Blink

Team Ray-Bens 🕶

Ray Grant
Benjamin Roach
Benjamin Shapiro
Bradley Walters

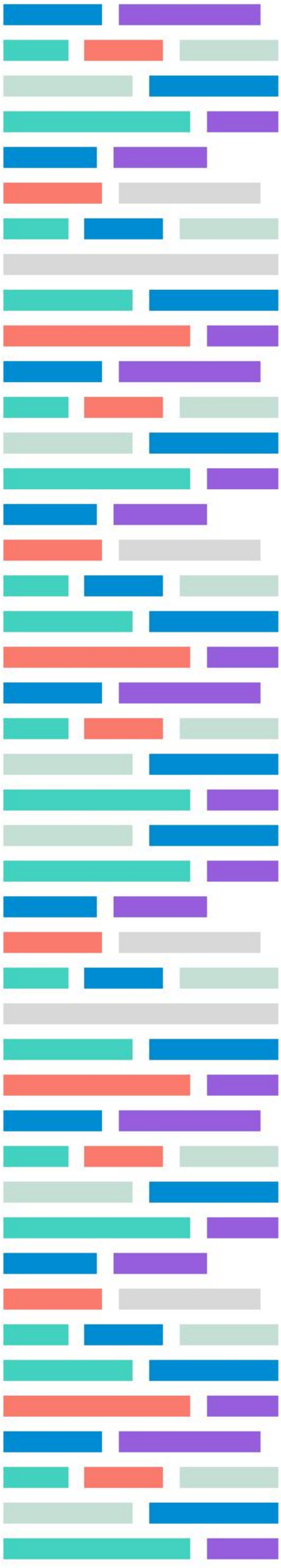


Table Of Contents

1. Executive Summary

The ratio of time developers spend reading code to writing code is higher than ten to one, yet precious little has been done within modern text editors to enable users to quickly navigate between related pieces of code. It is largely up to developers to organize their code sequentially within files. Modern text editors often ship with features such as "go to definition" and "list usages", but they're bolted onto a fundamentally file-based navigation scheme, and are often slow to use as a result.

Blink is a next-generation source code editor that breaks from these shackles, improving software developer speed and efficiency as a result. Rather than navigating a project by scrolling through or switching files, the user moves around based on the relationships between functions in the code itself. While the user is editing one function, previews of its callers and callees are displayed nearby. Selecting any one of these functions moves it to the focused slot, and refreshes the previews of related functions. By empowering the user to always work within logically grouped functions, even when exploring their entire codebase, Blink greatly reduces wasted time spent scrolling, searching for functions, and navigating their project.

Features/Components

- Editor Pane - Editable code pane with syntax highlighting and common shortcuts.
- Preview Panes - Previews of code related to what currently occupies the Editor Pane.
- Language Server Client - Retrieves semantic program information necessary to populate the Preview Panes and Editor Pane.
- Keyboard and Mouse Navigation - Shortcuts to swap code from the Preview Panes to the Editor Pane.
- Contexts - Groupings of related code, mapped to files behind the scenes.
- Code Creation - Adding new code to a context.
- Alternative Navigation - Jump to a function by name or by project structure to enter a distant area of the project.
- User Guide - Explanatory documentation and intro to unique editor features.
- Installer - Installer programs for the various supported operating systems.

2. Background

Although there are numerous text editors and IDEs available to developers, most are built around a file-based navigation system. This can be very inefficient and time consuming because the relevant code will need to be searched for in sometimes enormous files, and the search facilities text editors have can be lacking when it comes to addressing this inefficiency.

Our text editor facilitates the reading and writing of code by abstracting away the files in a text editor and by having it by default show three dependent functions and the three dependee functions of the function that the user is currently editing. Without worrying about files, the programmer can spend less time navigating large files. The target user is a Python developer who is looking for a fast, convenient, and powerful text editor that is fundamentally different than what they are accustomed to. Blink would also be useful for developers trying to understand an existing large project or for reverse engineering purposes, as relational navigation becomes easier to follow than file-based navigation.

Similar Ideas

Traditional editors like Atom, VSCode, and Sublime are common editors that includes many features for customizing the users experience via plugins. However, their navigating systems are primarily file based and thus force the user to lose context while navigating.
Vim is a text editor based on the Vi text editor. It is extensible and makes use of keyboard shortcuts, composite commands, and macros to try to save the programmer's time. By leveraging these features, experienced Vim users can navigate within their project extremely quickly. However, this navigation is always between locations within files, rather than program semantics and relationships between pieces of code.
Jetbrains IDEs and others contain more powerful features than simple text editors such as debugging, refactoring, and code analysis for many languages. Jetbrains additionally exposes several useful navigation features such as "Jump to Definition," "List Usages," and function implementation previews on hover. Our "Preview Panes" tackle similar problems as these features, but go far beyond them by being always-present and serving as the main method of navigation within a project.

Xcode is a text editor for macOS with common file-based navigation features, as well as a unique "Assistant Editor" feature. The "Assistant Editor" is an additional editor pane that automatically displays a relevant file in a few specific cases. For example, when editing a class implementation, the assistant editor may automatically display the corresponding header file. Our editor provides a superset of this experience; any related code will be displayed as a preview within the editor window, rather than just in specific scenarios.

Required Technology

Electron is a framework for easily developing cross-platform (Linux, macOS, Windows) apps using web technology. We will use Electron as the framework for our editor.
CodeMirror is a JavaScript library text editor implementation including syntax highlighting, line numbering, and keyboard shortcuts. We will use CodeMirror as a foundation for our text editor UI component.

