Elizabeth Baker – lizbaker@umich.edu
Zain Azeeza – zainazee@umich.edu

Collaborative Governance in the MuseScore Open-Source Workflow

Project Report

Selected Project

MuseScore is a free and open-source music notation software that enables users to create, play back, and print sheet music with a user-friendly WYSIWYG interface. Developed primarily in C++ using the Qt framework, MuseScore offers a comprehensive set of features, including support for unlimited staves, up to four voices per staff, input via MIDI keyboard, and a wide range of musical symbols and notations. It supports import and export in various formats such as MusicXML, MIDI, PDF, and audio files like WAV and MP3. MuseScore is cross-platform, running on Windows, macOS, and Linux, and is available in over 40 languages. The project is maintained by the Muse Group and has an active community of contributors on GitHub. MuseScore's commitment to accessibility and community-driven development makes it a valuable tool for musicians, educators, and composers worldwide.

Social Good Indication

Yes, we believe MuseScore contributes to Social Good, aligning with the United Nations Sustainable Development Goal 4: Quality Education. As a free and open-source music notation software, MuseScore democratizes access to music education and composition tools, enabling individuals worldwide to learn, create, and share music without financial barriers. Its commitment to accessibility is further exemplified by initiatives like the Open Goldberg Variations and Open Well-Tempered Clavier projects, which produced high-quality, freely available scores and recordings, including braille editions for visually impaired musicians.

Project Context

MuseScore began in 2002 as a personal project by developer Werner Schweer, who branched off the notation component from the MusE sequencer to build a standalone score-writing tool. Initially Linux-only, it expanded to Windows by 2007 and macOS by 2009. The launch of MuseScore.org in 2008 helped foster a growing community, with downloads rising from 15,000 to 80,000 per month within two years. MuseScore 1.0 was released in 2011 as the first stable version, and subsequent updates (MuseScore 2 in 2015, MuseScore 3 in 2018, and MuseScore

4 in 2022) brought significant enhancements, including tablature support, automatic layout, and a new engraving engine.

From a hobbyist endeavor, MuseScore evolved into one of the most widely used music notation platforms, with tens of millions of downloads worldwide. MuseScore’s open-source GPL license and zero-cost model are central to its mission of broad accessibility. The project explicitly commits to remaining free and open-source, distinguishing itself from expensive commercial alternatives. This accessibility has made it a key resource in music education, especially in underfunded schools and for amateur composers. Educational institutions around the world—including school districts installing it across thousands of devices—rely on MuseScore for notation, playback, and sharing.

Its intuitive interface, rich feature set, and no-cost barrier have democratized access to professional-grade music notation tools. In the broader music software ecosystem, MuseScore competes with proprietary programs like Finale, Sibelius, and Dorico, and with web-based tools like Noteflight. Unlike these paid offerings, MuseScore is available across all major operating systems and is developed collaboratively by a global community. It offers nearly all core features required by composers and arrangers: multi-stave layouts, chord symbols, lyrics, percussion notation, and MusicXML/MIDI import/export.

Although proprietary tools once led in engraving precision or bundled sound libraries, MuseScore has rapidly closed the gap through continual updates. Today, it is widely regarded as fulfilling the vast majority of user needs, making it a powerful, inclusive, and globally adopted alternative in the world of music notation software

Project Governance

MuseScore is developed and maintained as a collaborative open-source project hosted on GitHub, where contributions are coordinated using a Git-based workflow. The governance of the project is semi-formal—clear contribution guidelines, tooling standards, and communication norms exist, but without a rigid top-down structure. Contributors are expected to follow a suggested Git workflow that enables distributed development while maintaining codebase integrity.

The process begins by forking the MuseScore repository to a contributor's own GitHub account. Developers clone this fork locally, create a feature branch based on the issue they want to address, and commit their changes with meaningful commit messages. Contributors are strongly encouraged to reference issue numbers in their commit messages (e.g., Fix #12345: Description) to automatically link changes to issue tracker entries.

To ensure alignment with the main repository, contributors must regularly rebase their branches on the upstream master and resolve any merge conflicts. Once the feature or bug fix is complete and rebased, contributors push the branch to their fork and initiate a Pull Request

(PR) via GitHub’s interface. After submission, PRs are reviewed by maintainers or experienced community members, who may request changes, suggest improvements, or approve the contribution. If the contributor needs to update the PR, they may push additional commits or perform a forced push if they amend or squash commits.

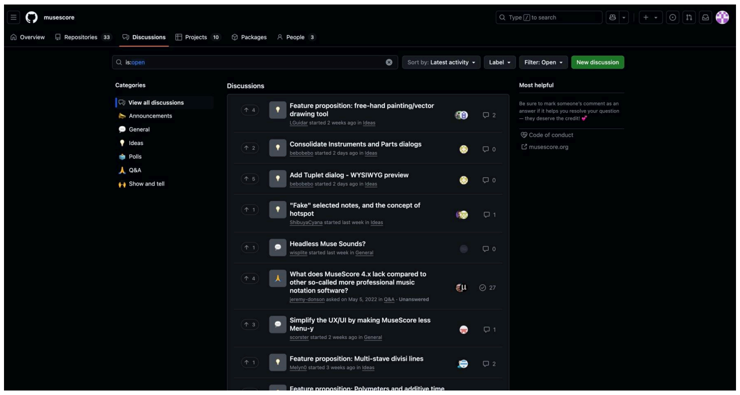
Acceptance Process Overview

1. Fork → Clone → Branch Creation: Developers fork MuseScore, clone the repo, and create a topical branch (e.g., 404-new-feature).
2. Code Implementation: Developers write and test their code locally using the prescribed build process and architecture documentation.
3. Rebase and Push: Code is kept up to date with the upstream repository using git fetch and rebase. Once ready, it’s pushed to the contributor's GitHub fork.
4. Pull Request: A PR is opened to MuseScore’s main repo. This triggers CI workflows and static analysis (e.g., code formatting checks via Uncrustify).
5. Review & Feedback: A maintainer reviews the PR. The contributor may revise the code based on feedback.
6. Merge: After approval and passing all CI checks, the PR is merged. The contributor’s changes are now part of the main MuseScore codebase.

The project enforces a specific code style, especially for C++, which is based on Qt standards but includes custom rules (e.g., 4-space indentation, mu:: namespacing, camelCase for variables, and brace alignment). Style conformance is checked during PRs via automated CI tools. Violations block the merge until resolved. Contributors are also expected to sign the MuseScore Contributor License Agreement (CLA) to have their code accepted.

For quality assurance, developers are encouraged to provide accompanying tests, which are placed in the mtest directory. The build process includes platform-specific pipelines (e.g., GitHub Actions for Linux/macOS and AppVeyor for Windows), and artifacts can be generated and tested before merge. Code reviews, static analysis, and regression testing are common QA practices.

Communication among developers happens mainly through GitHub issues and pull request comments, but real-time discussions also take place via MuseScore’s official Discord server (e.g., #support and #development channels). Developers can also coordinate via the MuseScore forums.



