -S libressl \
    -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_SKIP_INSTALL_ALL_DEPENDENCY=ON \
    -DCMAKE_INSTALL_PREFIX=<your install path> \
    -DLIBRESSL_APPS=OFF \
    -DLIBRESSL_TESTS=OFF
cmake --build libressl-build
cmake --install libressl-build
```

Build zlib

Build static and shared lib

```
git clone --depth 1 -b v1.3 https://github.com/madler/zlib.git
cmake -GNinja \
    -DCMAKE_TOOLCHAIN_FILE=<full path to toolchain.cmake> \
    -B zlib-build \
    -S zlib \
    -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_SKIP_INSTALL_ALL_DEPENDENCY=ON \
    -DCMAKE_INSTALL_PREFIX=<your install path> \
    -DLIBRESSL_APPS=OFF \
    -DLIBRESSL_TESTS=OFF
cmake --build zlib-build
cmake --install zlib-build
```

Build uriparser

Build static or shared lib, set `-DBUILD_SHARED_LIBS=ON|OFF` accordingly

```
git clone --depth 1 -b uriparser-0.9.7 https://github.com/uriparser/uriparser.git
cmake -GNinja \
    -DCMAKE_TOOLCHAIN_FILE=<full path to toolchain.cmake> \
    -B uriparser-build \
    -S uriparser \
    -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_SKIP_INSTALL_ALL_DEPENDENCY=ON \
    -DCMAKE_INSTALL_PREFIX=<your install path> \
    -DURIPARSER_BUILD_DOCS=OFF \
    -DURIPARSER_BUILD_TESTS=OFF
cmake --build uriparser-build
cmake --install uriparser-build
```

Build cJSON

Build static and shared libs in one go

```
git clone --depth 1 -b v1.7.16 https://github.com/DaveGamble/cJSON.git
cmake -GNinja \
    -DCMAKE_TOOLCHAIN_FILE=<full path to toolchain.cmake> \
    -B cJSON-build \
    -S cJSON \
    -DCMAKE_BUILD_TYPE=Release \
```

```
    -DCMAKE_SKIP_INSTALL_ALL_DEPENDENCY=ON \
    -DCMAKE_INSTALL_PREFIX=<your install path> \
    -DENABLE_CJSON_TEST=OFF \
    -DBUILD_SHARED_AND_STATIC_LIBS=ON
cmake --build cJSON-build
cmake --install cJSON-build
```

Build SDL2

Build static and shared libs in one go

```
git clone --depth 1 -b release-2.28.1 https://github.com/libsdl-org/SDL.git
cmake -GNinja \
    -DCMAKE_TOOLCHAIN_FILE=<full path to toolchain.cmake> \
    -B SDL-build \
    -S SDL \
    -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_SKIP_INSTALL_ALL_DEPENDENCY=ON \
    -DCMAKE_INSTALL_PREFIX=<your install path> \
    -DSDL_TEST=OFF \
    -DSDL_TESTS=OFF \
    -DSDL_STATIC_PIC=ON
cmake --build SDL-build
cmake --install SDL-build
```

Build SDL2_ttf

Build static or shared lib, set `-DBUILD_SHARED_LIBS=ON|OFF` accordingly

```
git clone --depth 1 --shallow-submodules --recurse-submodules -b release-2.20.2
https://github.com/libsdl-org/SDL_ttf.git
cmake -GNinja \
    -DCMAKE_TOOLCHAIN_FILE=<full path to toolchain.cmake> \
    -B SDL_ttf-build \
    -S SDL_ttf \
    -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_SKIP_INSTALL_ALL_DEPENDENCY=ON \
    -DCMAKE_INSTALL_PREFIX=<your install path> \
    -DSDL2TTF_HARFBUZZ=ON \
    -DSDL2TTF_FREETYPE=ON \
    -DSDL2TTF_VENDORED=ON \
    -DFT_DISABLE_ZLIB=OFF \
    -DSDL2TTF_SAMPLES=OFF
```

