AUTOMATIC BUILD VERSION NUMBERING

FUN WITH VERSION NUMBERS!

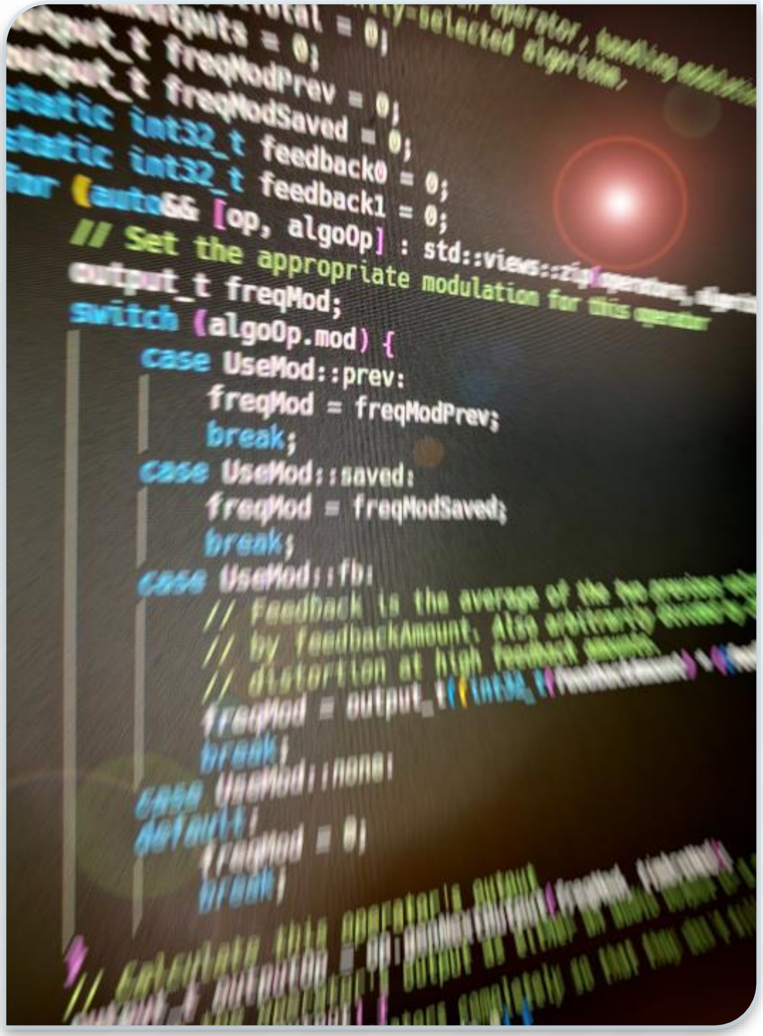
ABOUT ME

Len Popp

Retired Software Guy,

Software & Hardware Hobbyist

https://lenp.net/



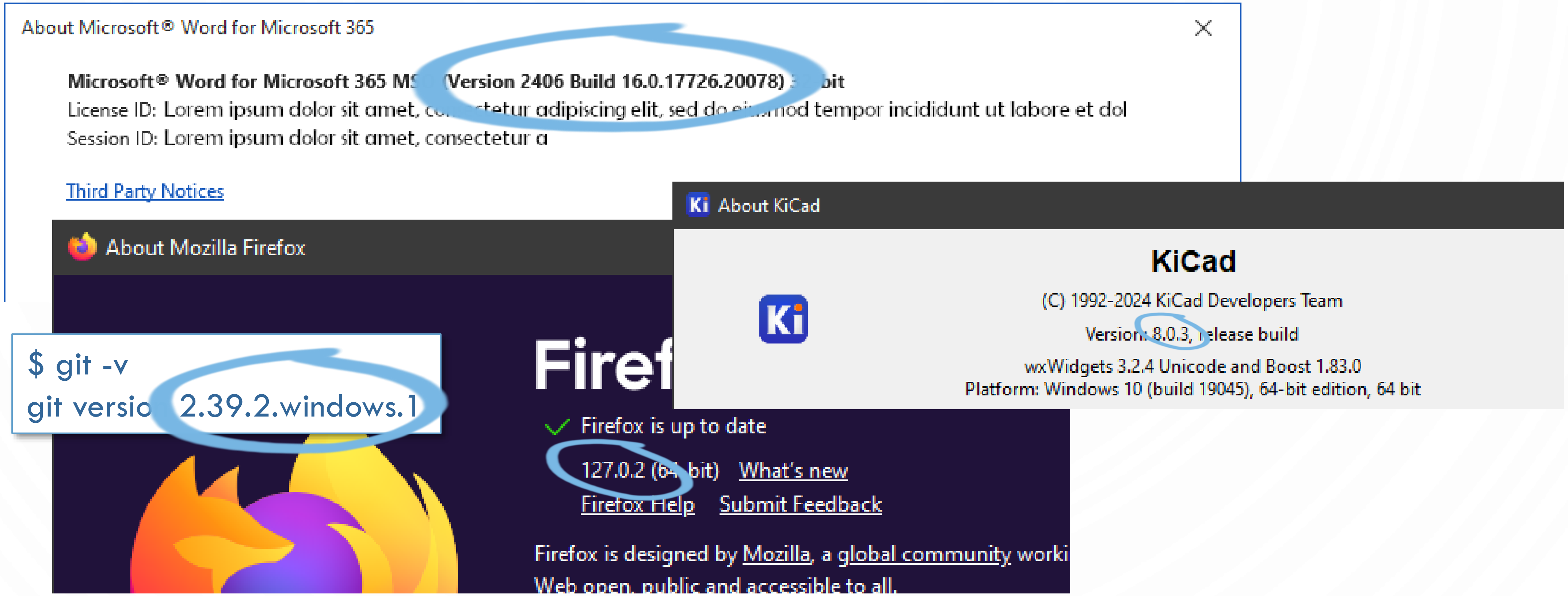
```
[illegible]utputs = 0;
[illegible]t_t freqModPrev = 0;
output_t freqModSaved = 0;
static int32_t feedback0 = 0;
static int32_t feedback1 = 0;
for (auto&& [op, algoOp] : std::views::zip([illegible]
    // Set the appropriate modulation for this operator
    output_t freqMod;
    switch (algoOp.mod) {
        case UseMod::prev:
            freqMod = freqModPrev;
            break;
        case UseMod::saved:
            freqMod = freqModSaved;
            break;
        case UseMod::fb:
            // Feedback is the average of the [illegible]
            // by feedbackAmount. Also [illegible]
            // distortion at high feedback [illegible]
            freqMod = output_t((int32_t([illegible]
            break;
        case UseMod::none:
        default:
            freqMod = 0;
            break;
```

AGENDA

- About Version Numbers
- What I Want
- How I Do It
- Questions?

ABOUT VERSION NUMBERS

EXAMPLES



About Microsoft® Word for Microsoft 365

Microsoft® Word for Microsoft 365 MS (Version 2406 Build 16.0.17726.20078) bit
