Benni-cmake-v6

Dies ist eine reine C/C++ Toolchain für Linux und Windows, 64-bit auf Basis von cmake das einen Dependency Graphen auflösen kann. Es werden nur reine cmake Befehle benutzt und nur eine Funktion zum schreiben von allen wichtigen cmake und benni Variablen auf die Konsole.

This is a pure C/C++ toolchain for Linux and Windows, 64-bit based on cmake, that can solve/decompose a dependency graph. It uses only cmake built in functions, only one custom function that writes important cmake and benni variables to the console.

Wir mögen keine eigenen Funktionen, wie tsd.buildenv, wenn cmake schon das Tool für Alles ist. Die Existenz von tsd.buildenv ist eher ein Hinweis darauf dass jemand cmake nicht verstanden hat.

We dont like custom functions like tsd.buildenv when cmake is already king.
This is more of a hint that somebody did not understand what cmake is about and already should do as-is.

Wir mögen auch keinen Microsoft Compiler da dies der Ursprung von viel Elend und Leid ist, inklusive Microsoft Headern die zueinander inkompatibel sind, waren und mir mittlerweile total egal sind.

Wir mögen auch keine doppelten Microsoft Runtimes MD, MDd, MT, MTd, die den Projektpflegeaufwand für JEDEN einfach mal verdoppelt. Mit dem MinGW Compiler gibt es EINE Runtime, und nicht mehr. Für den Hobbyprogrammierer vereinfacht das wirklich ungemein das eigene Leben. Wenn du es mir nicht glaubst, schön für dich, ist ja dein langweiliges Leben dass du überbrücken möchtest.

START:

Grundsätzlich benötigen wir einen LinuxTerminal (Emulator auf Windows), einen GCC Compiler und meist noch die Qt Bibliotheken für schicke Apps.

Nötige Tools auf Windows x64:
1. MSYS 2 oder das gute alte MSYS 1.11.0
(LinuxTerminalEmulatoren)
2. QtSDK 64-bit mit integriertem, aktuellem MinGW x64 Compiler >= 7.3.0

D.h. wir haben min. C++17 Standard aktiviert beim Bauen

3. CMake herunterladen und installieren, nicht in den Pfad eintragen wenn es nicht sein muss, wir machen das händisch im .bat Skript
4. .bat Skript starten - es verbindet alle wichtigen Pfade (gcc, g++, cmake) und startet den TerminalEmulator msys2.

ARBEITEN im LinuxTerminal(Emulator):

1. In dem Projektpfad navigieren `cd Documents/benni-cmake-v6`
2. Hauptskripte benutzen `./clean.sh` und `./build_win.sh`

Nötige Tools auf Linux x64:

1. sudo apt-get install cmake qt5base-dev g++ gcc libxxf86vm1-dev
2. Mein .sh

Um auf Windows sofort starten zu können benötigen wir also einen LinuxTerminalEmulator wie - seit eh und je - msys1.11.0

Oder den neueren Terminal msys2, den ich mittlerweile selber verwende und in meinem StartSkript aufrufe.

Der Terminal Emulator muss im NormalFall Linux Befehle wie cd, cp, make, gcc, g++, bash und wohl viele andere verstehen damit C und C++ Rezepte funktionieren können.

Aber tatsächlich benutze ich weder configure noch sonst ein externes Skript das nicht cmake ist. Von daher ist es wohl möglich auch mit Visual Studio Compiler zu versuchen, aber warum.

MSYS2 ist macht einiges deutlich besser als die 20 Jahre alte cmd.exe von Microsoft

2. Wir benötigen das Qt SDK x64 für Windows mit integriertem MinGW Compiler x64 oder auf Linux einfach die normale 64-Bit Version des QtSDKs
Bezugsaddresse: www.qt.io.

Mit integriertem MinGW 7.3 Compiler x64, und schon können wir vollständige Linux App auf Windows bauen, unter Nutzung von Linux und Windows API, wie man es braucht. Der riesen Vorteil ist die im Gesamten gleiche CodeBasis, gleiche Bibliotheken soweit geht, und wenig Portierungsstress da im MinGW Compiler tatsächlich alle Microsoft Windows Header von irgend jemandem Open Source nachgebaut wurde.