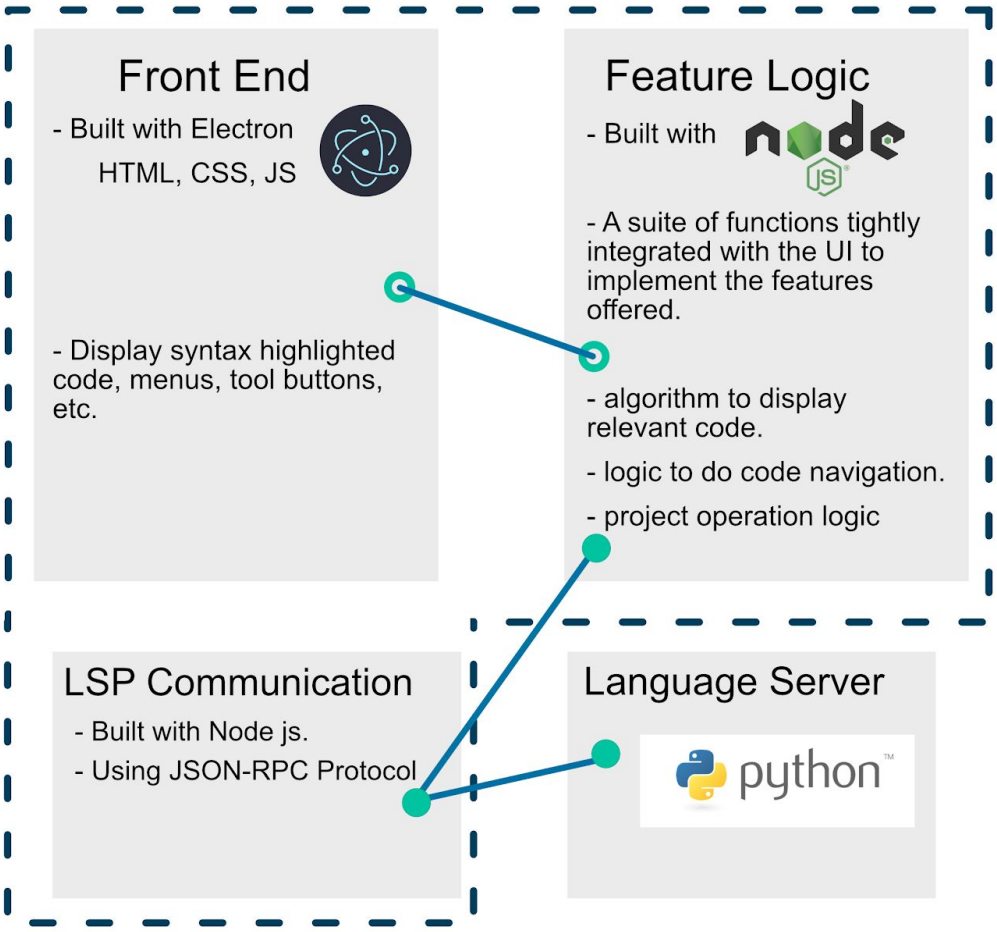
The Language Server Protocol is an open standard for text editors to communicate with a local server that exposes semantic information for code written in a particular language. Blink will utilize this protocol and its implementations to perform static code analysis and find relevant functions to display as previews.

Software Requirements

- Linux, MacOS, or Windows (any version supported by Electron)
- Our Software
- Python 3.7+

3. Requirements Analysis

System Architecture



Our program can be split into four major functional areas:

- Electron frontend: This is the UI of our application, consisting of the text editor view, preview panes, user-facing menus, and other elements. Every part of the UI will be built in the Electron framework and will have no critical logic, but will work tightly with the Feature Logic module to update it's elements.
- Feature Logic: This consists of the code within the electron frontend that handles changes and events received from the language server. Within this module, every change within the UI will be analyzed and determined whether to update some data structures, such as the navigation graph, or just to pass these updates to the language server. This module will also make appropriate updates to backend data structures and relevant UI elements based on responses from the language server.
- LSP client: The LSP client lies within the Electron frontend and sends updates and performs operations using JSON-RPC messages sent to the LSP server.
- LSP server: This is a separate process that implements the Language Server Protocol which handles parsing and language-aware requests from the LSP frontend, such as finding line numbers of functions within a file and syntax highlighting.

Personnel

- Ray Grant — Ray has extensive experience interfacing with backend APIs and has substantial interest with network communications and protocols. He will be responsible for:
    - LSP-Provided Editor Features: autocomplete, errors, jump to definition.
    - Symbol Search for code finding, allowing the user to search for code in different scopes such as primary pain, secondary pains, entire context.
    - Keyboard shortcuts for all basic editing and navigation operations.
    - Support for top level code viewing and editing of any given piece of code.
    - Backend File and Context Management - Implementing CRUD operations on files/contexts.
    - Rename Symbol: changing the name of variable/object within a given context. (Advanced)
- Ben Roach — Ben has experience with parsing and databases, and an interest in programming languages. He will be responsible for:
    - Retrieving Locations from Language Server.
    - Maintaining Dependency Graph of Code Relationships.
    - Backend for Navigation via Preview Panes.
    - Backend for Project Structure Pane.
    - Execution of Code from the Editor. (Advanced)
    - TODO Pane (Advanced)
- Bradley Walters — Bradley has built web and mobile applications in several frameworks, and has a strong interest in programming languages and compilers. He will be responsible for:
    - Language Server Initialization and Communication Framework.
    - Language Server Synchronization.
    - Frontend File and Context Management.
    - Scoped Search. (Advanced)
- Ben Shapiro — Ben has a background in web-based technologies, programming language (PL) theory, API design, and database development. He will be responsible for:
    - Text Editor Pane.
    - Project structure graph view front-end.
    - Frontend for Navigation via Preview Panes.
    - Pin and Unpin Preview Panes.
    - Keyboard Shortcuts.
    - Debug code from the editor.
    - Syntax highlighting from the language server.