License ID: Lorem ipsum dolor sit amet, co tetur adipiscing elit, sed do ei mod tempor incididunt ut labore et dol
Session ID: Lorem ipsum dolor sit amet, consectetur a

Third Party Notices

About Mozilla Firefox

Firef

Firefox is up to date
127.0.2 (6 bit) What's new
Firefox Help Submit Feedback

Firefox is designed by Mozilla, a global community worki
Web open, public and accessible to all.

```
$ git -v
git versio 2.39.2.windows.1
```

About KiCad

KiCad
(C) 1992-2024 KiCad Developers Team
Versio 8.0.3, elease build
wxWidgets 3.2.4 Unicode and Boost 1.83.0
Platform: Windows 10 (build 19045), 64-bit edition, 64 bit

WHY VERSION NUMBERS?

- Know which version of software you’re running
- Dependencies
- Bug reporting/debugging
    - Connect an executable to a particular version of the source code

WHERE VERSION NUMBERS ARE USED

- In the code, for display ("About…", `git -v`)
- EXE file properties
- Text files (README)
- Installer (Windows Control Panel)
- Documentation (WinHelp, Doxygen)
- Version control system (`git tags`, GitHub releases)
- *These should all match!*

THE EASY WAY

version.h

```
// Update this whenever a new release is built
constexpr unsigned verMajor = 1;
constexpr unsigned verMinor = 0;
constexpr unsigned verRevision = 5;
constexpr char verString[] = "1.0.5";
```

THE END

WHY IS THIS PRESENTATION NOT OVER?

