TEAM DATA - DELIVERABLE 2

11.03.2020

—

DHARMIK SHAH
YUN JIE (JEFFREY) LI
JEFFREY SO
ALVIN TANG
JIMMY PANG

Table of Contents

Promising Bugs and Features

Empty Lines Not Getting Commented Out

Github Issue #88480: Empty Lines Not Getting Commented-Out

Initially a reported bug, this ticket is now part of a feature request for Visual Studio Code. The issue was opened on January 11 by a user who believed that the editor not commenting out entirely whitespace or empty lines was a bug. However, it was later confirmed to be the expected behaviour of the editor.

Afterwards, a Github user suggested that a setting could be added to give developers a preference on this issue. They suggested that a setting such as `"commenting.includeEmptyLines": true` could be added to allow users to choose whether or not to comment out empty lines as well when using the line comment commands. These commands include adding/removing comments(CtrlCmd + K and CtrlCmd + C) and toggling comments (CtrlCmd + /). With enough upvotes, the VSCode bot automatically added this feature request into their backlog. This feature request could take up to **12 hours** to complete mostly because two portions needed to be figured out: adding the setting into the editor and adjusting the existing code based off the setting.

Concatenation of “+” at the end of key combinations in screencast mode

Github Issue #91235: "+" is concatenated at the end of key combination in screencast mode

This bug was reported on February 23, and it shows that there is a trailing ‘+’ sign on key combinations when the combination does not end with a non modifier key in screencast mode. To figure out the components required, it should take **2-3 hours** to track down the location the string is passed from and find any tests that need to be modified. Risks involved in fixing this bug depends on what level the bug is fixed at, if the bug is fixed at low level, it could affect other processes that use the function that adds the “+” seperator. In that case, it would be good to investigate all test cases that involve the function and see if any modifications need to be made or if there is a more upper level fix for the issue.

Side note: Screencast mode is a hidden tool in Visual Studio Code that highlights a cursor position and keystrokes for users making videos/gifs. This mode can be enabled through the command palette.

Allow Selection of Multiple Editor Tabs

Github issue #89958: Allow Selection of Multiple Editor Tabs

This ticket was reported as a feature request on February 3. The requested feature was to be able to select multiple editor tabs in Visual Studio Code by holding the shift key and selecting the tabs with your mouse. The ticket was highly upvoted and has been moved to the feature backlog. We considered to work on this ticket as it was potentially a good way to gain insight to the workspace functionality in VS Code and how the tabs are managed, however it most likely involved a lot of time spent looking for how certain keypresses are listened to, and a new UI element showing multiple tabs being selected needed to be created. Overall this ticket could take up to an **entire day** to properly implement and test.

Add “Save File” to the context menu of a Tab

GitHub Issue #90231: Add "Save File" to the context menu of a Tab

This ticket was reported as a feature request on February 7. The requestor wanted to be able to save from the tab menu, in addition to saving from the open editors menu. The feature request has been a candidate for VS Code’s backlog since February 10. Though there was only one upvote for this ticket, we considered working on this ticket as it appears to be a good first issue, but it involves modifying the graphical user interface which may take almost half a day to figure out. Given the large complex codebase of Visual Studio Code involving user interface elements, we chose not work on this request for the time being. This would likely take more than **2 days** to fully understand and implement.

Wrong color of console.log token

Github Issue #91818: Wrong color of console.log token

This ticket was reported as a feature request on February 29. This reported issue, points out that using a fresh installation of Visual Studio Code with the default dark theme named “Dark + “, the editor incorrectly highlights key words for the Javascript language. The post includes a screencap showing what the proper highlighted colour is supposed to be as well as what colour is incorrectly being displayed. The poster also included (in a previous instance) a video showing that the proper colour does show up for a brief moment, but quickly gets replaced, the description still remains on the page.

A Github user suggested a potential workaround for the discolouration by disabling a setting called `semantic highlighting`. However since this is a loophole and not an actual solution, investigation needs to be done to find the source of the overlapping incorrect colour as well as potentially how the semantic highlighty actually functions to prevent it from overlapping colours. Overall this bug fix is estimated to take up to **6 hours** to investigate thoroughly.

Solutions and Commentary

Feature Request: Empty Lines Not Commented Out

Github Issue #88480: Empty Lines Not Getting Commented-Out
Branch: team19/comment-whitespace-88480 (CSCD01/vscode-team19)

We believed that this issue was a good choice as a first issue due to the underlying concept of how Visual Studio Code registers “selected lines”. Since VS Code is all about text editing, we were curious to understand what content is being selected and how to modify it to appropriate standards such as: language syntax conventions, language coloration, function calls. Additionally, since the feature request implies key binded function calls will be invoked (for example: `Ctrl+/`, to comment code blocks), it may be easier and more time efficient to locate the code that needs to be investigated and edited, compared to some of the other issues.