System Features

Basic

- Text Editor Pane — Editing code in the active editor pane with syntax highlighting and common keyboard shortcuts.
- Communication with a Language Server — Synchronizing the view of the project with a process implementing the Language Server Protocol, and retrieving information from it such as function and class locations within source code files.
- Preview Panes — Populating preview panes with function bodies related to the function in the active editor window, with the ability to page through panes.
- Navigation via Preview Pane — Swap a function that is previewed in a preview pane to the active editor pane.
- Create, Open, and Save Project — Read projects created by Blink from disk and write them back to disk.
- Mapping Contexts to Files — Mapping contexts and context groups to files and folders, containing the code that has been edited by blink.
- Supporting Top Level Code — View imports and top level code for a given context.

Planned

- View Project Structure — View the logical structure of the project, as a hierarchy of contexts and items within contexts.
- Pin and Unpin Preview Panes — Pin the contents of a preview pane to keep it around when that code is no longer related directly to the code in the active editor.
- Symbol Search — Search either the whole project or the code currently in the main pane, for functions, classes, and other items by name.
- Keyboard Shortcuts — Control the navigation process and other editor features with custom keyboard shortcuts.
- User Guide / Getting Started — Timely documentation and information to help a user learn how to use the program, such as when creating a new project.
- Installer — Installer program for Linux, macOS, Windows.

Advanced

- Scoped Search — Broaden keyword search from the code in the main pane to the code in related functions. Also broaden scope to context and context groups
- Rename Symbol — The user can right click on a symbol such as a variable, edit it, and it changes everywhere in the project.
- Execute code from the editor — Allow Python interpreter to work within Blink, letting users get output quicker, and iterate on the output more quickly.
- TODO Pane — A Pane that that shows all the TODO comments in the users project. This list is automatically updated as TODOs are created and deleted.
- Debug code from the editor — Debug code within the editor, with breakpoints, stepping through/over code, variable previews, and other features.
- Syntax highlighting from the language server — Communication with the Python Language Server to get syntax highlighting information to use in highlighting/syntax coloring our code, as opposed to the basic regex-based highlighting in CodeMirror.

4. Timeline

Phase	Ray Grant	Ben Roach	Ben Shapiro	Bradley Walters
Alpha: Multiple Files and Project Structure	Support saving active contexts to multiple files. Support classes properly. Integrate basic shortcut keys. Refactor for collaborative code changes.	Support multiple contexts within symbol graph. Turn symbol graph into linear list of functions for saving files. View project structure.	Create context management UI in panes. Create project structure UI and data structure. Associate preview panes to contexts with navigation.	Apply changes in the editor window to the correct context. Support extracting and displaying top-level code for a context (file) in panes. Support global variables properly.
Beta: QoL	Preview pane paging. Integrate features for quicker and easier python development for users.	Jump to symbol by name. Rename symbol within workspace. TODO pane.	Pinning and unpinning preview panes. Refactor UI. Scoped find and replace.	Investigate changes required to support multiple languages. Multiple editor window support.
Production: Distribution and Advanced Features	Polishing controller code. Syntax highlighting from the language server	Polishing data structure code. Running and debugging code from the editor.	Installer. Resizing panes, swapping code into and between panes.	User Guide. Scoped find and replace. Documentation in preview panes.

5. Use Cases

Title	Use Case #1: Create a New Project
Description	The user will create a new project, selecting its name and the location it should be saved in.
List of Steps	1. Navigate to the “File > New Project” menu option.

	2. A standard file browser prompt appears. 3. Select where the project should be created, and the name of the project file to create. 4. Click the Create button. 5. A new, empty project will be opened in the editor. The active editor pane and preview panes will be empty.
Related UI	Figure 8 - File Menu, See Figure 9 - Create Context

Title	Use Case #2: Open an Existing Project
Description	The user will open an existing project by selecting it from a file browser within the app.
List of Steps	1. Navigate to the “File > Open Project” menu option, or press ctrl+o. 2. A standard file browser prompt appears. 3. Select which file should be opened. 4. Click the Open button. 5. The selected project will be opened in the editor.
Related UI	Figure 8 - File Menu

Title	Use Case #3: Save a Project
Description	The user will save a whole project to disk within the editor.
List of Steps	1. Navigate to the “File > Save All” menu option, or press ctrl+s. 2. The project being edited will be saved in the location it currently resides.
Related UI	Figure 8 - File Menu (save button on the left bar and in file menu)

Title	Use Case #4: Create New Code
Description	The user will create new code within a given context.
List of Steps	1. Press the “New Pane” button, or press ctrl+n. 2. Select a context (group for code) from a list of already created contexts, or create a new context and name it. 3. The active editor pane will become an empty pane. 4. The previously active editor pane, if any, will become pinned in the first available preview pane position. 5. After editing code in the active pane and navigating away from it (by creating a new pane or navigating to a preview pane), the context

	for the active pane will be re-analyzed, with its code sorted topologically and placed within the file backing said context.
Related UI	Figure 2 - Select Context, Figure 1 - Standard Edit View (left plus button)

