

Creating a Formatter Extension

November 21, 2016 by Johannes Rieken, @johannesrieken (<https://twitter.com/johannesrieken>)

Since its introduction, the Visual Studio Code extension API has provided support for source code formatters. The first language extensions we built, for example TypeScript, C# and Go, used the formatting API. We wrote this blog to explain the best practices for implementing formatters.

VS Code's extension API follows a set of guiding principles. The essence of these principles is that VS Code provides the skeleton and extensions provide the "smarts". The common pattern is for VS Code to provide the UI around a feature and the extensions provide the necessary data to make it shine.

The core benefit of using the extension API for implementing a formatter comes from the exposure of the **Format Document** and **Format Selection** actions. These actions are available in the editor context menu, bound to keyboard shortcuts, and visible in the **Command Palette**. Using the API leads to a consistent user experience across all formatter extensions.

The Formatting API

The code snippets below show what to do and what not to do when implementing a formatter. The best practice is to use the formatting API and not create a new action, such as "Format Foo File." The full extension example can be found on GitHub (<https://github.com/jrieken/vscode-formatter-sample>).

```
// 🐛 formatter implemented as separate command
vscode.commands.registerCommand('extension.format-foo', () => {
  const { activeTextEditor } = vscode.window;

  if (activeTextEditor && activeTextEditor.document.languageId === 'foo-lang') {
    const { document } = activeTextEditor;
    const firstLine = document.lineAt(0);

    if (firstLine.text !== '42') {
      const edit = new vscode.WorkspaceEdit();
      edit.insert(document.uri, firstLine.range.start, '42\n');

      return vscode.workspace.applyEdit(edit);
    }
  }
});

// 🍌 formatter implemented using API
vscode.languages.registerDocumentFormattingEditProvider('foo-lang', {
  provideDocumentFormattingEdits(document: vscode.TextDocument): vscode.TextEdit[] {
    const firstLine = document.lineAt(0);
    if (firstLine.text !== '42') {
      return [vscode.TextEdit.insert(firstLine.range.start, '42\n')];
    }
  }
});
```

Recently, we added the "Format on Save" feature. An extension properly implementing the formatting API supports this feature without any new code.

Tip: To take advantage of this, a formatting extension needs to be registered using the `registerDocumentFormattingEditProvider` (<https://github.com/microsoft/vscode/blob/a80f904d4e9afa7a05ae57a0123c11b727097129/src/vs/vscode.d.ts#L10089>) API call.

Multiple Formatters

A common misunderstanding is that when contributing a formatter, you must support all programming languages. When an extension registers as a formatter with `registerDocumentFormattingEditProvider` (<https://github.com/microsoft/vscode/blob/a80f904d4e9afa7a05ae57a0123c11b727097129/src/vs/vscode.d.ts#L10089>), it indicates with a `DocumentSelector` (<https://github.com/microsoft/vscode/blob/a80f904d4e9afa7a05ae57a0123c11b727097129/src/vs/vscode.d.ts#L1940>) which programming languages it supports. With that information, the editor can enable the formatting actions when for example, an HTML document is open. Likewise, the editor will disable the formatting actions when displaying documents for which no formatter is registered.

What happens when there are multiple formatters for one language? This can be a problem when different formatters' actions contradict. In the October release, we added settings to enable or disable the default formatters that ship with VS Code. The best practice is for extension authors to add a similar setting as what we did in VS Code as shown below.

```
"html.format.enable": true,
"javascript.format.enable": true,
"typescript.format.enable": true,
"json.format.enable": true
```

An extension adds settings through the `contributes.configuration` (/docs/extensionAPI/extension-points#_contributesconfiguration) extension point.

Formatters in the Marketplace

Last, we want to bring more awareness to formatters and have added a new "Formatters" category (<https://marketplace.visualstudio.com/search?target=VSCode&category=Formatters&sortBy=Downloads>) to the Marketplace. We have seeded it with popular formatting extensions and invite formatter authors to add theirs as well. You can also use extension packs (/updates/v1_7#_extension-packs) to bundle a formatter extension with other extensions for your favorite language.

Summary

To summarize, an extension that implements the formatting extension API properly will do the following :

1. Register formatters via `registerDocumentFormattingEditProvider` (<https://github.com/microsoft/vscode/blob/a80f904d4e9afa7a05ae57a0123c11b727097129/src/vs/vscode.d.ts#L10089>).
2. Implement the formatting logic per the `DocumentFormattingEditProvider` interface (<https://github.com/microsoft/vscode/blob/a80f904d4e9afa7a05ae57a0123c11b727097129/src/vs/vscode.d.ts#L3425>).
3. Have a setting to enable / disable the formatter.
4. Add the "Formatters" category to the extension manifest.

We are not done with the feature work for formatters. "Format on Paste", "Format Files in Folder" and more are on our roadmap and we are happily awaiting more of your feedback and ideas.

#HappyCoding (<https://twitter.com/hashtag/HappyCoding?src=hash>)

Johannes Rieken, VS Code Team Member

@johannesrieken (<https://twitter.com/johannesrieken>)


IN THIS BLOG POST

The Formatting API

Multiple Formatters

Formatters in the Marketplace

Summary

 [Tweet\(https://twitter.com/intent/tweet?original_referer=https://code.visualstudio.com/blogs/2016/11/15/formatters-best-practices&ref_src=twsrc%5Etfw&text=Creating%20a%20Formatter%20Extension&tw_p=tweetbutton&url=https://code.visualstudio.com/blogs/2016/11/15/format-link-best-practices&via=code\)](https://twitter.com/intent/tweet?original_referer=https://code.visualstudio.com/blogs/2016/11/15/formatters-best-practices&ref_src=twsrc%5Etfw&text=Creating%20a%20Formatter%20Extension&tw_p=tweetbutton&url=https://code.visualstudio.com/blogs/2016/11/15/format-link-best-practices&via=code)

 [Subscribe\(/feed.xml\)](#)

 [Ask questions\(https://stackoverflow.com/questions/tagged/vscode\)](https://stackoverflow.com/questions/tagged/vscode)

 [Follow @code\(https://go.microsoft.com/fwlink/?LinkID=533687\)](https://go.microsoft.com/fwlink/?LinkID=533687)

 [Request features\(https://go.microsoft.com/fwlink/?LinkID=533482\)](https://go.microsoft.com/fwlink/?LinkID=533482)

 [Report issues\(https://www.github.com/Microsoft/vscode/issues\)](https://www.github.com/Microsoft/vscode/issues)

 [Watch videos\(https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w\)](https://www.youtube.com/channel/UCs5Y5_7XK8HLDX0SLNwkd3w)

Hello from Seattle. Follow @code (<https://go.microsoft.com/fwlink/?LinkID=533687>)

Star

137,219

Support (<https://support.serviceshub.microsoft.com/supportforbusiness/create?sapId=d66407ed-3967-b000-4cfb-2c318cad363d>)

Privacy (<https://privacy.microsoft.com/privacystatement>)

Terms of Use (<https://www.microsoft.com/legal/terms-of-use>)

License (/License)

 Microsoft (<https://www.microsoft.com>)

© 2022 Microsoft