# Windows build files for DLF\_TO\_CSV (mail branch)

This canvas contains all files to add to a new windows/ folder on a mail branch. Each file is provided verbatim in a code block — copy each file into your repository under windows/ and commit.

## File: windows/build-cross-ubuntu.sh

#!/usr/bin/env bash  
# build-cross-ubuntu.sh  
# Cross-compile libdivecomputer and dlf\_parser\_helper for Windows x86\_64 on Ubuntu 24.04  
# - idempotent: cleans previous dist/ and temporary build dirs  
# - calls check-deps.sh  
# - builds libdivecomputer with host=x86\_64-w64-mingw32  
# - builds dlf\_parser\_helper.exe using x86\_64-w64-mingw32-gcc  
# - collects dlls (libdivecomputer.dll and common mingw runtime dlls) into windows/dist/win-x86\_64  
  
set -euo pipefail  
IFS=$'\n\t'  
  
BASE\_DIR="$(cd "$(dirname "$0")" && pwd)"  
DIST\_DIR="$BASE\_DIR/dist/win-x86\_64"  
TMP\_ROOT="/tmp/libdivecomputer\_build"  
LIBDC\_REPO="https://github.com/libdivecomputer/libdivecomputer.git"  
REPO\_ROOT="$(cd "$BASE\_DIR/.." && pwd)" # repo root assumed to be parent of windows/  
  
# allow override with environment variables  
TARGET\_HOST="${TARGET\_HOST:-x86\_64-w64-mingw32}"  
NPROC=${NPROC:-$(nproc)}  
  
echo "=== Windows cross-build: start ==="  
  
# 0) run dependency check  
"$BASE\_DIR/check-deps.sh"  
  
# 1) prepare directories (idempotent)  
rm -rf "$DIST\_DIR"  
mkdir -p "$DIST\_DIR"  
  
rm -rf "$TMP\_ROOT"  
mkdir -p "$TMP\_ROOT"  
BUILD\_DIR="$TMP\_ROOT/build"  
mkdir -p "$BUILD\_DIR"  
  
# 2) install required apt packages (non-interactive). This may ask for sudo.  
if ! dpkg -s mingw-w64 >/dev/null 2>&1; then  
 echo "Installing system packages (requires sudo)..."  
 sudo apt update  
 sudo apt install -y git build-essential mingw-w64 pkg-config autoconf automake libtool wget unzip  
fi  
  
# 3) clone and build libdivecomputer  
cd "$BUILD\_DIR"  
if [ -d libdivecomputer ]; then rm -rf libdivecomputer; fi  
git clone --depth 1 "$LIBDC\_REPO" libdivecomputer  
cd libdivecomputer  
  
echo "Running autoreconf..."  
autoreconf --install || true  
  
STAGING\_PREFIX="$BUILD\_DIR/install"  
mkdir -p "$STAGING\_PREFIX"  
  
export CC="${TARGET\_HOST}-gcc"  
export AR="${TARGET\_HOST}-ar"  
export RANLIB="${TARGET\_HOST}-ranlib"  
export STRIP="${TARGET\_HOST}-strip"  
  
echo "Configuring libdivecomputer for target=$TARGET\_HOST (prefix=$STAGING\_PREFIX)"  
./configure --host="$TARGET\_HOST" --build=$(uname -m)-pc-linux-gnu --prefix="$STAGING\_PREFIX" --enable-shared --disable-static  
  
echo "Building libdivecomputer..."  
make -j"$NPROC"  
make install  
  
# 4) build dlf\_parser\_helper using cross compiler  
cd "$REPO\_ROOT/tools"  
  
# prefer to use pkg-config from cross install if present  
export PKG\_CONFIG\_LIBDIR="$STAGING\_PREFIX/lib/pkgconfig"  
export PKG\_CONFIG\_PATH="$PKG\_CONFIG\_LIBDIR:$PKG\_CONFIG\_PATH"  
  