Title	Use Case #5: Edit Code
Description	The user will edit code in the active editor pane, with the preview panes updating accordingly.
List of Steps	1. While the active editor pane is selected, the user can modify its contents as desired, with standard editing tools. 2. The preview panes will initially contain code related to the code in the active editor pane. 3. As code is added or removed, syntax will be highlighted. 4. As new code cross-references are created or removed, the preview panes will be updated accordingly.
Related UI	Figure 1 - Standard Edit View

Title	Use Case #6: Navigate to Related Code
Description	The user will navigate to a function or object that is related to the one currently being edited. For example, the target function may be a caller or callee of the function being edited.
List of Steps	1. Locate the desired code in one of the preview panes. This may involve paging through the preview panes. 2. Click on the pane to swap the targeted preview pane to the active editing pane, or press ctrl+# where # is the pane's numeric index.
Related UI	Figure 4 - Switch Pane

Title	Use Case #7: Edit Top Level Code/Imports
Description	The user will view the top level constants, code, and import statements for a given context.
List of Steps	1. In the active editor pane, select the "View Top Level" button. 2. The active editor pane will become the special "top level" pane for the context of the previously active editor pane. 3. The previously active editor pane, if any, will become pinned in the first available preview pane position. 4. The user can add top-level code, constants, or import statements that are in scope for the given constants. 5. After editing top level code in the active pane and navigating away from it, the context for the active pane will be re-analyzed and placed within the file backing said context.
Related UI	Figure 5 - Import & Top Level View

Title	Use Case #8: Pin and Unpin Preview Panes
Description	The user will pin a preview pane to prevent its contents from being altered by navigation within the editor.
List of Steps	1. Press the "pin" button within the preview pane that should be pinned, or press alt+# where # is the pane's numeric index. 2. The preview pane will change visually to indicate that it is pinned. 3. Editor navigation that alters the preview panes will not alter pinned preview panes. 4. The "pin" button, when pressed again, will unpin the preview pane. Navigating to a pinned preview pane will also unpin it.

Related UI	See top right of small panes in Figure 1 - Standard Edit View

Title	Use Case #9: Navigate to Code by Name
Description	The user will jump to a piece of code by name.
List of Steps	1. Press the “Jump by Name” button, or press ctrl+j. 2. Enter the name of a top-level function, class, or variable and submit. 3. The active editor pane will switch to contain the chosen code. 4. The preview panes (aside from pinned panes) will update to contain code relevant to the newly chosen code.
Related UI	See Figure 7 - Search View

Title	Use Case #10: View Project Structure
Description	The user will view the overall hierarchy of the project, and be able to jump to any function in the project.
List of Steps	1. Press the “View Project Structure” button. 2. A list of contexts within the project, as well as a list of code within each context will be displayed. 3. Select code within a context to switch the active editor pane to the chosen code (and update the preview panes as described). 4. Select a context to rename it.
Related UI	See Figure 6 - Context View

Tier 3 Use Cases

Title	Use Case #11: Find/Find and Replace Text
Description	The user will find all instances of a particular string, and can optionally replace them with a new string.
List of Steps	1. Navigate to “Edit > Find” or “Edit > Find and Replace”, or press ctrl+f or ctrl+h, respectively. 2. Enter the search term and select the scope of the search (active editor pane, active + preview panes, or entire project). 3. Tab through all found instances of the search term. 4. If Find and Replace was selected, enter a new term to replace the search term, and select the scope of the operation to perform it (replace one, all, etc).
Related UI	Figure 7 - Find View,

Title	Use Case #12: Renaming a Variable or Function
Description	The user will rename a variable or function such that everywhere the variable or function is used, the name changes accordingly.
List of Steps	1. While a variable or function name is selected, select “Rename” within the context menu, or press ctrl+r. 2. A prompt will appear, with space for a new name to be entered. 3. Type the new name in the box and click rename, or press enter. 4. The variable or function will be renamed everywhere it is referenced.
Related UI	Figure 11 - Rename Variable/Function

Title	Use Case #13: Executing Code from the Editor
Description	The user will execute the project from within the editor.
List of Steps	1. Navigate to the “Run” menu option, or press F5. 2. The currently open project will be compiled if necessary using predefined options, then executed.
Related UI	

Title	Use Case #14: Debugging Code from the Editor
Description	The user will debug running code in the editor’s environment, with breakpoints, value previews, stack traces, and other basic debugging tools.
List of Steps	1. A breakpoint can be added at the current line by selecting “Breakpoint” in the context menu, or pressing ctrl+b. 2. When the code is executed within the editor, breakpoints will cause execution to pause. When this happens, the user can view the values of variables and step through code. 3. If an exception is thrown, a stack trace will be displayed, and the offending line will be highlighted within the editor until execution is halted.
Related UI	

Title	Use Case #15: Viewing/Adding TODOs
Description	The user can view, add and check off todos in this pane.
List of Steps	3. Click bottom TODOs button to convert bottom pane to TODO pane. 4. TODO pane will display any comments in the project that begin with “TODO” 5. Clicking on an item will jump to that comment in the project.
Related UI	Figure 10 - TODO Pane