Discussions

4 Feature proposition: free-hand painting/vector drawing tool
LGuidar started 2 weeks ago in Ideas 2

2 Consolidate Instruments and Parts dialogs
bebobebo started 2 days ago in Ideas 0

5 Add Tuplet dialog - WYSIWYG preview
bebobebo started 2 days ago in Ideas 0

1 "Fake" selected notes, and the concept of hotspot
ShibuyaCyana started last week in Ideas 1

1 Headless Muse Sounds?
wisplite started last week in General 0

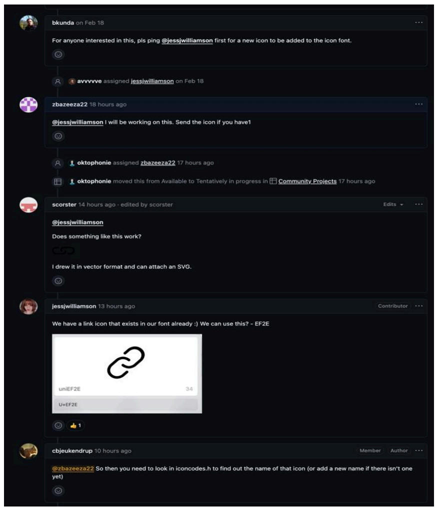
4 What does MuseScore 4.x lack compared to other so-called more professional music notation software?
jeremy-donson asked on May 5, 2022 in Q&A · Unanswered 27

3 Simplify the UX/UI by making MuseScore less Menu-y
scorster started 2 weeks ago in General 1

1 Feature proposition: Multi-stave divisi lines
Melyn0 started 3 weeks ago in Ideas 2

Feature proposition: Polymeters and additive time

Be sure to mark someone's comment as an answer if it helps you resolve your question — they deserve the credit! 💕



bkunda on Feb 18

For anyone interested in this, pls ping @jessjwilliamson first for a new icon to be added to the icon font.

avvvvve assigned jessjwilliamson on Feb 18

zbazeeza22 18 hours ago

@jessjwilliamson I will be working on this. Send the icon if you have1

oktophonie assigned zbazeeza22 17 hours ago

oktophonie moved this from Available to Tentatively in progress in Community Projects 17 hours ago

scorster 14 hours ago · edited by scorster

@jessjwilliamson

Does something like this work?

I drew it in vector format and can attach an SVG.

jessjwilliamson 13 hours ago

We have a link icon that exists in our font already :) We can use this? - EF2E

cbjeukendrup 10 hours ago

@zbazeeza22 So then you need to look in iconcodes.h to find out the name of that icon (or add a new name if there isn't one yet)

Task Descriptions

We decided to switch projects because we were unable to get the original project from Part A to fully build and run beyond the initial test build. Despite following the setup instructions and attempting various fixes, unresolved compatibility issues and missing dependencies prevented us from progressing past the test phase. As a result, we were unable to meaningfully contribute to the codebase or implement the planned changes. Below is a description of the new tasks we worked on in our new project. Since we are completing new tasks from Part A, we included some parts from part A to give more context on the task.

Task 1

The task was to replace the textual label "[LINK]" in MuseScore's Layout panel for linked staves with a visual icon to improve clarity and enhance user experience by making the interface cleaner and more intuitive. This involved locating the code responsible for rendering the "[LINK]" label, designing or reusing an appropriate icon from MuseScore's existing resource set, and integrating it using Qt's graphics framework. Additionally, we implemented a tooltip using Qt's setToolTip() function to dynamically display the name or identifier of the linked staff when hovered over, providing immediate contextual information without clutter. While the icon replacement was the primary goal, the tooltip served as a valuable accessibility enhancement, especially useful in complex scores.

Task Link [#26532 – Linked staves representation in Layout panel: Use icon instead of [LINK]]

Functional Requirements

- Replace "[LINK]" text in the Layout panel with an icon that visually indicates a staff is linked.
- Tooltip should appear on hover, displaying the name or identifier of the linked staff.

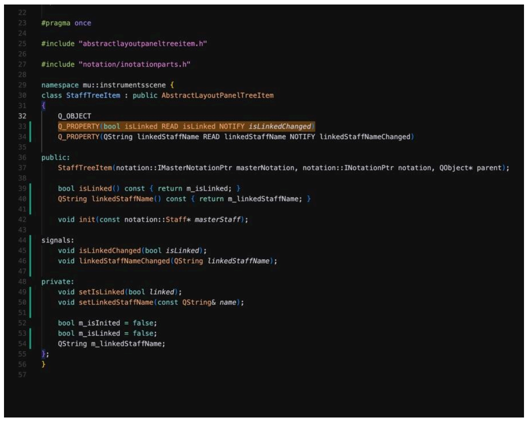
- Tooltip content must be dynamically generated based on which staff is being viewed.

Quality Requirements

- Icons must be legible, non-intrusive, and consistent with MuseScore's design system.
- Tooltip must appear promptly and contain accurate, concise information.
- The change must not affect the underlying linking logic or layout behavior.
- All changes must pass existing tests and not break build compatibility across platforms.

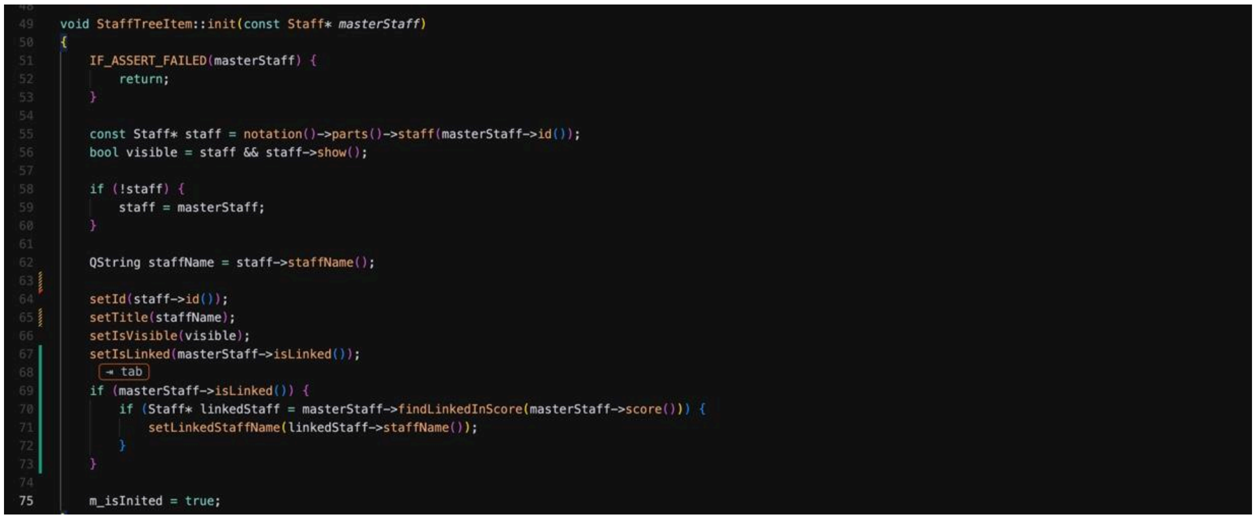
Task 1 Artifacts

Added Properties to `StaffTreeItem` Class (`stafftreeitem.h`):
We introduced two new `Q_PROPERTY` declarations, `isLinked` and `linkedStaffName`, to the `StaffTreeItem` class. These properties expose whether a staff is linked and the name of the linked staff directly to QML, allowing the user interface to react dynamically to linking information. To support these new properties, corresponding member variables, getter and setter methods, and notification signals were also added. This change improves modularity by separating UI logic from backend data while enabling more flexible and maintainable QML designs.



```
#pragma once

#include "abstractlayoutpaneltreeitem.h"

#include "notation/inotationparts.h"

namespace mu::instrumentsscene {
class StaffTreeItem : public AbstractLayoutPanelTreeItem
{
    Q_OBJECT
    Q_PROPERTY(bool isLinked READ isLinked NOTIFY isLinkedChanged)
    Q_PROPERTY(QString linkedStaffName READ linkedStaffName NOTIFY linkedStaffNameChanged)

public:
    StaffTreeItem(notation::IMasterNotationPtr masterNotation, notation::INotationPtr notation, QObject* parent);

    bool isLinked() const { return m_isLinked; }
    QString linkedStaffName() const { return m_linkedStaffName; }

    void init(const notation::Staff* masterStaff);

signals:
    void isLinkedChanged(bool isLinked);
    void linkedStaffNameChanged(QString linkedStaffName);

private:
    void setIsLinked(bool linked);
    void setLinkedStaffName(const QString& name);

    bool m_isInited = false;
    bool m_isLinked = false;
    QString m_linkedStaffName;
};
}
```

