Trinkets

It's the little things that matter

The word "trinket" means "ornament or small item of little importance". That exactly describes my collection of digital trinkets: you'll probably either use them once and delete them afterwards, or you might set them somewhere and forget about it while they trudge along, doing what they do, forever.

What's included

- pisrv: a script that starts a Visual Studio Code tunnel as soon as my Raspberry Pi boots up, making it 99% power failure resilient. This trinket is so simple that I have decided it would be a waste of time to create a planning document for it.
- buildgcc: An extremely user-friendly and scriptable GCC installer that takes a Windows Installer-type approach to make fine-tuning your GCC builds a whole lot easier.
- c-project-config: this application also takes a Windows Installer-type approach to help you code; it can set up and configure a C/C++ (one or both) programming environment for you so you can get to coding without having to set up debugging configurations, makefiles, and dependencies from scratch.

The creation of this document involved several steps:

1. It was originally created on a reMarkable 2 tablet. Most of the editing work happened here, in mostly black-and-white other than colored markers and highlighting.

2. Then, it was exported to a PDF from the tablet and that PDF was imported into GIMP.

3. Then I "dark-modeified" the entire document in GIMP and changed the colors of some things like drawings etc.

4. Then I exported it to a PNG, which you're probably reading right now! Or you may be reading the HTML version, which is a manually created file that attempts to replicate the PNG in HTML format.

buildgcc

An extremely user-friendly and scriptable GCC installer that takes a Windows Installer-type approach to make fine-tuning your GCC builds a whole lot easier.

- Licensed under the MIT License
- Written in C++23 with GNU extentions (may support older standards, but not guaranteed)
- Has two dependencies: Boost 1.81.0 and ncurses 6.4. These will be built automatically the first time this program is built by the

Section 1: Program Flow

The program's flow is illustrated with a "text-based flowchart", which is basically a list of steps that goes from top to bottom. Yellow highlighting means "jump to step #"; pink means "a program entry/exit point"; and green means "wait for user action". If a step gives no instruction on which step to go to next, go to the next step that's directly below it, regardless of its indentation.

0. The program starts.

1. The program parses command-line arguments, the configuration file, and environment variables. (Done with Boost's program_options library)

 1.1. If there are no arguments, go to step 2. If there are arguments, go to step 1.2.

 1.2. If the program has been passed some arguments:

  1.2.1. Check if the passed arguments are valid. If yes, go to step 1.2.2. If not, exit with code -1.

  1.2.2. If the program has been told to automate compilation and all arguments are correct, go to step 1.2.2.1. If the program has been given arguments but don't tell it to automate compilation, go to step 1.2.3.

   1.2.2.1. Check if all dependencies are installed. If yes, skip to step 1.2.2.2. If not:

    1.2.2.1.1. Check if the program is allowed to try to automatically install dependencies. If not, exit with code 1.

    1.2.2.1.2. Attempt to install dependencies. If it has failed, exit with code 2. If the correct versions of all dependencies have been installed, go to step 1.2.2.2.

   1.2.2.2. Check if all GCC dependencies are present. If yes, skip to step 1.2.2.3. If not:

    1.2.2.2.1. Check if the program is allowed to automatically download GCC dependencies. If not, exit with code 3.

    1.2.2.2.2. Attempt to download and extract dependencies. If it has failed, exit with code 4. If the correct versions of all GCC dependencies have been installed, go to step 1.2.2.3.

   1.2.2.3. Start the build.

2. Usermode

3. Automated

About

Created on 3/10/2023 by Akshat Singh (HackerDaGreat57).