```
cmake --build SDL_ttf-build
cmake --install SDL_ttf-build
```

Build SDL2_image

Build static or shared lib, set `-DBUILD_SHARED_LIBS=ON|OFF` accordingly

```
git clone --depth 1 --shallow-submodules --recurse-submodules -b release-2.8.1
https://github.com/libsdl-org/SDL_image.git
cmake -GNinja \
    -DCMAKE_TOOLCHAIN_FILE=<full path to toolchain.cmake> \
    -B SDL_image-build \
    -S SDL_image \
    -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_SKIP_INSTALL_ALL_DEPENDENCY=ON \
    -DCMAKE_INSTALL_PREFIX=<your install path> \
    -DSDL2IMAGE_SAMPLES=OFF \
    -DSDL2IMAGE_DEPS_SHARED=OFF
cmake --build SDL_image-build
cmake --install SDL_image-build
```

libusb (optional)

Build static or shared lib, set `-DLIBUSB_BUILD_SHARED_LIBS=ON|OFF` accordingly NOTE: we're using a 3rd party repo here to build with `CMake` as that is easier to handle for cross compilation

```
git clone --depth 1 --shallow-submodules --recurse-submodules -b v1.0.26
https://github.com/libusb/libusb-cmake.git
cmake -GNinja \
    -DCMAKE_TOOLCHAIN_FILE=<full path to toolchain.cmake> \
    -B libusb-cmake-build \
    -S libusb-cmake \
    -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_SKIP_INSTALL_ALL_DEPENDENCY=ON \
    -DCMAKE_INSTALL_PREFIX=<your install path> \
    -DLIBUSB_BUILD_EXAMPLES=OFF \
    -DLIBUSB_BUILD_TESTING=OFF \
    -DLIBUSB_ENABLE_DEBUG_LOGGING=OFF
cmake --build libusb-cmake-build
cmake --install libusb-cmake-build
```

FFMPEG (optional)

see https://trac.ffmpeg.org/wiki/CompilationGuide/CrossCompilingForWindows for instructions

```
git clone --depth 1 -b n6.0 https://github.com/FFmpeg/FFmpeg.git

mkdir ffmpeg-build
cd ffmpeg-build
../FFmpeg/configure --arch=x86_64 --target-os=mingw64 --cross-prefix=x86_64-w64-mingw32- -
-prefix=<your install dir>
make -j
make -j install
```

openh264 (optional)

Build static and shared libs in one go

https://mesonbuild.com/Cross-compilation.html

meson toolchain file

```
[binaries]
c = 'x86_64-w64-mingw32-gcc'
cpp = 'x86_64-w64-mingw32-g++'
ar = 'x86_64-w64-mingw32-ar'
strip = 'x86_64-w64-mingw32-strip'
exe_wrapper = 'wine64'

[host_machine]
system = 'windows'
cpu_family = 'x86_64'
cpu = 'x86_64'
endian = 'little'

[properties]
sysroot = '/usr/x86_64-w64-mingw32'
```

```
git clone --depth 1 -b v2.3.1 https://github.com/cisco/openh264.git
meson setup \
    -Dprefix=<your install path> \
    -Db_pie=true \
    -Db_lto=true \
    -Dbuildtype=release \
    -Dpkgconfig.relocatable=true \
    -Dtests=disabled \
    -Ddefault_library=both \
```

```
    openh264-build \
    openh264
ninja -C openh264-build
ninja -C openh264-build install
```

Build FreeRDP

- Disable USB with `-DCHANNEL_URBDRC=OFF` (libusb required for `ON`)
- Enable OpenH264 with `-DWITH_OPENH264=ON`
- Enable FFMPEG with `-DWITH_FFMPEG=ON -DWITH_SWSCALE=ON -DWITH_DSP_FFMPEG=ON`
- Build static binaris/libraries with `-DBUILD_SHARED_LIBS=OFF` unless further instructions are available for dependency
- Build fully static (including runtime) with `-DCMAKE_MSVC_RUNTIME_LIBRARY=MultiThreaded` (windows) `set(CMAKE_FIND_LIBRARY_SUFFIXES ".a")` (add to toolchain file, others)

NOTE: Check your c++ compiler/libs are up to date, the ones shipped with debian 12 might lack some features. In that case disable our C++ projecst with `-DWITH_CLIENT_SDL=OFF -DWITH_PROXY_MODULES=OFF`