Modified Staff Initialization (`stafftreeitem.cpp`):
The `init` method in `StaffTreeItem` was updated to set the new `isLinked` and `linkedStaffName` properties during staff initialization. Previously, linked staff were indicated by manually prefixing the staff title with "[LINK]", which was removed for a cleaner approach. Now, if a staff is linked, the object automatically retrieves the associated linked staff name using the existing `findLinkedInScore()` method and sets it appropriately. This refactor makes the code more structured, eliminates hardcoded text modifications, and lays the groundwork for a more sophisticated and dynamic UI.



```cpp
void StaffTreeItem::init(const Staff* masterStaff)
{
    IF_ASSERT_FAILED(masterStaff) {
        return;
    }

    const Staff* staff = notation()->parts()->staff(masterStaff->id());
    bool visible = staff && staff->show();

    if (!staff) {
        staff = masterStaff;
    }

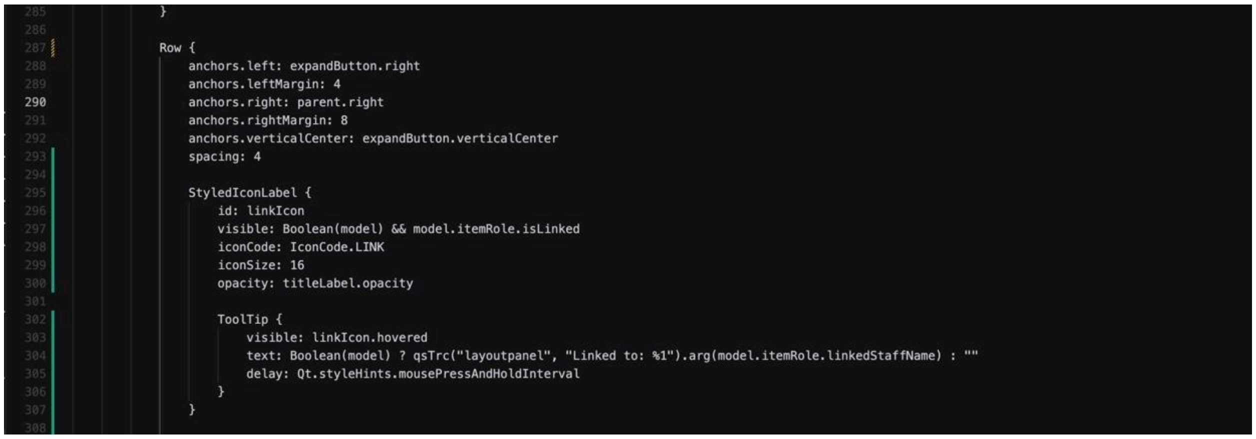
    QString staffName = staff->staffName();

    setId(staff->id());
    setTitle(staffName);
    setIsVisible(visible);
    setIsLinked(masterStaff->isLinked());

    if (masterStaff->isLinked()) {
        if (Staff* linkedStaff = masterStaff->findLinkedInScore(masterStaff->score())) {
            setLinkedStaffName(linkedStaff->staffName());
        }
    }

    m_isInited = true;
```

Updated QML Layout (`LayoutPanelItemDelegate.qml`):
The QML layout was updated to replace the old text-based "[LINK]" label with a styled link icon that appears only when a staff is linked. A tooltip was also added to the icon, revealing the name of the linked staff when hovered over.

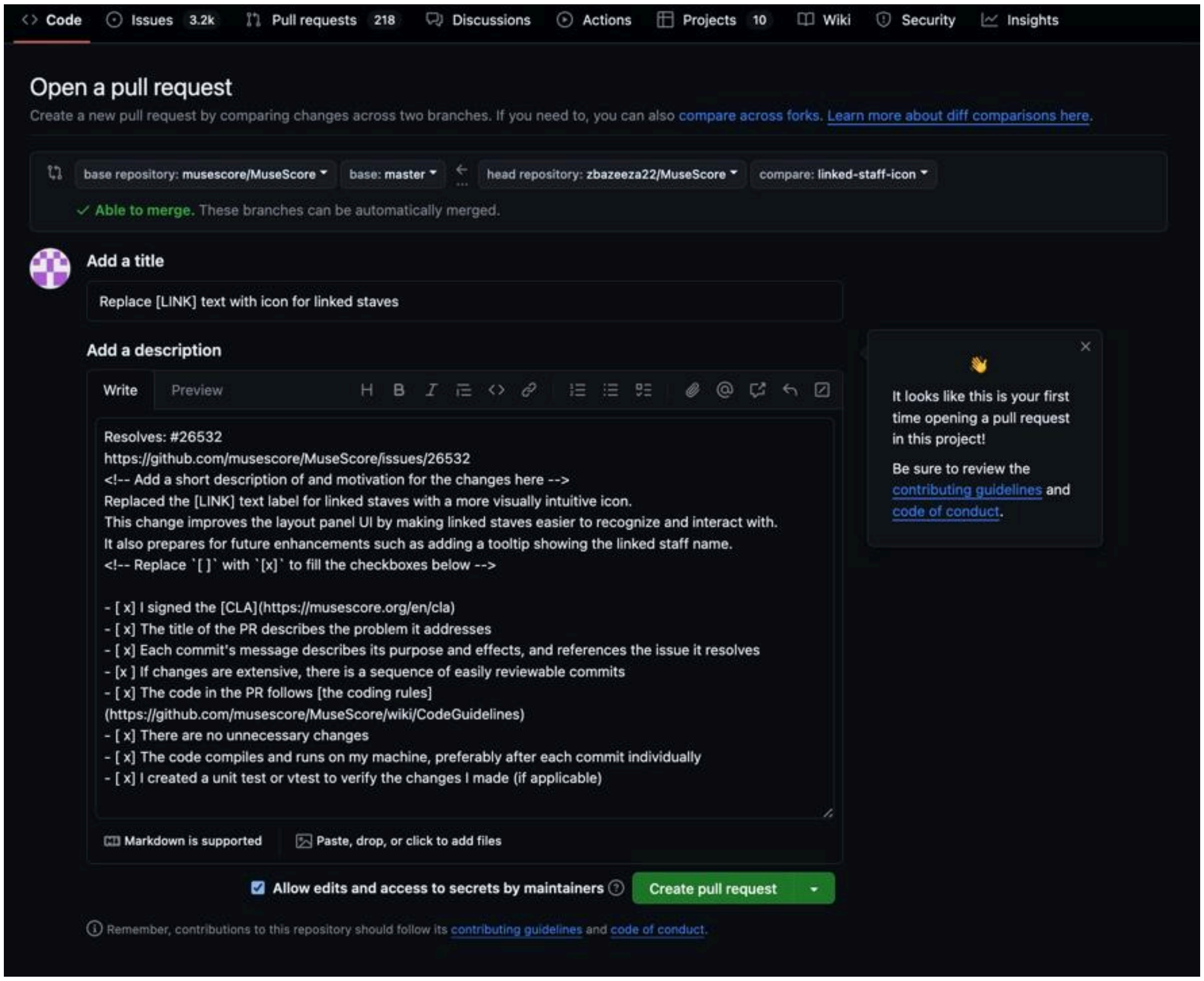


```
        }

        Row {
            anchors.left: expandButton.right
            anchors.leftMargin: 4
            anchors.right: parent.right
            anchors.rightMargin: 8
            anchors.verticalCenter: expandButton.verticalCenter
            spacing: 4

            StyledIconLabel {
                id: linkIcon
                visible: Boolean(model) && model.itemRole.isLinked
                iconCode: IconCode.LINK
                iconSize: 16
                opacity: titleLabel.opacity

                ToolTip {
                    visible: linkIcon.hovered
                    text: Boolean(model) ? qsTrc("layoutpanel", "Linked to: %1").arg(model.itemRole.linkedStaffName) : ""
                    delay: Qt.styleHints.mousePressAndHoldInterval
                }
            }
```



Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks. Learn more about diff comparisons here.

base repository: musescore/MuseScore   base: master   ← head repository: zbazeeza22/MuseScore   compare: linked-staff-icon

✓ Able to merge. These branches can be automatically merged.

Add a title

Replace [LINK] text with icon for linked staves

Add a description

Resolves: #26532
https://github.com/musescore/MuseScore/issues/26532
<!-- Add a short description of and motivation for the changes here -->
Replaced the [LINK] text label for linked staves with a more visually intuitive icon.
This change improves the layout panel UI by making linked staves easier to recognize and interact with.
It also prepares for future enhancements such as adding a tooltip showing the linked staff name.
<!-- Replace `[ ]` with `[x]` to fill the checkboxes below -->

- [x] I signed the [CLA](https://musescore.org/en/cla)
- [x] The title of the PR describes the problem it addresses
- [x] Each commit's message describes its purpose and effects, and references the issue it resolves
- [x] If changes are extensive, there is a sequence of easily reviewable commits
- [x] The code in the PR follows [the coding rules]
(https://github.com/musescore/MuseScore/wiki/CodeGuidelines)
- [x] There are no unnecessary changes
- [x] The code compiles and runs on my machine, preferably after each commit individually
- [x] I created a unit test or vtest to verify the changes I made (if applicable)

It looks like this is your first time opening a pull request in this project!

Be sure to review the contributing guidelines and code of conduct.

☑ Allow edits and access to secrets by maintainers

Remember, contributions to this repository should follow its contributing guidelines and code of conduct.

Pull request : https://github.com/musescore/MuseScore/pull/27848

Task 2

We spent several days trying to compile MuseScore on our MacBook Airs, troubleshooting Homebrew Qt paths, CMake flags, and a complex VS Code toolchain. In the process, we found GitHub issue #16424, where a Windows contributor noted that Visual Studio 2022's default "Always use CMake Presets" setting prevents proper build folder generation. Although we primarily use macOS, we recognized that this undocumented issue could affect many contributors. To address it, we created a new guide—docs/Compile-in-Visual-Studio.md—that explains the problem in plain language, details the exact menu path to disable the setting, includes full build instructions, and links to the existing Qt Creator guide. By turning a buried workaround into official documentation, we helped reduce a key onboarding barrier for Windows users.

Task Link Compile Instructions guide step to disable CMakePresets if not already done #16424

Functional Requirements

- Create a new Markdown file docs/Compile-in-Visual-Studio.md.
- Document the full Windows workflow: clone, configure, build, and debug.
- Include a reproducible procedure for disabling "Always use CMake Presets" in VS 2022.
- Cross-reference the existing Qt Creator guide for consistency.

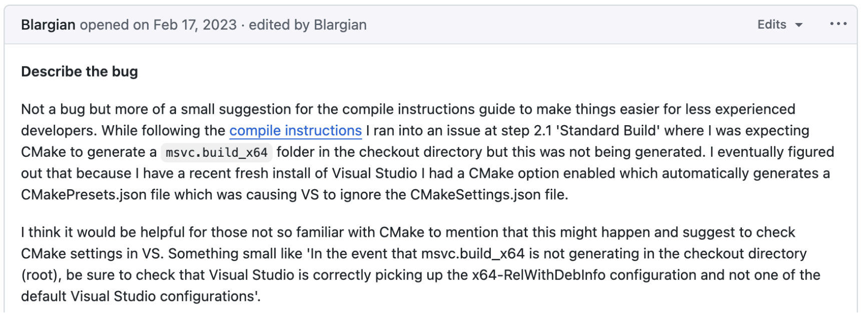
Quality Requirements

- The guide must render correctly on GitHub: headings, fenced code blocks, and internal links.
- Menu and option names must exactly match the Visual Studio 2022 UI.
- No spelling or grammar issues.
- The document must be reachable from the repository's docs/ index so newcomers can find it easily.

Task 2 Artifacts

Task evidence / Artifacts

Visual Studio will create a file called 'CMakePresets.json' then will try to build for the default VS x64Debug (or another default)

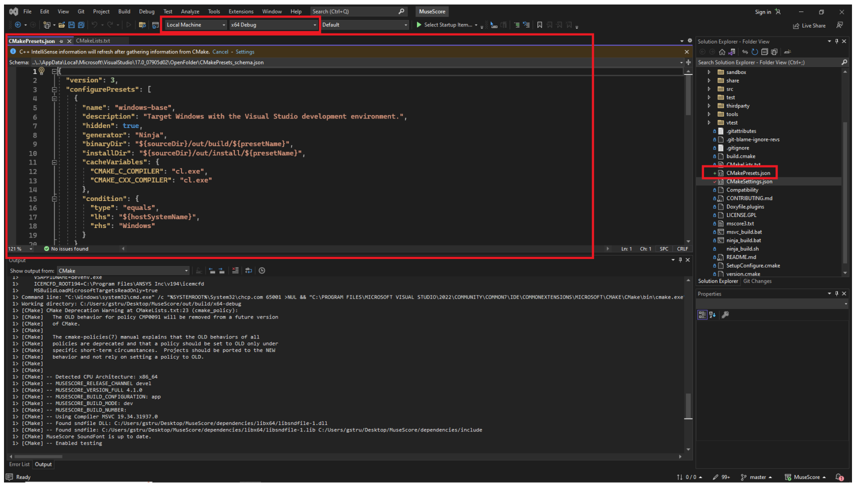


Blargian opened on Feb 17, 2023 · edited by Blargian

Describe the bug

Not a bug but more of a small suggestion for the compile instructions guide to make things easier for less experienced developers. While following the compile instructions I ran into an issue at step 2.1 'Standard Build' where I was expecting CMake to generate a `msvc.build_x64` folder in the checkout directory but this was not being generated. I eventually figured out that because I have a recent fresh install of Visual Studio I had a CMake option enabled which automatically generates a CMakePresets.json file which was causing VS to ignore the CMakeSettings.json file.