6. Appendix

UI Sketches

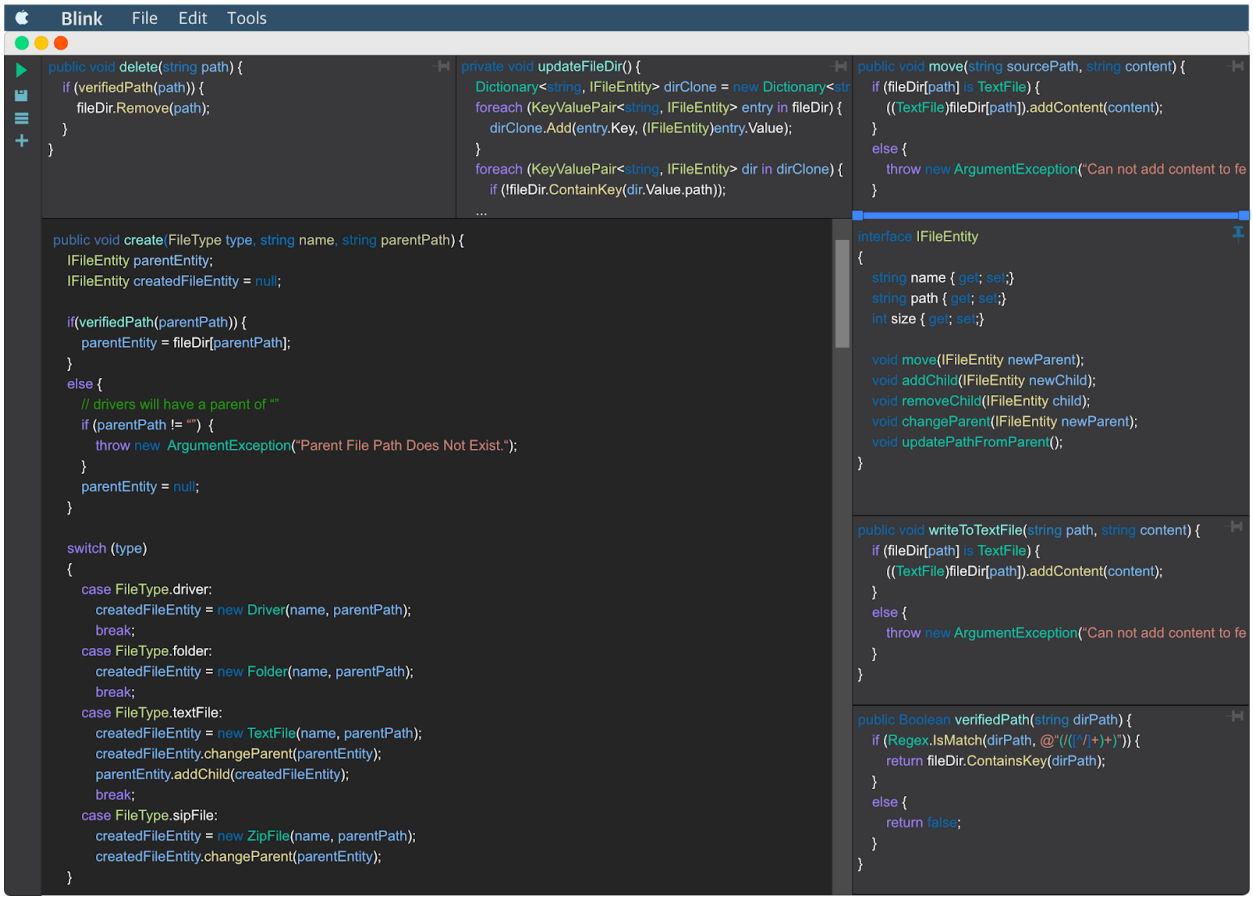


Figure 1 - Standard Edit View