# clean  
make clean || true  
  
echo "Building dlf\_parser\_helper.exe with $TARGET\_HOST-gcc"  
CC\_CMD="${TARGET\_HOST}-gcc"  
# Try make first (if Makefile honors CC), else compile directly  
if make CC="$CC\_CMD" -j"$NPROC"; then  
 echo "Built via Makefile"  
else  
 echo "Make failed or no target; compiling directly"  
 "$CC\_CMD" -Wall -O2 -I"$STAGING\_PREFIX/include" -L"$STAGING\_PREFIX/lib" -o dlf\_parser\_helper.exe dlf\_parser\_helper.c -ldivecomputer  
fi  
  
if [ ! -f dlf\_parser\_helper.exe ]; then  
 echo "ERROR: dlf\_parser\_helper.exe not found after build"  
 exit 1  
fi  
  
# 5) collect artifacts  
cp dlf\_parser\_helper.exe "$DIST\_DIR/"  
  
# copy libdivecomputer dll if present  
if [ -f "$STAGING\_PREFIX/bin/libdivecomputer.dll" ]; then  
 cp "$STAGING\_PREFIX/bin/libdivecomputer.dll" "$DIST\_DIR/"  
fi  
  
# 6) collect common mingw runtime dlls from sysroot location  
# Common runtime DLLs names that may be required  
RUNTIME\_DLLS=(libgcc\_s\_seh-1.dll libstdc++-6.dll libwinpthread-1.dll)  
# try candidate sysroot locations used by mingw package  
SYSROOTS=("/usr/${TARGET\_HOST}/lib" "/usr/${TARGET\_HOST}/lib64" "/usr/${TARGET\_HOST}/bin" "/usr/lib/${TARGET\_HOST}" "/usr/x86\_64-w64-mingw32/sys-root/mingw/bin")  
  
for s in "${SYSROOTS[@]}"; do  
 for dll in "${RUNTIME\_DLLS[@]}"; do  
 if [ -f "$s/$dll" ]; then  
 cp "$s/$dll" "$DIST\_DIR/" || true  
 fi  
 done  
done  
  
# 7) inspect exe for missing imports (optional, best-effort)  
if command -v "${TARGET\_HOST}-objdump" >/dev/null 2>&1; then  
 echo "Inspecting imports via ${TARGET\_HOST}-objdump"  
 ${TARGET\_HOST}-objdump -p "$DIST\_DIR/dlf\_parser\_helper.exe" | grep -i "DLL" || true  
fi  
  
# 8) finished  
echo "=== Build complete. Artifacts in: $DIST\_DIR ==="  
ls -la "$DIST\_DIR"  
  
# 9) package into a zip (optional auto-run)  
cd "$BASE\_DIR"  
ZIP\_NAME="dlf\_to\_csv-windows-$(date +%Y%m%d-%H%M%S).zip"  
( cd "$DIST\_DIR" && zip -r "../$ZIP\_NAME" . )  
echo "Created $BASE\_DIR/$ZIP\_NAME"  
  
exit 0

## File: windows/check-deps.sh

#!/usr/bin/env bash  
# check-deps.sh - verifies essential tools for cross-build  
set -euo pipefail  
  
err() { echo "ERROR: $\*" >&2; exit 2; }  
  
command -v x86\_64-w64-mingw32-gcc >/dev/null 2>&1 || err "x86\_64-w64-mingw32-gcc not found. Install mingw-w64."  
command -v git >/dev/null 2>&1 || err "git not found."  
command -v pkg-config >/dev/null 2>&1 || err "pkg-config not found."  
  
# optional helpful tools  
if command -v wine >/dev/null 2>&1; then  
 echo "Note: wine present — you can test the produced exe under Wine."  
fi  
  
x86\_64-w64-mingw32-gcc --version | head -n1  
pkg-config --version || true  
  
echo "All required tools appear present."

## File: windows/package-windows.sh

#!/usr/bin/env bash  
# package-windows.sh - create a zip of the dist/win-x86\_64 artifacts  
set -euo pipefail  
BASE\_DIR="$(cd "$(dirname "$0")" && pwd)"  
DIST\_DIR="$BASE\_DIR/dist/win-x86\_64"  
if [ ! -d "$DIST\_DIR" ]; then  
 echo "No dist directory found. Run build-cross-ubuntu.sh first." >&2; exit 1  
fi  
OUT="dlf\_to\_csv-windows-$(date +%Y%m%d-%H%M%S).zip"  
cd "$DIST\_DIR"  
zip -r "$BASE\_DIR/$OUT" .  
cd "$BASE\_DIR"  
echo "Created $OUT"

## File: windows/msys2-build-instructions.txt

MSYS2 / MinGW64 build instructions (recommended for producing Windows-native builds)  
  
1) Install MSYS2: https://www.msys2.org/ and open "MSYS2 MinGW 64-bit" shell.  
  