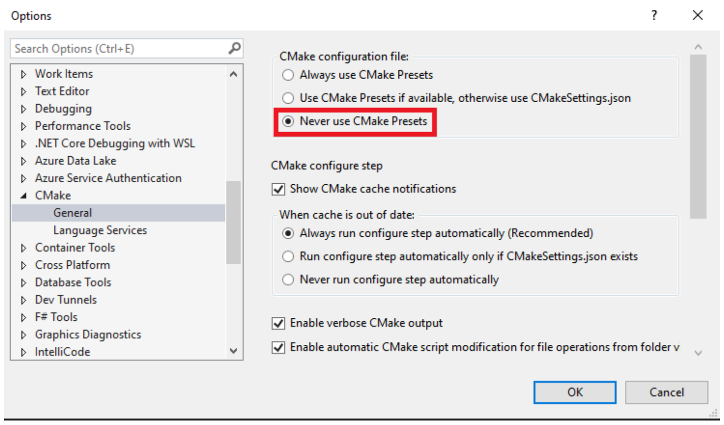
I think it would be helpful for those not so familiar with CMake to mention that this might happen and suggest to check CMake settings in VS. Something small like 'In the event that msvc.build_x64 is not generating in the checkout directory (root), be sure to check that Visual Studio is correctly picking up the x64-RelWithDebInfo configuration and not one of the default Visual Studio configurations'.



File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
MuseScore
Local Machine
x64 Debug
Default
Select Startup Item...
CMakePresets.json
CMakeLists.txt
C++ IntelliSense information will refresh after gathering information from CMake. Cancel - Settings
Schema: ..\..\AppData\Local\Microsoft\VisualStudio\17.0_07905d02\OpenFolder\CMakePresets_schema.json
121 %
No issues found
Ln: 1 Ch: 1 SPC CRLF
Output
Show output from: CMake
Error List
Output
Ready
Solution Explorer
Properties

```
{
  "version": 3,
  "configurePresets": [
    {
      "name": "windows-base",
      "description": "Target Windows with the Visual Studio development environment.",
      "hidden": true,
      "generator": "Ninja",
      "binaryDir": "${sourceDir}/out/build/${presetName}",
      "installDir": "${sourceDir}/out/install/${presetName}",
      "cacheVariables": {
        "CMAKE_C_COMPILER": "cl.exe",
        "CMAKE_CXX_COMPILER": "cl.exe"
      },
      "condition": {
        "type": "equals",
        "lhs": "${hostSystemName}",
        "rhs": "Windows"
      }
    }
```

```
1>    VSAPPIDNAME=devenv.exe
1>    ICEMCFD_ROOT194=C:\Program Files\ANSYS Inc\v194\icemcfd
1>    MSBuildLoadMicrosoftTargetsReadOnly=true
1> Command line: "C:\Windows\system32\cmd.exe" /c "%SYSTEMROOT%\System32\chcp.com 65001 >NUL && "C:\PROGRAM FILES\MICROSOFT VISUAL STUDIO\2022\COMMUNITY\COMMON7\IDE\COMMONEXTENSIONS\MICROSOFT\CMAKE\CMake\bin\cmake.exe"
1> Working directory: C:/Users/gstru/Desktop/MuseScore/out/build/x64-debug
1> [CMake] CMake Deprecation Warning at CMakeLists.txt:23 (cmake_policy):
1> [CMake]   The OLD behavior for policy CMP0091 will be removed from a future version
1> [CMake]   of CMake.
1> [CMake]
1> [CMake]   The cmake-policies(7) manual explains that the OLD behaviors of all
1> [CMake]   policies are deprecated and that a policy should be set to OLD only under
1> [CMake]   specific short-term circumstances.  Projects should be ported to the NEW
1> [CMake]   behavior and not rely on setting a policy to OLD.
1> [CMake]
1> [CMake]
1> [CMake] -- Detected CPU Architecture: x86_64
1> [CMake] -- MUSESCORE_RELEASE_CHANNEL devel
1> [CMake] -- MUSESCORE_VERSION_FULL 4.1.0
1> [CMake] -- MUSESCORE_BUILD_CONFIGURATION: app
1> [CMake] -- MUSESCORE_BUILD_MODE: dev
1> [CMake] -- MUSESCORE_BUILD_NUMBER:
1> [CMake] -- Using Compiler MSVC 19.34.31937.0
1> [CMake] -- Found sndfile DLL: C:/Users/gstru/Desktop/MuseScore/dependencies/libx64/libsndfile-1.dll
1> [CMake] -- Found sndfile: C:/Users/gstru/Desktop/MuseScore/dependencies/libx64/libsndfile-1.lib C:/Users/gstru/Desktop/MuseScore/dependencies/include
1> [CMake] MuseScore SoundFont is up to date.
1> [CMake] -- Enabled testing
```



CMake configuration file:

- ○ Always use CMake Presets
- ○ Use CMake Presets if available, otherwise use CMakeSettings.json
- ● Never use CMake Presets

CMake configure step

- ☑ Show CMake cache notifications

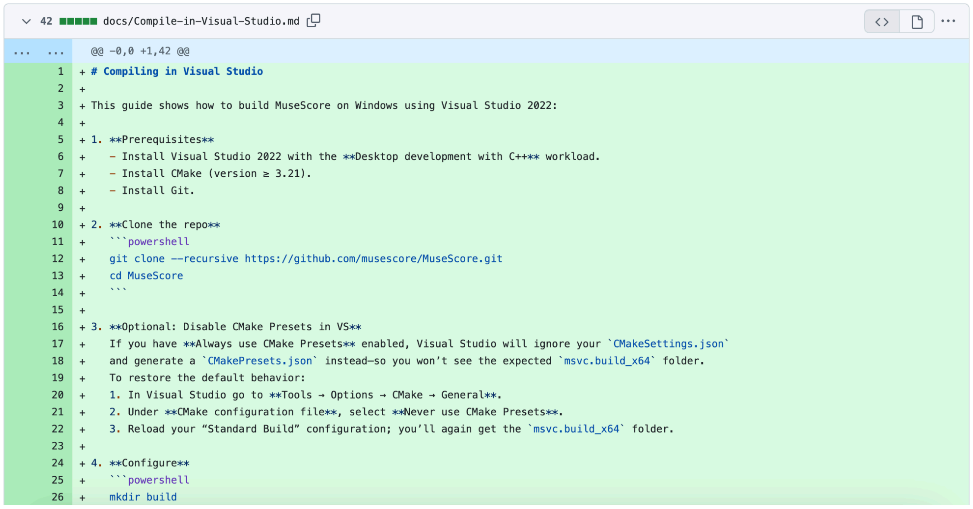
When cache is out of date:

- ● Always run configure step automatically (Recommended)
- ○ Run configure step automatically only if CMakeSettings.json exists
- ○ Never run configure step automatically

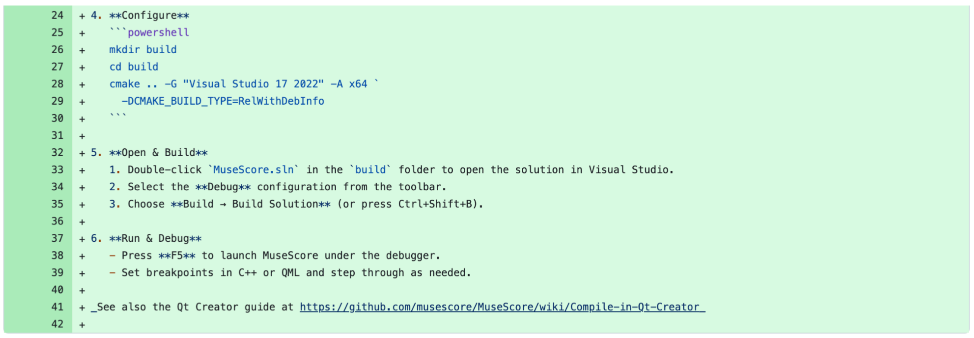
- ☑ Enable verbose CMake output
- ☑ Enable automatic CMake script modification for file operations from folder v