Overall, this feature request was implemented by modifying the behaviour of line-commenting and how VS Code detects whitespace lines. Additionally, we implemented the feature as a toggleable user setting, this introduces versatility for users who prefer to be able to distinguish code blocks from each other, without disrupting the experience of other normal users.

The following source files had to be updated:

- `editor/contrib/comment/comment.ts`
- `editor/contrib/comment/lineCommentCommands.ts`
- `editor/common/config/editorOptions.ts`
- `src/vs/monaco.d.ts`

The following test file had to be updated:

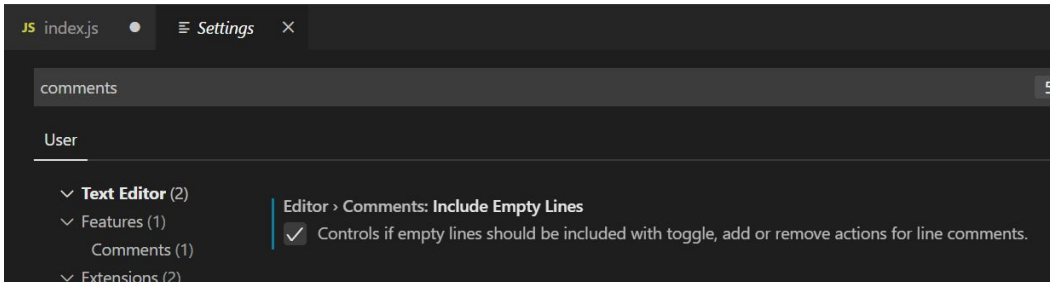
- `editor/contrib/comment/test/lineCommentCommand.test.ts`

comment.ts, editorOptions.ts, and monaco.d.ts

The file `editor/contrib/comment/comment.ts` is responsible for defining the actions that can be applied for commenting. `run()` in abstract class `CommentLineAction` which is extended by `ToggleCommentLineAction`, `AddLineCommentAction`, and `RemoveLineCommentAction`, actually runs the action based on the preferences of the user and the type of command they issued for the action. You can see that in a local variable, `commentOptions` in `run()`, gets a set of editor specific options set by the user in their VSCode settings (User or Workspace). Then for each selection of the line comment action, we take the `includeEmptyLines` option and add it as a parameter of a new instance of `LineCommentCommand` (each selection calls this command for such an action). `LineCommentCommand` is described in the next section.

```
for (const selection of selections) {
      commands.push(new LineCommentCommand(
            ...
            commentsOptions.insertSpace,
            commentsOptions.includeEmptyLines
      ));
}
```

`includeEmptyLines`, specifically `editor.comments.includeEmptyLines` was a setting that we added into `editor/common/config/editorOptions.ts` and `src/vs/monaco.d.ts`. These two files overall describe all the editor specific settings in VS Code. By adding the `includeEmptyLines` flag (and description) in both of these files, it ultimately allowed us to add the desired flag to the settings view as seen below.



We estimated the effort to be more than it did to implement. This is because we were originally adding the new setting under *Features* instead of *Text Editor*, which actually adds more complexity as we had to deal with dependency injection of `IConfigurationService` into the tests and the codebase. However, we later reverted our progress as we thought the setting should fall under the *Text Editor* category.

lineCommentCommands.ts

In the file `editor/contrib/comment/lineCommentCommands.ts`, a private static method `_analyzeLines` of `LineCommentCommand` analyzes a set of lines which decides which lines are relevant to have comments added or removed based on the command from the user. As we insert a boolean flag `includeEmptyLines` in the constructor of `LineCommentCommand`, we pass the value through to `_gatherPreflightData` which then calls `_analyzeLines` where we pass the value in again. Then inside `_analyzeLines`, there is a loop that iterates over each selected line, setting values for each object representing a line. One such key in the object is `ignore`, which allows other operations outside of `_analyzeLines` to decide whether or not to ignore that line.

Locally in `_analyzeLines`, information about a line selected within the loop (iterating over each line) is stored in a `lineData` object that includes the attribute `ignore`. The following code block in the original code determines whether or not the line is entirely whitespace:

```
if (lineContentStartOffset === -1) {
      // Empty or whitespace only line
      if (type === Type.Toggle) {
            lineData.ignore = true;
      } else if (type === Type.ForceAdd) {
            lineData.ignore = true;
      } else {
            lineData.ignore = true;
      }
      lineData.commentStrOffset = lineContent.length;
      continue;
}
```

As seen in the snippet, if the line is whitespace/empty then the `ignore` attribute is set to `true`. Whitespace lines in VS Code have a start offset of `-1`, and ignored lines are not commented.

As the `_analyzeLines` method is modified so that it takes in the boolean parameter `includeEmptyLines`, the `lineData.ignore` is now set to `!includeEmptyLines` for all cases, with `includeEmptyLines` defaulted to `false`.