- Can’t do everything with version.h
  - Multiple languages (e.g. C++ and C#), text files, installer, help files, git tags, etc.
  - Version info must be duplicated in multiple places
- Every distinguishable build should have a distinct version number, automatically
  - That includes unreleased builds during development and testing, and automated builds
  - That’s a pain!
- There must be an easy way

MY SPECIFIC MOTIVATION

- Problems encountered building releases of commercial software
- Version numbers were a pain to maintain properly – error-prone
- Lesson learned: *Every build that is seen by two people needs a version number*
- Needed a correct, streamlined build process
- Similar issues in my open-source hobby projects

WHAT I WANT

MY REQUIREMENTS FOR VERSION NUMBERS

REQUIREMENTS

- The version number is specified in one single place, simply & easily
- It can be used wherever it's needed
- Consistent everywhere
- Automatically append a build number
  - For intermediate "releases" that don't have an explicit version number
  - Auto numbering must be in increasing order
- Efficient to build

MY VERSION NUMBER SCHEMA

- Format: `[major].[minor].[revision].[build]-[stuff]`
- Example: `1.2.3.4567-stuff`
  - “stuff” will be described later
- Windows-compatible (4 numeric components)
- “Inspired by” *semantic versioning* but not quite the same
- `major`, `minor`, `revision` are set explicitly; `build` is generated automatically
- Caveat: This is just how I do it

HOW I DO IT

MY STREAMLINED PROCESS FOR SETTING VERSION NUMBERS

MY DEVELOPMENT ENVIRONMENT

- C++, C#
- Microsoft Visual Studio
- Git
- Various other tools for software & hardware development
- For other projects I use VS Code with make or CMake (with a few changes)
- It could easily work with other build systems

VERSION NUMBER IS SET BY A GIT TAG

- This is the “one place” where I set the version number
- To increment the version number for a new release, set a git tag
- Examples: `1, 1.2, 1.2.3`
  - Minor and revision numbers are optional; build number is omitted
- This connects an executable precisely to its source code
- Again, this is just how I do it.
  - Version number could be defined in a file, for example.

```
$ git tag
0
0.1
0.2
1.0
1.1
1.2
1.3
1.4
1.5
1.5.1
```

MAKEVERSIONINFO PROJECT

- Add this Visual Studio project to the solution
- Main C++ project depends on `MakeVersionInfo`
- `MakeVersionInfo` runs every time you do a build:
  - Gets the version number from git
  - Updates the version number references in any specified files
  - But only if the version number has changed
- Files are only recompiled if the version info has changed since the last build

WHICH FILES?

- Need a list of the files that will contain the up-to-date version number
- `init-version-info.bat` in the Visual Studio solution directory

```
:: Project settings for MakeVersionInfo
set SOLUTIONDIR=%~dp0
set TARGETS="%SolutionDir%version.h" "%SolutionDir%README.txt"
```

TEMPLATE FILES

- Each file listed in `init-version-info.bat` is created from a template file
- Example: `version.h` is defined by `version.htemplate`
- The template contains substitution items where the version numbers are to appear
- There's other info too, e.g. build date
- "`{`" characters are represented by "`{{`"
  - Unfortunate compromise

```
namespace Version
{{
    constexpr unsigned major = {verMajor};
    constexpr unsigned minor = {verMinor};
    constexpr unsigned revision = {verRevision};
    constexpr unsigned build = {verBuild};
    constexpr char commit[] = "{verCommit}";
    constexpr bool isDevBuild = {verIsDevBuild};
    constexpr char name[] = "{verString}";
    constexpr char date[] = "{verDatestamp}";
    constexpr char time[] = "{verTimestamp}";
}}
```

HOW IT WORKS

- `MakeVersionInfo` is a Visual Studio “Makefile” project (`nmake`)
  - Calls `init-version-info.bat` to get the list of files to be processed
  - Calls `git describe` to get version info from a recent tag
  - Calls a Python script to update `version-info` file, only if the version number changed
  - `nmake` rebuilds output files as required (using `Makefile` and another Python script)
- If any files were written, build targets depending on them will be re-built

AUTOMATIC BUILD NUMBER

- From `git describe`

```
$ git describe --tags --always --dirty
3.1-5-g8f5bc0e-dirty
```

tag build commit modified

- Build number is the number of commits past the most recent tag
- Resulting version string:

```
3.1.0.5-g8f5bc0e-dev
```