After changing this setting ^^ CMake will correctly generate the msvc.build_x64 folder

Created and Submitted **docs/Compile-in-Visual-Studio.md**, reproduced below



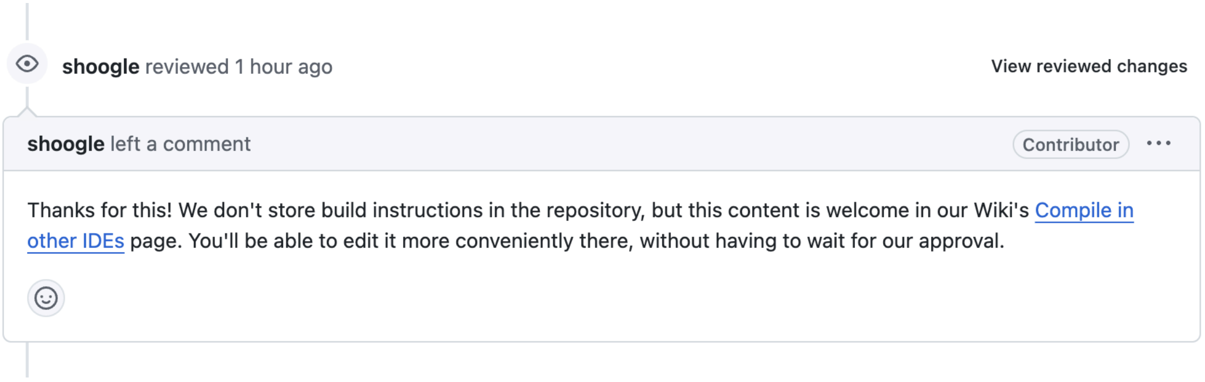
```
42 docs/Compile-in-Visual-Studio.md
@@ -0,0 +1,42 @@
+ # Compiling in Visual Studio
+
+ This guide shows how to build MuseScore on Windows using Visual Studio 2022:
+
+ 1. **Prerequisites**
+    - Install Visual Studio 2022 with the **Desktop development with C++** workload.
+    - Install CMake (version ≥ 3.21).
+    - Install Git.
+
+ 2. **Clone the repo**
+    ```powershell
+    git clone --recursive https://github.com/musescore/MuseScore.git
+    cd MuseScore
+    ```
+
+ 3. **Optional: Disable CMake Presets in VS**
+    If you have **Always use CMake Presets** enabled, Visual Studio will ignore your `CMakeSettings.json`
+    and generate a `CMakePresets.json` instead—so you won't see the expected `msvc.build_x64` folder.
+    To restore the default behavior:
+    1. In Visual Studio go to **Tools → Options → CMake → General**.
+    2. Under **CMake configuration file**, select **Never use CMake Presets**.
+    3. Reload your "Standard Build" configuration; you'll again get the `msvc.build_x64` folder.
+
+ 4. **Configure**
+    ```powershell
+    mkdir build
```



~~~~
+ 4. **Configure**
+    ```powershell
+    mkdir build
+    cd build
+    cmake .. -G "Visual Studio 17 2022" -A x64 `
+      -DCMAKE_BUILD_TYPE=RelWithDebInfo
+    ```
+
+ 5. **Open & Build**
+    1. Double-click `MuseScore.sln` in the `build` folder to open the solution in Visual Studio.
+    2. Select the **Debug** configuration from the toolbar.
+    3. Choose **Build → Build Solution** (or press Ctrl+Shift+B).
+
+ 6. **Run & Debug**
+    - Press **F5** to launch MuseScore under the debugger.
+    - Set breakpoints in C++ or QML and step through as needed.
+
+ _See also the Qt Creator guide at https://github.com/musescore/MuseScore/wiki/Compile-in-Qt-Creator_
+
~~~~

Pull request: https://github.com/musescore/MuseScore/pull/27849

Although our pull request wasn't merged into MuseScore's main repository due, a project maintainer appreciated our contribution and requested that we add our instructions directly to the official MuseScore Wiki on the 'Compile in other IDEs' page. This feedback highlights the community's recognition of our work's immediate practical value.



shoogle reviewed 1 hour ago

View reviewed changes

shoogle left a comment

Contributor

Thanks for this! We don't store build instructions in the repository, but this content is welcome in our Wiki's Compile in other IDEs page. You'll be able to edit it more conveniently there, without having to wait for our approval.

QA Strategy

For quality assurance, we performed a structured series of activities focused on validating both the functional and visual correctness of our contributions to the MuseScore project.

1. Unit Testing:

As part of MuseScore's standard QA practices, we ran the project's suite of automated unit tests. These tests cover key areas such as accessibility, UI interactions, audio playback, plugin behavior, and system diagnostics. Running these tests allowed us to verify that our changes did not introduce regressions into critical subsystems unrelated to our modification. While MuseScore's test suite includes broader system-level coverage, we focused on test suites like accessibility_tests and ui_tests, which are most relevant to our UI-based updates. MuseScore integrates these unit tests into its continuous integration (CI) system on GitHub, so completing them locally ensured a smoother PR review process and aligned with project expectations.

2. Visual Testing (VTests):

Since our first contribution primarily involved a UX-related change that modified the visual appearance of MuseScore's interface, we also performed Visual Tests (VTests). MuseScore uses a specialized workflow for detecting regressions in score rendering by comparing output images from two builds—a "reference build" (from master) and the "current build" containing our changes. We generated PNG images of test scores from both builds and used MuseScore's VTest comparison scripts (which utilize ImageMagick) to automatically compare them. We reviewed the resulting HTML report to confirm that no unintended layout shifts, rendering issues, or visual regressions were introduced.

3. Markdown Documentation Preview Testing:

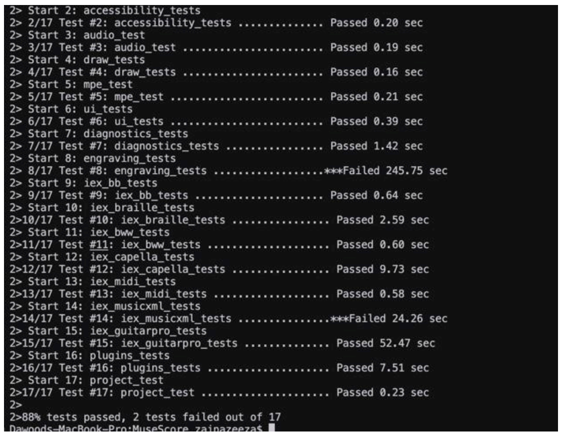
For Task 2, which involved contributing a new Markdown documentation file (docs/Compile-in-Visual-Studio.md), we validated formatting and readability using both local and remote renderers.

- Locally, we used VS Code's Markdown Preview feature (Cmd+Shift+V) to ensure headings, code blocks, and spacing were properly formatted.
- On GitHub, we reviewed the rendered output in the Pull Request to confirm everything displayed as expected.
- We also confirmed all PowerShell and CMake code examples were syntactically correct and tested copy-paste-ready.
- Finally, our PR was reviewed by a project contributor who confirmed the doc was helpful and better suited for the Wiki.

By combining functional verification through unit testing, visual comparison through VTests, and format validation for documentation, we followed a QA strategy tailored to the nature and risk surface of each task. These QA methods align with MuseScore's established contribution and review process, emphasizing cross-platform usability, visual integrity, and documentation clarity.

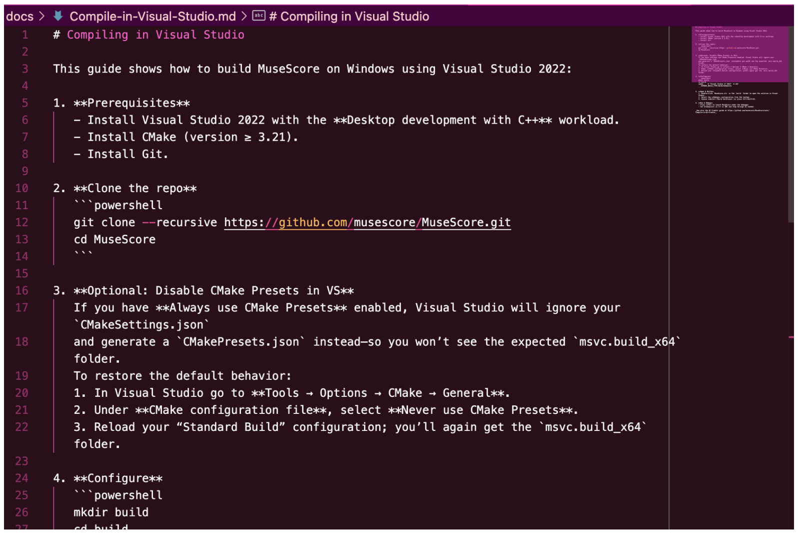
QA Evidence

The screenshot shows the results of running the unit test suite locally. A total of 17 test suites were executed, covering a wide range of functional components. Out of these, 15 tests passed successfully, while 2 tests failed: engraving_tests and iex_musicxml_tests. Based on the project's documentation and my own local observations, these failed tests are known to be unrelated to the parts of the system my changes touched. Specifically, the engraving_tests target deep internal layout rules