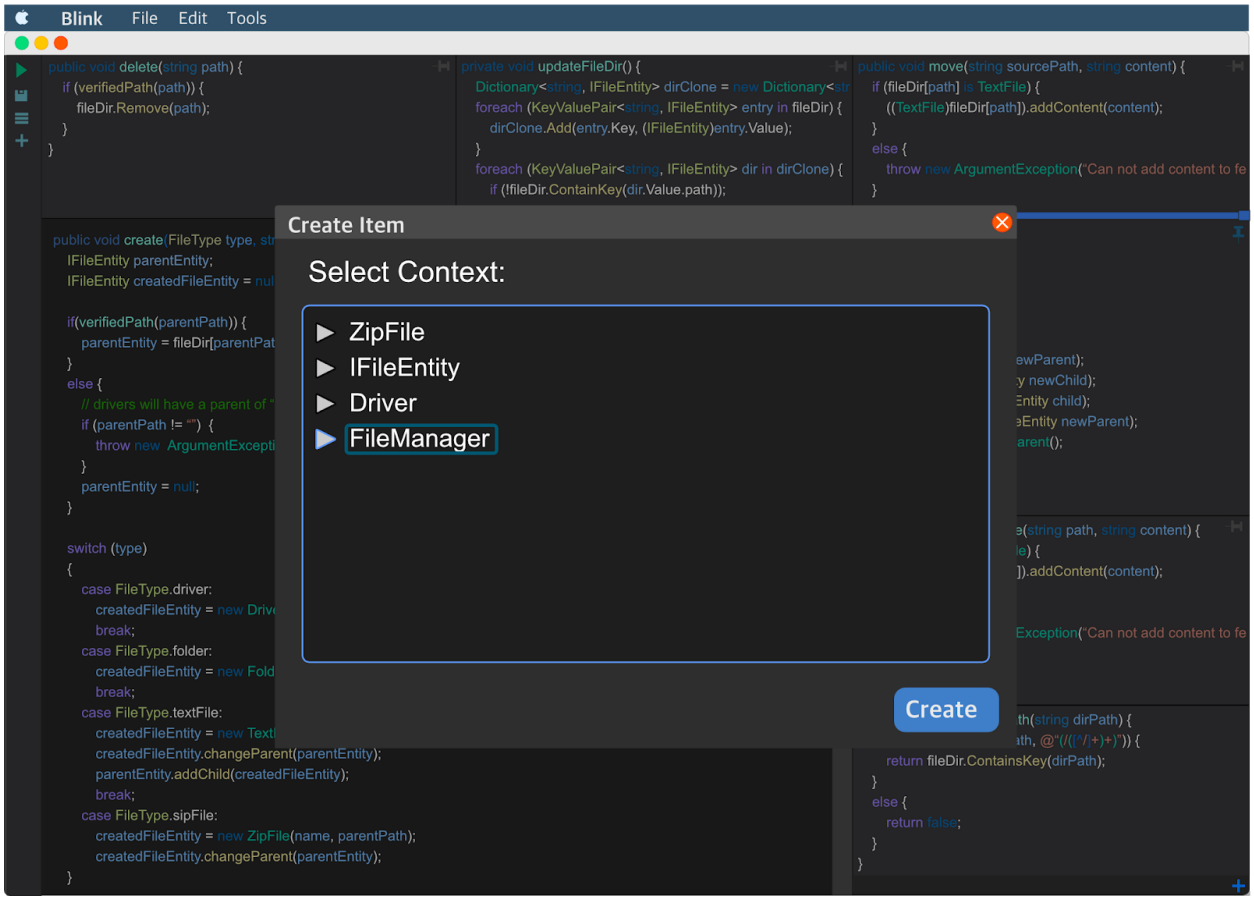


Figure 2 - Select Context

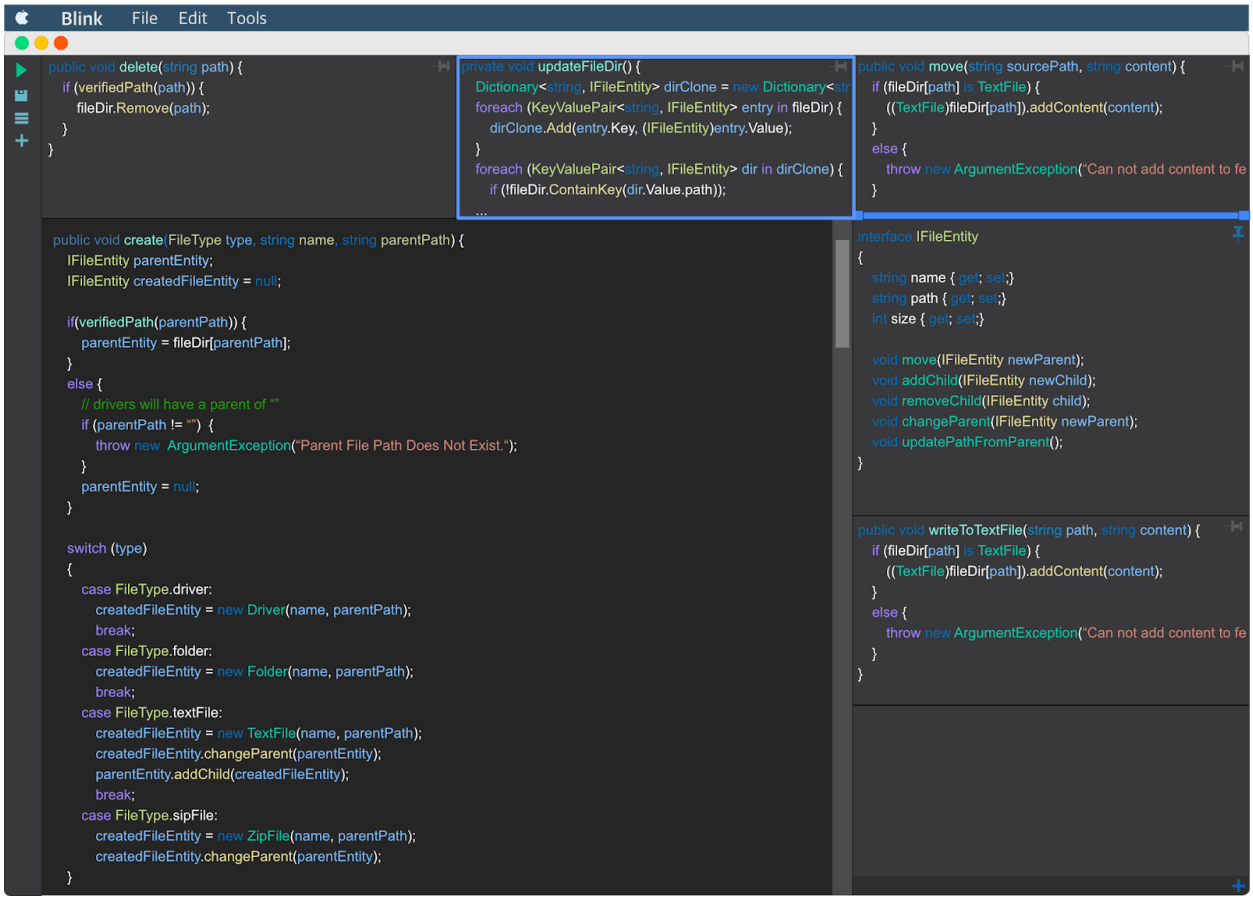


Figure 3 - Select Pane