2) Update and install packages in MinGW64 shell:  
 pacman -Syu # may require shell restart  
 pacman -S --needed base-devel mingw-w64-x86\_64-toolchain \  
 mingw-w64-x86\_64-autoconf mingw-w64-x86\_64-automake mingw-w64-x86\_64-libtool \  
 mingw-w64-x86\_64-pkg-config mingw-w64-x86\_64-glib2 mingw-w64-x86\_64-libusb  
  
3) Clone libdivecomputer and build:  
 git clone https://github.com/libdivecomputer/libdivecomputer.git  
 cd libdivecomputer  
 autoreconf --install  
 ./configure --prefix=/mingw64 --enable-shared --disable-static  
 make -j$(nproc)  
 make install  
  
4) Build helper (from repo root inside MinGW64 shell):  
 cd /path/to/DLF\_TO\_CSV/tools  
 export PKG\_CONFIG\_PATH=/mingw64/lib/pkgconfig  
 make  
 # copy dlf\_parser\_helper.exe and required DLLs from /mingw64/bin to a dist/ folder  
  
Notes:  
 - Building inside MSYS2 ensures all Windows-targeted dependencies (glib, libusb) are available as mingw packages.  
 - Package the exe and the DLLs in the same folder for distribution.

## File: windows/README\_windows.md

# DLF\_TO\_CSV - Windows usage for dlf\_parser\_helper  
  
This folder contains scripts to cross-build `dlf\_parser\_helper.exe` (Windows) from Ubuntu, and short instructions for Windows users.  
  
## What you will get (dist folder)  
- `dlf\_parser\_helper.exe` -- the Windows executable  
- `libdivecomputer.dll` -- required library  
- possibly other DLLs (libgcc, libstdc++, libwinpthread)  
  
## Windows user quick-start  
1. Install Python 3.12+ from https://www.python.org/downloads/windows/ and check "Add Python to PATH".  
2. Copy the entire `dist/` folder to a Windows machine (e.g. `C:\dlf-tools\`).  
3. Put `dlf\_parser\_helper.exe` and `libdivecomputer.dll` in the same folder.  
4. Copy `libdc\_json\_to\_csv.py` (from repository `tools/`) to the same folder or install the repo and run:  
  
```powershell  
python libdc\_json\_to\_csv.py --helper dlf\_parser\_helper.exe --out out.csv path\to\input.dlf

## Notes

* If running the exe yields a missing DLL error, copy the listed DLL into the same folder.
* PyInstaller-built Python exe is not provided; Windows users should run the Python wrapper using their system Python.

---  
  
## File: windows/BUILD\_MANUAL.md  
  
```markdown  
# BUILD\_MANUAL: Cross-compile Windows helper from Ubuntu 24.04  
  
This document is developer-facing. It explains the exact steps the scripts perform and how to troubleshoot.  
  
## Prerequisites on Ubuntu 24.04  
- sudo privileges  
- packages: git, build-essential, mingw-w64, pkg-config, autoconf, automake, libtool, zip  
- Optional: wine for quick testing of produced exe  
  