- The extra stuff disambiguates the git branch and marks in-development code

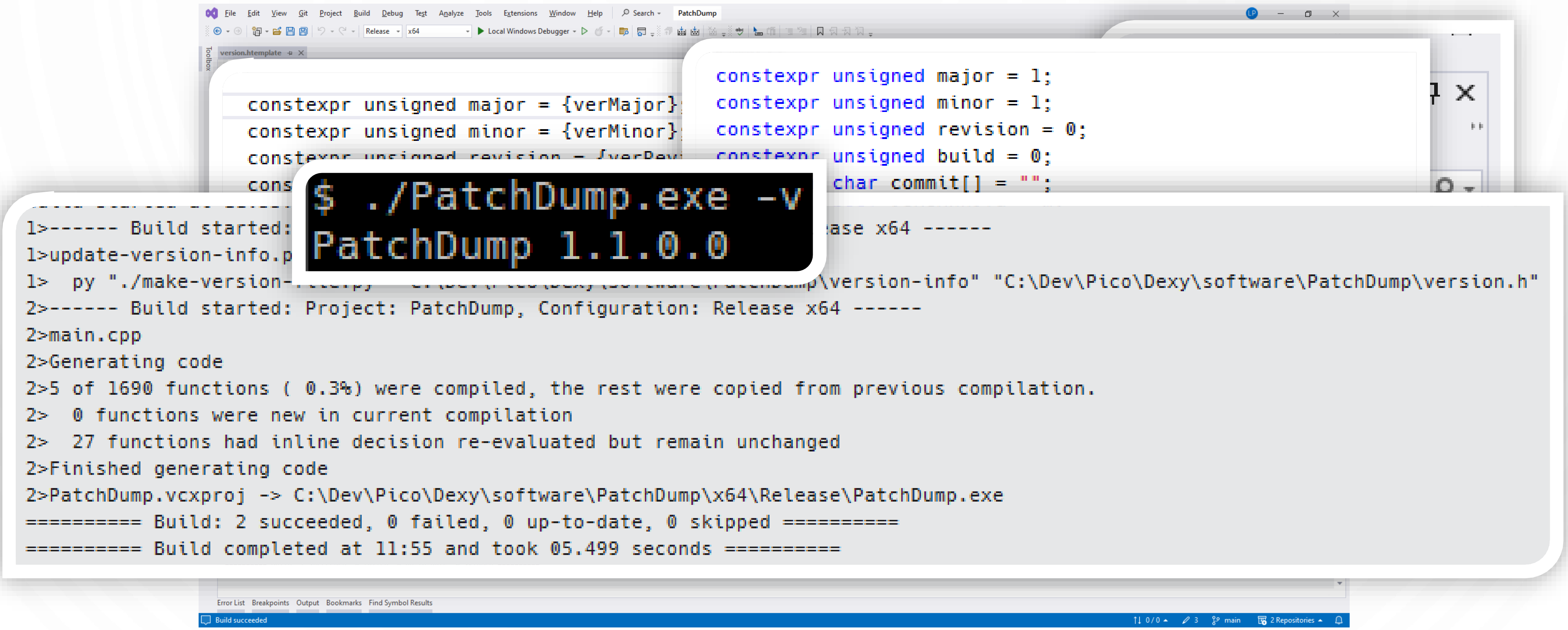
HOW-TO SUMMARY

- Add `MakeVersionInfo` to the Visual Studio solution
- Other projects that use version info depend on `MakeVersionInfo`
- Copy & modify `init-version-info.bat` to specify version-specific files
- Make template files for all those files
- `git tag 0.1`
- Build the solution

EFFICIENCY

- The `MakeVersionInfo` build step runs on every build
- When version number is unchanged, takes very little time
  - Python is faster than PowerShell
- Minimal rebuild
  - No unnecessary recompilation when version number is unchanged

EXAMPLE


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search
PatchDump
Release
x64
Local Windows Debugger
version.htemplate
Error List Breakpoints Output Bookmarks Find Symbol Results
Build succeeded
main
2 Repositories

```
constexpr unsigned major = {verMajor}
constexpr unsigned minor = {verMinor}
constexpr unsigned revision = {verRev
cons
```

```
constexpr unsigned major = 1;
constexpr unsigned minor = 1;
constexpr unsigned revision = 0;
constexpr unsigned build = 0;
          char commit[] = "";
```

```
$ ./PatchDump.exe -v
PatchDump 1.1.0.0
```

```
1>------ Build started:                                   ase x64 ------
1>update-version-info.p
1>  py "./make-version-                                   \version-info" "C:\Dev\Pico\Dexy\software\PatchDump\version.h"
2>------ Build started: Project: PatchDump, Configuration: Release x64 ------
2>main.cpp
2>Generating code
2>5 of 1690 functions ( 0.3%) were compiled, the rest were copied from previous compilation.
2>  0 functions were new in current compilation
2>  27 functions had inline decision re-evaluated but remain unchanged
2>Finished generating code
2>PatchDump.vcxproj -> C:\Dev\Pico\Dexy\software\PatchDump\x64\Release\PatchDump.exe
========== Build: 2 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Build completed at 11:55 and took 05.499 seconds ==========
```

SOURCE CODE

- Visual Studio implementation: https://github.com/Len42/MakeVersionInfoP
- CMake implementation: https://github.com/Len42/Dexy/tree/main/firmware
- Makefile implementation: https://github.com/Len42/dat-ting/tree/main/lib/MakeVersionInfo
- Or just lenp.net

\0

QUESTIONS?