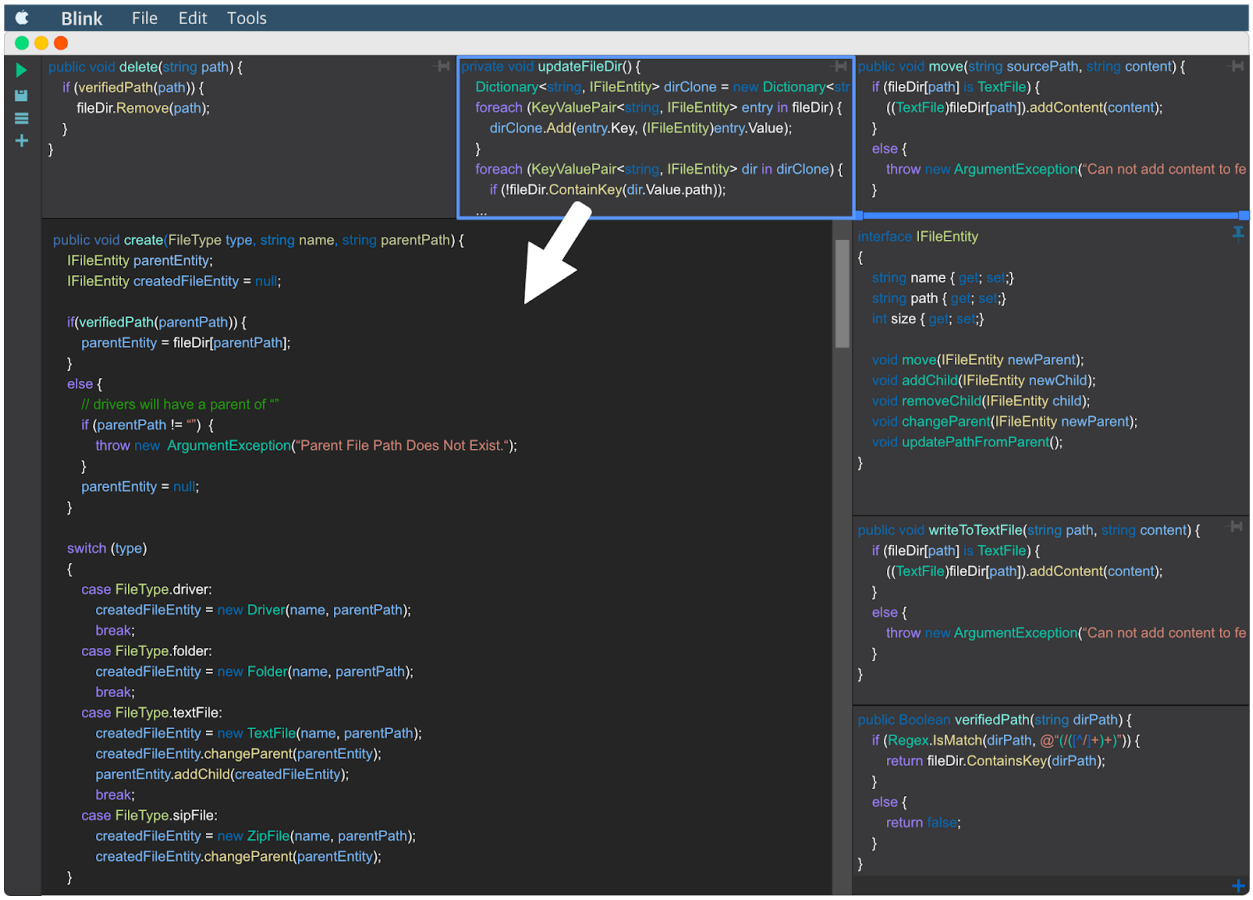


Figure 4 - Switch Pane

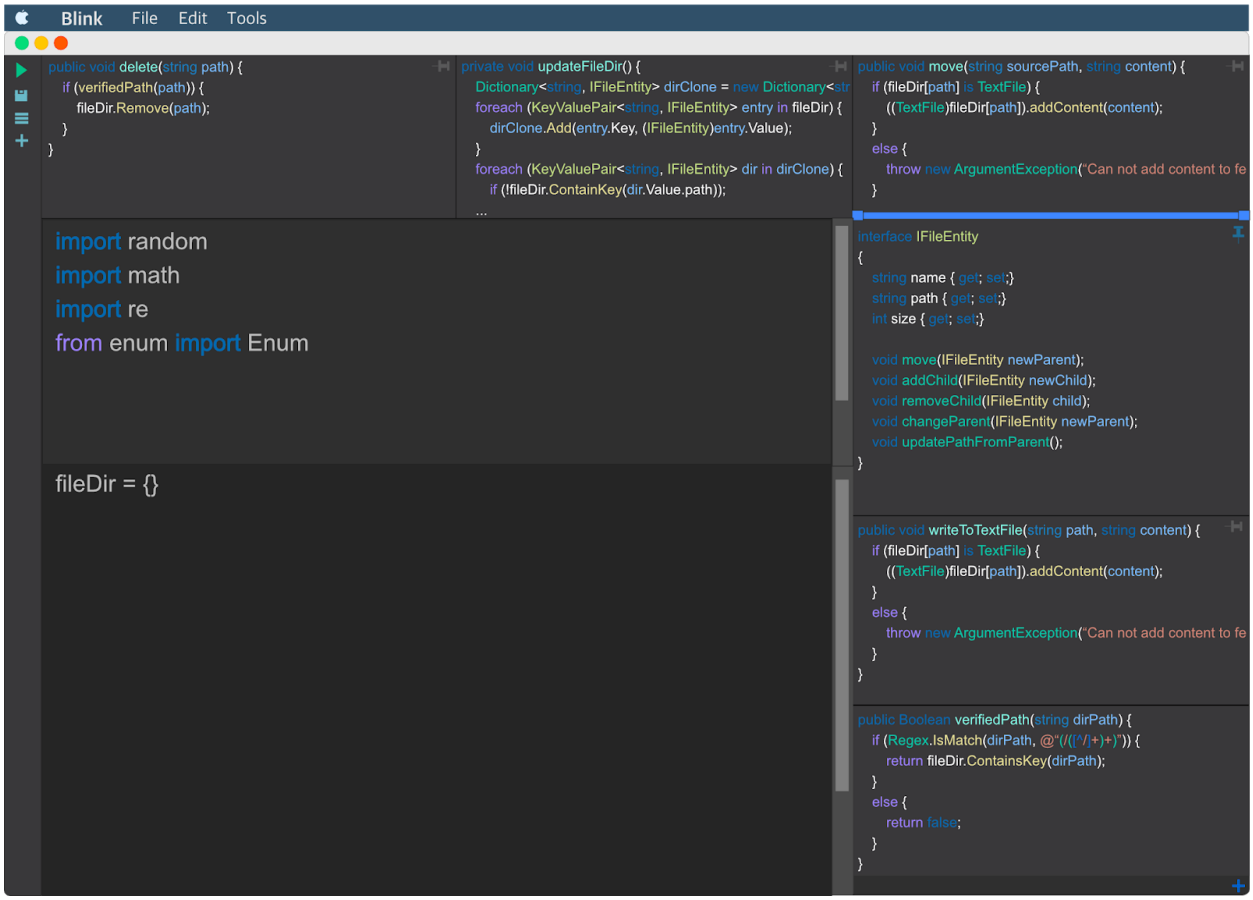