Install these packages:  
  
```bash  
sudo apt update  
sudo apt install -y git build-essential mingw-w64 pkg-config autoconf automake libtool zip

## Steps performed by build-cross-ubuntu.sh

1. Runs windows/check-deps.sh to verify x86\_64-w64-mingw32-gcc, pkg-config, git are available.
2. Cleans previous windows/dist/ and temporary build dir at /tmp/libdivecomputer\_build.
3. Clones libdivecomputer into the build dir and runs autoreconf --install.
4. Configures libdivecomputer with:

* ./configure --host=x86\_64-w64-mingw32 --build=$(uname -m)-pc-linux-gnu --prefix=$STAGING --enable-shared --disable-static

1. Builds and make install into a staging prefix.
2. Builds dlf\_parser\_helper.exe in tools/ using cross-compiler and links against $STAGING/lib.
3. Copies dlf\_parser\_helper.exe and libdivecomputer.dll (if produced) into windows/dist/win-x86\_64.
4. Attempts to copy common mingw runtime DLLs if found in known sysroot locations.
5. Creates a timestamped zip of the dist contents.

## Troubleshooting tips

* **configure fails**: inspect config.log for missing headers/libraries. On cross-build you may need to provide cross-compiled dependencies (glib, libusb). Usually using MSYS2/MinGW on Windows is easier.
* **undefined references at link time**: ensure -L points to the staging prefix lib dir and that the import library libdivecomputer.dll.a exists.
* **missing runtime DLLs on Windows**: use ${TARGET\_HOST}-objdump -p dlf\_parser\_helper.exe to list needed DLLs and copy them from the cross sysroot or MSYS2 /mingw64/bin.
* **test via Wine**: wine dlf\_parser\_helper.exe tests/00000002.dlf > out.json will show dynamic loader errors for missing DLLs.

## Where to find logs

* The script echoes commands and exits on error. For deeper logs, run with bash -x windows/build-cross-ubuntu.sh.

---  
  
## File: windows/.gitignore (snippet)  
  
```text  
# ignore distribution artifacts  
/dist/  
  
# local build temp used by scripts  
/tmp/libdivecomputer\_build

## File: windows/.github-actions-windows-build.yml (example workflow to add to repo/.github/workflows)

name: Build Windows helper (cross-compile)  
  
on:  
 push:  
 branches: [ mail ]  
 workflow\_dispatch:  
  
jobs:  
 build-win-cross:  
 runs-on: ubuntu-24.04  
 steps:  
 - uses: actions/checkout@v4  
  
 - name: Install cross toolchain and deps  
 run: |  
 sudo apt update  
 sudo apt install -y git build-essential autoconf automake libtool pkg-config g++-mingw-w64-x86-64 gcc-mingw-w64-x86-64 mingw-w64 zip  
  
 - name: Build libdivecomputer (cross)  
 run: |  
 WORK=/tmp/libdc\_ci  
 rm -rf $WORK  
 git clone --depth 1 https://github.com/libdivecomputer/libdivecomputer.git $WORK/libdivecomputer  
 cd $WORK/libdivecomputer  
 autoreconf --install || true  
 ./configure --host=x86\_64-w64-mingw32 --prefix=$WORK/libdc-win --enable-shared --disable-static \  
 CC=x86\_64-w64-mingw32-gcc AR=x86\_64-w64-mingw32-ar RANLIB=x86\_64-w64-mingw32-ranlib  
 make -j$(nproc)  
 make install  
  
 - name: Build helper  
 run: |  
 export PKG\_CONFIG\_LIBDIR=$WORK/libdc-win/lib/pkgconfig  
 cd tools  
 make CC=x86\_64-w64-mingw32-gcc AR=x86\_64-w64-mingw32-ar RANLIB=x86\_64-w64-mingw32-ranlib || true  
 mkdir -p windows/dist/win-x86\_64  
 if [ -f tools/dlf\_parser\_helper ]; then cp tools/dlf\_parser\_helper windows/dist/win-x86\_64/dlf\_parser\_helper.exe; fi  
 if [ -f dlf\_parser\_helper.exe ]; then cp dlf\_parser\_helper.exe windows/dist/win-x86\_64/dlf\_parser\_helper.exe; fi  
  
 - name: Upload artifact  
 uses: actions/upload-artifact@v4  
 with:  
 name: dlf\_to\_csv-windows  
 path: windows/dist

## Commit message suggestion

Add Windows cross-build scripts & CI

# How to use this canvas

Open the canvas and copy each file to windows/ in your repository. After saving the files locally, run:

# from repo root  
git checkout -b mail  
git add windows  
git commit -m "Add Windows cross-build scripts & CI"  
git push origin mail

If you want, I can also produce a single one-liner that writes these files into your repo for you. Request that and I will provide it.

If you want any file tweaked (different prefix, static vs shared default, add more DLL collection paths), tell me and I will update the canvas file accordingly.