Refactor code using GitHub Copilot Chat modes

6 minutes

GitHub Copilot can be used to refactor code in your codebase. You can use the Chat view modes to analyze, plan, and implement code refactoring tasks. The Chat view modes provide a user-friendly interface for managing chat conversations and accessing GitHub Copilot's features. The Chat view includes the following chat modes:

- **Ask mode**: Use the ask mode to ask questions about your codebase or technology concepts. You can use ask mode explain code, suggest revisions or fixes, or provide information related to the codebase.
- **Edit mode**: Use the edit mode to make edits across multiple files in your codebase. You can use edit mode to refactor code, add comments, or make other changes to your code.
- **Agent mode**: Use the agent mode to start an agentic coding workflow. You can use agent mode to run commands, execute code, or perform other tasks in your workspace.

> ⓘ **Important**
>
> When you use agent mode, GitHub Copilot may make multiple premium requests to complete a single task. Premium requests can be used by user-initiated prompts and follow-up actions Copilot takes on your behalf. The total premium requests used depends on the complexity of the task, the number of steps involved, and the model selected.

Use ask mode to refactor code

You can use ask mode to get help with coding tasks, understand tricky concepts, and improve your code. Ask mode is designed for interactive conversations with Copilot Chat. You can ask questions, get explanations, or request suggestions in real time.

1. Open the Chat view in ask mode.

2. Add context to the chat.

   The ask mode supports chat participants, slash command, and chat variables. You can add the `@workspace` chat participant or `#codebase` to the chat to provide context along

with files or folders.

3. Ask questions that help you understand the code that you want to refactor and the changes you want to make.

   For example, you might ask GitHub Copilot to help you understand your existing authentication code how to update the current method using OAuth.

4. Construct a prompt that describes the update that you want to implement.

   Your prompt should include a description of the code feature you want to implement. For example:

   ```plaintext
   @workspace I need to refactor the `EnumHelper` class and remove any code that uses reflection. Use static dictionaries to supply enum description attributes. Use a separate dictionary for each enum. The dictionaries should use values from the `LoanExtensionStatus.cs`, `LoanReturnStatus.cs`, and `MembershipRenewalStatus.cs` files. Explain how to update the EnumHelper class using dictionaries and show me the updated code.
   ```

5. Review the suggested code in the Chat view.

   The response displayed in the Chat view will include a code snippet that you can use to implement the feature. You can enter updated prompts to refine the code or ask for more details.

6. To implement suggested code, you can hover the mouse pointer over the code snippet and then select between the **Apply**, **Insert**, and **Copy** options.

   - **Apply**: Applies the code suggestion to the current file in the editor.
   - **Insert**: Inserts the code suggestion at the current cursor position in the editor.
   - **Copy**: Copies the code suggestion to the clipboard.

7. Test your refactored code to ensure it runs without errors and generates the expected result.

Use edit mode to refactor code

You can use edit mode when you want more granular control over the edits that Copilot proposes. In edit mode, you choose which files Copilot can make changes to, provide context to Copilot with each iteration, and decide whether or not to accept the suggested edits.

> **Tip**
>
> Use the ask mode to evaluate the code that you're interested in refactoring and evaluate your options before you start making changes. You can use edit mode to make the changes to your code.

1. Open the Chat view in edit mode.
2. Add context to the chat.

   The edit mode doesn't support chat participants. Specify workspace context using `#codebase` and by adding files to the chat.
3. Construct a prompt that describes the code feature that you want to implement.

   Your prompt should include a description of the code feature you want to implement. For example:

   ```plaintext
   #codebase I need to refactor the `EnumHelper` class and remove any code that uses reflection. Use static dictionaries to supply enum description attributes. Use a separate dictionary for each enum. The dictionaries should use values from the `LoanExtensionStatus.cs`, `LoanReturnStatus.cs`, and `MembershipRenewalStatus.cs` files.
   ```
4. Review the suggested edits in the code editor.
5. Accept or discard the suggested edits.

