Create unit tests using Chat view modes

6 minutes

The Chat view in Visual Studio Code provides three modes that can be used to create unit tests: Ask, Edit, and Agent. Each mode has its own strengths and weaknesses, and the best mode to use depends on the specific task at hand.

- The ask mode is optimized for asking questions about your code projects, coding topics, and general technology concepts.
- The edit mode is optimized for making edits across multiple files in your codebase.
- The agent mode is optimized for starting an agentic coding workflow.

> **Important**
>
> When you use the Chat view in 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 the ask mode to create unit tests

The ask mode can be used to analyze a workspace and then create unit tests. The ask mode is useful when you want to create tests for multiple functions or methods in a file, or when you want to create tests for an entire file.

To create unit tests using the ask mode, follow these steps:

1. Open the file that contains the code you want to test.
2. Open the Chat view and start a new chat session using the ask mode.
3. Add context to the chat session.
    - You can add context to the chat session by dragging and dropping files from Visual Studio Code's EXPLORER view into the Chat view. You can also use the **Add Context** button.

- You can open external files in the code editor to include resources that aren't part of the workspace and use them to provide specific context. For example, you can open markdown files that contain contributor guidelines or contact information and then use the **Add Context** button to add them to the Chat view context.
- You can use the `@workspace` chat variable to specify the workspace as part of your prompt. The workspace context is useful when you want to create tests for multiple functions or methods in a file, or when you want to create tests for an entire file.

4. Enter a prompt that asks for unit tests for the code in the file.

    - For example: "`@workspace /explain I need to create unit tests for the code in this file. The tests should be written in Python and use the unittest framework.`"

5. Review the suggested unit tests, and refine the results using updated prompts if necessary.

6. Move the suggested unit tests into a test file.

    - For example, create a test file in the same directory as the code file, and then insert the suggested unit tests into the file.
    - You can use the ask mode to suggest updates for specific tests after creating the test file, or use other GitHub Copilot tools to help with updates.
    - You can also use the **Apply in Editor** button to apply the suggested unit tests directly to the code file.

7. Save the test file.

    - Test files are typically saved to a separate "tests" directory in a project that's configured for unit tests. Your options depend on your project's structure and testing framework.
    - You can use the ask mode to suggest updates for specific tests after creating the test file, or use other GitHub Copilot tools to help with updates.

8. Run the tests to ensure they pass and verify the functionality of your code.

9. If necessary, refine the tests by adding more test cases or modifying existing ones.

10. Save the file again after making any changes to the tests.

Use the edit mode to create unit tests

The edit mode can be used to create unit tests by adding context files to the chat and then creating or updating test files. The edit mode is useful when you want to create tests for

specific functions or methods in a file, or when you want to create tests for an entire file.

To create unit tests using the edit mode, follow these steps:

1. Open the file that contains the code you want to test.
2. Open the Chat view and start a new chat session using the edit mode.
3. Add context to the chat session.
    - Chat participants aren't available in edit mode, so you can't specify `@workspace` as part of your prompt. However, you can add context to the chat session using `#codebase` and by dragging and dropping files or folders from Visual Studio Code's EXPLORER view into the Chat view. Use Visual Studio Code to open external files, such markdown files that contain contributor guidelines, and then use the **Add Context** button to add them to the chat context.
4. Enter a prompt to create the intended unit tests.
    - For example: "`I need to create unit tests for the code in this file. The tests should be written in Python and use the unittest framework. Create a test file in the same directory as the code file.`"
5. Review the test file created using edit mode, and then save or discard the file.
    - You can update the file using new prompts to correct or enhance specific tests if necessary.
6. Save the test file.
    - Test files are typically saved to a separate "tests" directory in a project that's configured for unit tests. Your options depend on your project's structure and testing framework.
7. Run the tests to ensure they pass and verify the functionality of your code.
8. If necessary, refine the tests by adding more test cases or modifying existing ones.
9. Save the file again after making any changes to the tests.

Use the agent mode to create unit tests

The agent mode can be used to automate tasks within your unit testing process. For example, you can use the agent mode to scaffold a test project, create test files, run tests, generate test

reports, or perform other tasks related to unit testing. The agent mode is best for creating unit tests that require a more in-depth understanding of the project.

To create unit tests using the agent mode, follow these steps:

1. Open the file that contains the code you want to test.
2. Open the Chat view and start a new chat session using the **Agent** mode.
3. Let agent mode determine the context.

   In agent mode, you don't need to specify the context. Copilot will automatically determine the relevant context and files to edit.
4. Optionally, select the Tools icon to configure which tools can be used for responding to your request.

   - You can select the tools that you want to use for responding to your request. For example, you can select the **Test Explorer** tool to run tests or the **Terminal** tool to run commands.
   - You can also select the **GitHub Copilot** tool to use Copilot's code generation capabilities.
5. Enter a prompt that defines the intended tasks.

   - For example: "`Ensure that a suitable unit tests project is prepared for the selected code file. Create a test file in the unit test project that includes unit tests for all methods in the selected file. Unit tests should be written in C# and use the xUnit framework. Run the tests to ensure expected results.`"
6. Monitor the progress of the agent mode as it performs the tasks.

   - Confirm tool invocations and terminal commands. You can confirm or reject the tool invocations and terminal commands that agent mode suggests. For example, you can confirm the command to run the tests or the command to generate a test report.
   - Interrupt the agent mode if necessary. You can interrupt the agent mode if you want to stop the tasks that it's performing. For example, you can interrupt the agent mode if you want to change the context or if you want to change the tools that are being used.
7. Review the files that agent mode created or updated during the specified tasks, and then keep or discard updates.

- You can use new prompts to correct or enhance specific tests if necessary.

Summary

GitHub Copilot's Chat view provides three modes that can be used to create unit tests: Ask, Edit, and Agent. Each mode has its own strengths and weaknesses, and the best mode to use depends on the specific task at hand. The ask mode is optimized for asking questions about your code projects, coding topics, and general technology concepts. The edit mode is optimized for making edits across multiple files in your codebase. The agent mode is optimized for starting an agentic coding workflow.

Next unit: Exercise - Develop unit tests using GitHub Copilot

Previous Next