```
git clone --depth 1 https://github.com/freerdp/freerdp.git
cmake -GNinja \
    -DCMAKE_TOOLCHAIN_FILE=<full path to toolchain.cmake> \
    -B freerdp-build \
    -S freerdp \
    -DCMAKE_BUILD_TYPE=Release \
    -DCMAKE_SKIP_INSTALL_ALL_DEPENDENCY=ON \
    -DCMAKE_INSTALL_PREFIX=<your install path> \
    -DWITH_SERVER=ON \
    -DWITH_SAMPLE=ON \
    -DWITH_PLATFORM_SERVER=OFF \
    -DUSE_UNWIND=OFF \
    -DWITH_SWSCALE=OFF \
    -DWITH_FFMPEG=OFF \
    -DWITH_WEBVIEW=OFF
cmake --build freerdp-build
cmake --install freerdp-build
```

Linux Specifics

Build a Flatpak

Dependencies

- Install `flatpak` and configure "flathub.org":https://flatpak.org/setup/

Compilation

- Run `flatpak-builder --repo=repo <build dir> packaging/flatpak/com.freerdp.FreeRDP.json` from the checkout root
- For a easy to install package run `flatpak build-bundle repo com.freerdp.FreeRDP.flatpak com.freerdp.FreeRDP --runtime-repo=https://flathub.org/repo/flathub.flatpakrepo`

Installation

- `flatpak install com.freerdp.FreeRDP.flatpak` for bundle

Traditional packages (`deb` and `rpm`)

Nightly build for debian based systems

`sudo apt build-dep freerdp-x11` or `sudo apt build-dep freerdp2-x11`

`ln -s packaging/deb/freerdp-nightly debian  dpkg-buildpackage`

Install the suggested base dependencies:

debian based

1. ensure you have enabled `contrib non-free` on `debian` in `/etc/apt/sources.list`
2. ensure you have enabled `universe multiverse` on `ubuntu` in `/etc/apt/sources.list`
3. the following installs all (even optional) libraries to build everything `FreeRDP` is capable of supporting:

```
sudo apt-get install \
    ninja-build \
    build-essential \
    git-core \
    debhelper \
    cdbs \
    dpkg-dev \
    cmake \
    cmake-curses-gui \
    clang-format \
    ccache \
    opencl-c-headers \
    ocl-icd-opencl-dev \
```

```
        libmp3lame-dev \
        libopus-dev \
        libsoxr-dev \
        libpam0g-dev \
        pkg-config \
        xmlto \
        libssl-dev \
        docbook-xsl \
        xsltproc \
        libxkbfile-dev \
        libx11-dev \
        libwayland-dev \
        libxrandr-dev \
        libxi-dev \
        libxrender-dev \
        libxext-dev \
        libxinerama-dev \
        libxfixes-dev \
        libxcursor-dev \
        libxv-dev \
        libxdamage-dev \
        libxtst-dev \
        libcups2-dev \
        libpcsclite-dev \
        libasound2-dev \
        libpulse-dev \
        libgsm1-dev \
        libusb-1.0-0-dev \
        uuid-dev \
        libxml2-dev \
        libfaad-dev \
        libfaac-dev \
        libsdl2-dev \
        libsdl2-ttf-dev \
        libcjson-dev \
        libpkcs11-helper-dev \
        liburiparser-dev \
        libkrb5-dev \
        libsystemd-dev \
        libfuse3-dev \
        libswscale-dev \
        libcairo2-dev \
        libavutil-dev \
        libavcodec-dev \
        libswresample-dev \
        libwebkit2gtk-4.0-dev \
        libpkcs11-helper1-dev
```

Fedora 39 and close relatives

1. ensure you have 3rdparty repositories enabled
2. the following installs all required dependencies (even optional ones) to build `FreeRDP`:

```
sudo dnf -y install \
    ninja-build \
    cups-devel \
    systemd-devel \
    libuuid-devel \
    pulseaudio-libs-devel \
    gcc-c++ libXrandr-devel \
    gsm-devel \
    gcc \
    cmake \
    ccache \
    git-clang-format \
    pam-devel \
    fuse3-devel \
    opus-devel \
    lame-devel \
    ocl-icd-devel \
    docbook-style-xsl \
    openssl-devel \
    libX11-devel \
    libXext-devel \
    libXinerama-devel \
    libXcursor-devel \
    libXi-devel \
    libXdamage-devel \
    libXv-devel \
    libxkbfile-devel \
    alsa-lib-devel \
    openh264-devel \
    libavcodec-free-devel \
    libavformat-free-devel \
    libavutil-free-devel \
    libswresample-free-devel \
    libswscale-free-devel \
    libusb1-devel \
    uriparser-devel \
    SDL2-devel \
    SDL2_ttf-devel \
    pkcs11-helper-devel \
    webkit2gtk4.0-devel \
    krb5-devel \
    cjson-devel \
    cairo-devel \
```

```
      soxr-devel \
      wayland-devel \
      wayland-protocols-devel
```

compilation & installation

proceed with the build as described in Build FreeRDP

Android specifics

1. Builds have been tested on linux hosts, so your mileage on MacOS and Windows may vary (please add additional instructions here if you are using such a build host)
2. Android SDK and NDK need to be installed

To build first simply run:

```
./scripts/android-build-freerdp.sh \
  --ndk <path_to_ndk> \
  --sdk <path_to_sdk> \
  --openh264-ndk <path_to_ndk_15c> \
  --conf android-build-release.conf
```

the script can be found in the repo:

android-build-freerdp.sh

configuration file examples at android-build.conf

for the native dependency build and then open the project in AndroidStudio / run `./gradlew` just like with any other android project.

macOS Specifics

Using a package manager (xfreerdp)

If you're using MacPorts then just install FreeRDP port:

```
sudo port install FreeRDP
```

FreeRDP is now available as a homebrew recipe.

```
brew install freerdp
```

Use it like `xfreerdp ...`

Compiling from sources (native and SDL client)

1. create a toolchain file for your system:
    i. `-DCMAKE_OSX_ARCHITECTURES="arm64;x86_64"` for a universal build (or just set the architecture if you only want one)
    ii. `-DCMAKE_OSX_DEPLOYMENT_TARGET=14` to set `-mmacosx-version-min` to macOS 14 (or any other version you desire and is supported by your toolchain)
2. with that toolchain then proceed with the build as described in common

Ready to use build script

The following script downloads builds and installs FreeRDP on mac.

1. It installs everything to `install/` subdirectory of where the script is called from
2. It does not link external stuff from `brew` or `ports`
3. It is build without `FFMPEG`

bundle-mac-os.sh

Compiling for Visual Studio or Windows SDK

Prerequisites

- Visual Studio or Windows SDK
- "meson":https://mesonbuild.com/ (also contains 'ninja')
- "cmake":https://cmake.org/
- "nasm":https://www.nasm.us
- "perl":https://www.perl.org

1. Start a visual studio command prompt (or windows sdk prompt) (`vcvars_all.bat`)
    - Ensure the correct environment is used (32 or 64 bit)
2. Ensure `meson` , `cmake` , `ninja` , `nasm` , `perl` and `git` are in your `PATH`
3. from that prompt then proceed with the build as described in common

Compiling for MINGW

1. Install dependencies and compilers for mingw

2. Create a toolchain file for `CMake` to pick up libraries/compilers https://cmake.org/cmake/help/book/mastering-cmake/chapter/Cross%20Compiling%20With%20CMake.html (fedora has `mingw64-cmake` doing that for you)
3. `cmake -GNinja -B<builddir> -S<sourcedir> -DCMAKE_TOOLCHAIN_FILE=<toolchain.cmake> -DCMAKE_INSTALL_PREFIX=<install prefix>`
4. `cmake --build <builddir> --target install`

Fedora build

with Fedora it is quite easy to do, just

1. Install