We were careful in modifying this portion of the codebase because we anticipated that if we messed up here, it could mess up specific edge cases when using commands for line commenting (e.g. line commenting a selected line that is just whitespace). As such, we estimated that this would take half of the hours we predicted.

lineCommentCommand.test.ts

In this file, a suite of tests were added under the main suite of tests, where we overwrote `testLineCommentCommand`, a function used in every test that actually creates an instance of `LineCommentCommand` for each test. We passed in `true` for the parameter `includeEmptyLines` when initializing the object within that suite. In the suite, we ran multiple tests:

- Does not ignore whitespace lines - main thing to test
- Removes its own comments when toggling line comments - same as if `false`
- Comments should work only when in whitespace - same as if `false`
- Comments single line - same as if `false` and tests a line that has characters
- Detects indentation - same as if `false`

Most of the tests above tested that the same output occurred if `includeEmptyLines` was `false` to ensure that each edge case worked as expected to ensure stability as the setting should not affect cases other than when selecting multiple lines with whitespace lines.

Note that the main suite of tests had set `includeEmptyLines` to `false` as each original test anticipated for empty lines to not be commented as that was what was expected by the authors of the codebase. Therefore, no modification was needed and only another suite nested over this suite was added in order to test when `true`.

We anticipated that this would be easy but understanding how to select lines for each test case proved difficult - it meant understanding the `Selection` class which we ended up doing by understanding how each line was selected in each of the original test cases.

Bug Fix: Concatenation of Plus Character in Screencast Mode

Github Issue #91235: "+" is concatenated at the end of key combination in screencast mode
Branch: team19/screencast-concat-91235 (CSCD01/vscode-team19)

This feature request seemed like a good ticket to start on because it was marked as a good first ticket to take on. Additionally, it could give us good insight to how data is passed throughout the VScode application and how information is displayed on the screen. In order to fix this bug, we had to change the way the key presses are parsed and how the “+” separator is added.

The following source file was modified:

- `src/vs/base/common/keybindingLabels.ts`

The following test files were modified:

- `workbench\services\keybinding\test\electron-browser\macLinuxFallbackKeyboardMapper.test.ts`
- `workbench\services\keybinding\test\electron-browser\windowsKeyboardMapper.test.ts`
- `workbench\services\keybinding\test\electron-browser\macLinuxFallbackKeyboardMapper.test.ts`

keybindingLabels.ts

The + character is only added when the key combination input contains a “modifier” key (e.g. ALT, CTRL, WINDOWS key). The function `_simpleAsString` pushes all the key string representations into a list and then joins them with the separator (+ in this case). However, if only modifier keys are pressed, an empty representing an actual key press (not modifier) is appended to the list, meaning that it will join the separator “+” to the end of the string even though there is nothing behind it. To fix this issue, we added an empty string check to the appending of the key label:

```
if (key) {
      result.push(key);
}
```

After removing the trailing “+” symbol, we fixed all tests that tested the label provider that had only the modifier as input by simply removing the trailing “+” sign in the expected output.

Another way to fix this could be to simply check in where the screencast mode gets the string if the label ends with a “+” and just take it out. This method means that we will not have to change any tests and will fix this specific issue only.

Our Software Process

Trello board: https://trello.com/b/JPvqOrCc/vs-code

As mentioned in our previous deliverable, we decided to use Kanban as our software development process and so we did. We used a digital tool, Trello to visualize our progress using these columns: Backlog, To Do, In Progress and Complete.

The **Backlog** column was used for any ticket that we considered to do, but we chose not to use it for this current deliverable because everything was clear on what to do and there weren't too many tickets that made us use this column. We expect to use this column when further extending VS Code for the next few deliverables.

The **To Do** column was used for the tickets that we plan to complete in the current deliverable. Each card would thus be assigned to at least one team member. For the column, we started off with cards to research open issues on Github but cards were later added for implementing each chosen issue and the tests of each implementation. This allowed us to easily breakdown what needed to be done and efficiently assign work. As predicted, Kanban allowed us to be flexible with how to do work and when.

The **In Progress** column was used for any assigned tickets that were currently being worked on. We didn't have a WIP limit enforced because each ticket was small in size and tickets were dynamically being written. However, a WIP limit may be decided upon for other deliverables as we expect the number of tickets to increase.

Finally, tickets assigned to the **Complete** column were completed and verified by at least another member of the team.

In order to further organize our Trello board, in anticipation for more Task Cards, our team developed a naming convention for easy identification purposes. Cards that begin with FR represent work for feature requests, BF represents bug fixes, and DOC represents documentation. Even though we did end up creating tickets dynamically by documentation, the implementation and then the test of each implementation, FR and BF allowed us to separate the two items from our shortlist as we focused on one feature request and one bug fix. This separation made it easy for everyone to focus on their own respective tasks