   You can navigate through the edits using the up and down arrows. You can Keep (accept) or Undo (discard) the suggested edits individually using the popup menu that appears over each edit. You can also accept or reject all of the edits at once using the **Keep** and **Undo** buttons at the bottom of the editor tab (or Chat view).
6. Test your refactored code to ensure it runs without errors and generates the expected result.

Use agent mode to refactor code

You can use agent mode when you want to automate the process of refactoring code. In agent mode, Copilot acts as an autonomous agent that can take actions on your behalf. You can ask

Copilot to perform specific tasks, and it generates code based on your requests.

In agent mode, you can use natural language to specify a high-level task, and let Copilot autonomously reason about the request, plan the work needed, and apply the changes to your codebase. Agent mode uses a combination of code editing and tool invocation to accomplish the task you specified. As it processes your request, it monitors the outcome of edits and tools, and iterates to resolve any issues that arise.

> **Important**
>
> When you use agent mode, GitHub Copilot may make multiple premium requests to complete a single task. Premium requests can be used by user-initiated prompts and follow-up actions Copilot takes on your behalf. The total premium requests used depends on the complexity of the task, the number of steps involved, and the model selected.

1. Open the Chat view in agent mode.
2. Construct a prompt that describes the task you want GitHub Copilot to perform.

   Suppose you have a code project that uses two or more processes/techniques to accomplish the same task. You can ask GitHub Copilot to refactor you code using a single approach to improve consistency, maintainability, and performance.

   For example:

   plaintext

   ```
   Review the LINQ code used in the JsonData and JsonLoanRepository
   classes. Refactor the methods in the JsonPatronRepository class using
   LINQ queries. Ensure that existing code functionality is maintained.
   ```

3. Agent mode might invoke multiple tools to accomplish different tasks. Optionally, select the Tools icon to configure which tools can be used for responding to your request.
4. Confirm tool invocations and terminal commands.

   Before Copilot runs a terminal command or a tool that isn't built in, it requests confirmation to continue. Confirmation is required because tools might run locally on your machine and perform actions that modify files or data.

5. Copilot detects issues and problems in code edits and terminal commands and will iterate and perform actions to resolve them.

> **Important**
>
> Although Copilot is capable of self-healing, it might not always be able to resolve issues on its own. You can pause the process and provide more context to help Copilot understand the problem. You can also cancel the process and start over with a new prompt.

6. Review the suggested edits and accept or discard the suggested edits.
7. Test your refactored code to ensure it generates the expected result.

   Agent mode should resolve issues one its own, but you should still test the code to ensure it works as expected.

   You can revert edits made by agent mode using the **Undo** button in the Chat view. You can also use the **Revert** option in the editor to revert changes made by agent mode.

When to choose agent mode over edit mode

Consider the following criteria to choose between edit mode and agent mode:

- Edit scope: Agent mode autonomously determines the relevant context and files to edit. In edit mode, you need to specify the context yourself.
- Task complexity: Agent mode is better suited for complex tasks that require both code edits and the invocation of tools or terminal commands.
- Duration: Agent mode uses multiple steps to process a request, so it might take longer to generate a response. For example, to determine the relevant context and files to edit, determine the plan of action, and more.
- Self-healing: Agent mode evaluates the outcome of the generated edits and might iterate multiple times to resolve intermediate issues.
- Request quota: In Agent mode, depending on the complexity of the task, one prompt might result in many requests to the backend.

Summary

GitHub Copilot can be used to refactor code in your codebase. You can use the Chat view modes to analyze, plan, and implement code refactoring tasks. The Chat view includes three chat modes: Ask mode, Edit mode, and Agent mode. You can use the ask mode to ask questions about your codebase or technology concepts. You can use the edit mode to make edits across multiple files in your codebase. You can use the agent mode to start an agentic coding workflow.

Next unit: Examine the GitHub Copilot code review features

Previous Next