```
sudo dnf install mingw64-zlib.noarch mingw64-winpthreads.noarch mingw64-pixman.noarch mingw64-openssl.noarch mingw64-libstdc++.x86_64 mingw64-libgcc.x86_64 mingw64-jsoncpp.noarch mingw64-SDL2.noarch mingw64-SDL2_ttf.noarch mingw64-crt.noarch mingw64-fontconfig.noarch mingw64-gsm.noarch mingw64-harfbuzz.noarch mingw64-gcc-c++.x86_64 mingw64-gcc.x86_64 mingw64-libgcc.x86_64
```

1. then proceed with the build as described in common (replace `cmake` configuration calls with `mingw64-cmake`)

Debian build

Toolchain file for debian (mingw64)

```
SET(CMAKE_SYSTEM_NAME Windows CACHE STRING "toolchain default")

SET(CMAKE_SYSTEM_PROCESSOR amd64 CACHE STRING "toolchain default")

SET(CMAKE_C_COMPILER /usr/bin/x86_64-w64-mingw32-gcc CACHE STRING "toolchain default")
SET(CMAKE_CXX_COMPILER /usr/bin/x86_64-w64-mingw32-g++ CACHE STRING "toolchain default")
SET(CMAKE_RC_COMPILER_INIT /usr/bin/x86_64-w64-mingw32-windres CACHE STRING "toolchain default")
SET(CMAKE_RC_COMPILER /usr/bin/x86_64-w64-mingw32-windres CACHE STRING "toolchain default")
SET(CMAKE_AR /usr/bin/x86_64-w64-mingw32-ar CACHE STRING "toolchain default")
SET(CMAKE_C_COMPILER_AR /usr/bin/x86_64-w64-mingw32-ar CACHE STRING "toolchain default")
SET(CMAKE_CXX_COMPILER_AR /usr/bin/x86_64-w64-mingw32-ar CACHE STRING "toolchain default")
SET(CMAKE_RANLIB /usr/bin/x86_64-w64-mingw32-ranlib CACHE STRING "toolchain default")
SET(CMAKE_C_COMPILER_RANLIB /usr/bin/x86_64-w64-mingw32-ranlib CACHE STRING "toolchain default")
SET(CMAKE_CXX_COMPILER_RANLIB /usr/bin/x86_64-w64-mingw32-ranlib CACHE STRING "toolchain
```

```
default")
SET(CMAKE_LINKER /usr/bin/x86_64-w64-mingw32-ld CACHE STRING "toolchain default")
SET(CMAKE_NM /usr/bin/x86_64-w64-mingw32-nm CACHE STRING "toolchain default")
SET(CMAKE_READELF /usr/bin/x86_64-w64-mingw32-readelf CACHE STRING "toolchain default")
SET(CMAKE_OBJCOPY /usr/bin/x86_64-w64-mingw32-objcopy CACHE STRING "toolchain default")
SET(CMAKE_OBJDUMP /usr/bin/x86_64-w64-mingw32-objdump CACHE STRING "toolchain default")

SET(CMAKE_SYSROOT /usr/x86_64-w64-mingw32 CACHE STRING "toolchain default")

set(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER CACHE STRING "toolchain default")
set(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY CACHE STRING "toolchain default")
set(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY CACHE STRING "toolchain default")
set(CMAKE_FIND_ROOT_PATH_MODE_PACKAGE ONLY CACHE STRING "toolchain default")
```

For `CMake` based projects you need to add the following to your command line (or add to the toolchain with the `set(VARIABLE_NAME <install prefix>)` syntax) to control installation and library search locations:

```
-DCMAKE_INSTALL_PREFIX=<install path> \
-DCMAKE_PREFIX_PATHS=<install path> \
-DCMAKE_FIND_ROOT_PATH=<install path>
```

then proceed with the build as described in common

Compilation

Clone this wiki locally

```
https://github.com/FreeRDP/FreeRDP.wiki.git
```