```
2> Start 2: accessibility_tests
2> 2/17 Test #2: accessibility_tests ............... Passed 0.20 sec
2> Start 3: audio_test
2> 3/17 Test #3: audio_test ........................ Passed 0.19 sec
2> Start 4: draw_tests
2> 4/17 Test #4: draw_tests ........................ Passed 0.16 sec
2> Start 5: mpe_test
2> 5/17 Test #5: mpe_test .......................... Passed 0.21 sec
2> Start 6: ui_tests
2> 6/17 Test #6: ui_tests .......................... Passed 0.39 sec
2> Start 7: diagnostics_tests
2> 7/17 Test #7: diagnostics_tests ................. Passed 1.42 sec
2> Start 8: engraving_tests
2> 8/17 Test #8: engraving_tests ..................***Failed 245.75 sec
2> Start 9: iex_bb_tests
2> 9/17 Test #9: iex_bb_tests ...................... Passed 0.64 sec
2> Start 10: iex_braille_tests
2>10/17 Test #10: iex_braille_tests ................ Passed 2.59 sec
2> Start 11: iex_bww_tests
2>11/17 Test #11: iex_bww_tests .................... Passed 0.60 sec
2> Start 12: iex_capella_tests
2>12/17 Test #12: iex_capella_tests ................ Passed 9.73 sec
2> Start 13: iex_midi_tests
2>13/17 Test #13: iex_midi_tests ................... Passed 0.58 sec
2> Start 14: iex_musicxml_tests
2>14/17 Test #14: iex_musicxml_tests ..............***Failed 24.26 sec
2> Start 15: iex_guitarpro_tests
2>15/17 Test #15: iex_guitarpro_tests .............. Passed 52.47 sec
2> Start 16: plugins_tests
2>16/17 Test #16: plugins_tests .................... Passed 7.51 sec
2> Start 17: project_test
2>17/17 Test #17: project_test ..................... Passed 0.23 sec
2>
2>88% tests passed, 2 tests failed out of 17
Dawoods-MacBook-Pro:MuseScore zainazeeza$
```

affecting musical engraving, and the iex_musicxml_tests focus on MusicXML import/export parsing logic. Since my change was limited to improving the visual linking of staves in the UI and did not affect layout calculation algorithms or file I/O, these failures are irrelevant to our contribution. Furthermore, my local build of MuseScore continued to function correctly after the changes. We manually verified the intended functionality inside the application and confirmed that the updated UX behavior appeared as expected, without triggering any crashes, regressions, or errors in normal operation.



docs > Compile-in-Visual-Studio.md > # Compiling in Visual Studio

~~~
# Compiling in Visual Studio

This guide shows how to build MuseScore on Windows using Visual Studio 2022:

1. **Prerequisites**
   - Install Visual Studio 2022 with the **Desktop development with C++** workload.
   - Install CMake (version ≥ 3.21).
   - Install Git.

2. **Clone the repo**
   ```powershell
   git clone --recursive https://github.com/musescore/MuseScore.git
   cd MuseScore
   ~~~

3. **Optional: Disable CMake Presets in VS**
   If you have **Always use CMake Presets** enabled, Visual Studio will ignore your `CMakeSettings.json`
   and generate a `CMakePresets.json` instead—so you won’t see the expected `msvc.build_x64` folder.
   To restore the default behavior:
   1. In Visual Studio go to **Tools → Options → CMake → General**.
   2. Under **CMake configuration file**, select **Never use CMake Presets**.
   3. Reload your “Standard Build” configuration; you’ll again get the `msvc.build_x64` folder.

4. **Configure**
   ```powershell
   mkdir build
```

^ Preview of Compile-in-Visual-Studio.md in VS Code’s Markdown renderer to validate formatting, structure, and code block visibility prior to PR submission

Plan Updates

Initially, we planned to contribute to the Home Assistant project, focusing on enhancing sensor data validation and improving the integration configuration UI. However, we encountered major difficulties setting up the full development environment due to extensive dependencies, compatibility issues, and complicated Python version mismatches. We spent approximately 12 hours troubleshooting these setup issues but ultimately decided to pivot to another project after it became clear these problems would not be resolved quickly.

This forced us to switch our project entirely. After exploring alternatives, we selected MuseScore for its strong community support and clear contributor guidelines. This change was not anticipated, which significantly impacted our original timeline.

Upon switching to MuseScore, we faced a new set of challenges. Setting up MuseScore on our MacBooks required about 10 hours, significantly exceeding the 3 hours we originally budgeted for environment setup. Resolving issues related to Homebrew Qt paths and CMake configurations were more complicated than expected.

For Task 1 in MuseScore, replacing the “[LINK]” label with a visual icon, we originally estimated around 8 hours total (5 for analysis and 3 for implementation). In practice, it took approximately 12 hours, including time spent familiarizing ourselves with the MuseScore UI framework, integrating the icon and tooltip logic, and thoroughly testing across different views.

Task 2, creating documentation for disabling “Always use CMake Presets” in Visual Studio, was not in our original plans. We added this task after realizing that a crucial workaround for Windows developers was undocumented. Completing this documentation took about 6 hours, slightly above expectations because we wanted to ensure consistency with existing guides.

Our original schedule anticipated approximately 50 combined hours split evenly between the two initial Home Assistant tasks. In reality, we spent closer to 60 combined hours, with a significant portion of that time dedicated to environment setup and overcoming unexpected hurdles with MuseScore’s complex build system.

Overall, deviations primarily stemmed from unforeseen technical setup complexities, underestimation of the initial learning curve for MuseScore’s codebase, and added time spent on an entirely new documentation task. These challenges significantly altered our planned schedule but provided valuable experience in navigating real-world open-source contributions and adapting quickly to new, unexpected issues.

Experiences and Recommendations

Because we had to switch projects early on, we experienced firsthand the significant challenges associated with compiling and setting up large open-source projects. Initially, we selected a different project and managed to get the test runs working successfully. However, compiling the entire project turned out to be far more difficult than expected. This taught us that small technical details, such as operating system version, development environment, and tooling, can have a major impact on the ability to work with open-source software. Every project’s infrastructure and dependencies are often tuned to specific versions, and slight mismatches can cause serious build issues. Eventually, after struggling to make meaningful progress, we made the decision to switch our focus to MuseScore.

During the process of searching for projects, we learned that the open-source community is very eager to welcome new contributors. Many projects had actively curated lists of beginner-friendly tasks ("good first issues") and posted clear guidelines for new participants. This lowered the barrier of entry to the open-source world significantly. However, while the community's welcoming attitude was a positive surprise, the technical setup challenges remained a persistent hurdle.

Once we moved to working on MuseScore, we encountered even more difficulties building and running the project locally. One of the major challenges was that much of the official documentation was outdated, especially for macOS users like ourselves running version 11.6.5.

Specific problems arose when attempting to configure and install dependencies like Qt, which were highly sensitive to OS version differences. We ran into version mismatches, missing libraries, and dependency problems that were not clearly documented. Although MuseScore provided written instructions for setup, they were not always detailed enough for the particular configuration we were using. It took several hours — around 10 hours total — to fully troubleshoot, install the necessary packages, and get the project to compile and build locally. This process really emphasized to me how critical up-to-date, platform-specific setup guides are in maintaining a healthy open-source contributor pipeline.

Understanding MuseScore's codebase itself presented another major challenge. The project contains hundreds of files spanning a massive and highly interconnected codebase. For someone used to school projects or personal side projects — where codebases are often neatly structured and relatively small — it was initially overwhelming to even figure out where to start.

We spent several hours exploring the directory structure and documentation just to get a sense of the architecture. One strategy that helped was leveraging documentation on the GitHub page that explained how to search for specific parts of the codebase using keywords. We also used our IDE's navigation features (like "go to definition") to jump through files more efficiently. Additionally, we leaned on what we had learned in classes like EECS 281, where we also organized code using header files and implementation files. Seeing MuseScore follow a similar convention with .h and .cpp files made the structure a little more familiar and less intimidating over time.

When interacting with the MuseScore open-source community, we found the maintainers to be extremely supportive. We primarily used Discord to reach out when we needed help setting up the project, such as configuring CMake properly. The community members were very responsive when they realized we were first-time contributors, providing fast and detailed replies even when questions were basic. When Discord was not immediately responsive, leaving comments directly on GitHub issues also proved effective. We were pleasantly surprised at how fast the responses came back — often within a few hours. One thing we wish had been available, though, would be a step-by-step video tutorial on setting up the build environment. We recognize that this is unrealistic given the number of operating systems and versions people might use, but a more centralized or updated documentation hub would have made things easier.

In terms of working with the code itself, one of the biggest differences from our prior experiences (in coursework and personal projects) was simply how difficult it was to understand what each part of the code was doing. In our past projects, the codebases were much smaller and often well-scoped to a particular class assignment. In MuseScore, trying to trace through even a single function call could mean jumping through multiple layers of abstraction, modules, and files. It was often unclear at first what a function's role was or what other parts of the project it interacted with. We were also nervous about accidentally breaking something major in the project. In smaller projects, it's easy to roll back or rework changes quickly, but in large-scale projects like MuseScore, introducing a mistake could mean affecting multiple subsystems and forcing a much more difficult debugging process.

Despite these challenges, the project collaboration culture within MuseScore really helped. Maintainers were friendly, the community was active, and there was an overall welcoming tone for new contributors. Without this support system, we think the technical challenges could have been overwhelming. Even though we struggled at times, knowing that help was available made it much easier to keep pushing forward when we hit roadblocks.

If we were starting a new open-source project from scratch, one major change we would make is to test the build environment immediately after selecting a project, even before choosing a specific task. Alternatively, we would reach out early through GitHub Discussions or Discord to ask directly about the difficulty level of setting up the project for beginners. Doing this upfront would have saved us many hours that we lost switching between projects and troubleshooting builds. We would also suggest that future contributors spend time exploring the project’s file structure thoroughly before committing to a task, just to avoid underestimating the time needed to understand where and how to make changes.

Overall, this experience taught us far more than just how to submit a pull request. We gained a much deeper understanding of the real-world obstacles in distributed software development — from environment setup frustrations, to communication best practices, to maintaining resilience when things do not go according to plan. It gave us a strong appreciation for the importance of clear documentation, organized mentorship, and having a patient community in large-scale software projects. While this was our first time contributing to a system of this size, it has given us much greater confidence to work on complex projects in the future and a strong motivation to continue participating in open-source communities.

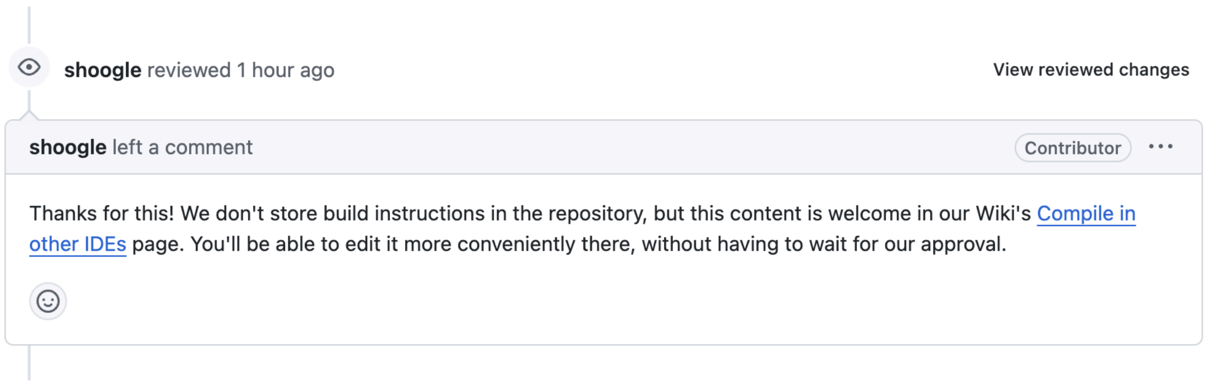
Advice for Future Students

Setting up the development environment and testing software is usually the hardest part, so try to get a head start on that early.

Regularly communicate with maintainers and the community early on; they can significantly simplify your setup process and clarify task difficulty.

EC

Our changes were invited into the official Wiki



shoogle reviewed 1 hour ago

View reviewed changes

shoogle left a comment

Contributor

Thanks for this! We don't store build instructions in the repository, but this content is welcome in our Wiki's Compile in other IDEs page. You'll be able to edit it more conveniently there, without having to wait for our approval.