Figure 5 - Import & Top Level View

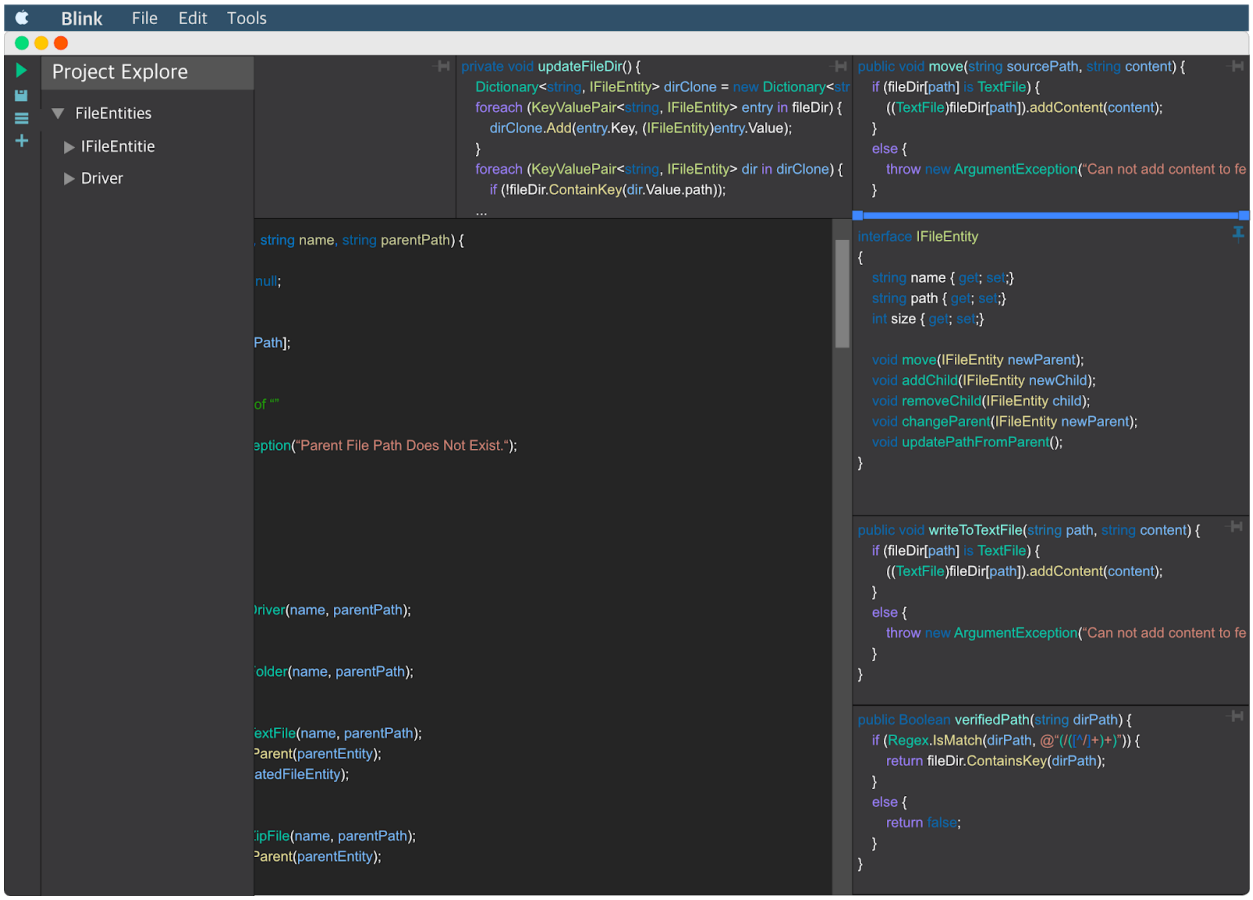


Figure 6 - Context View

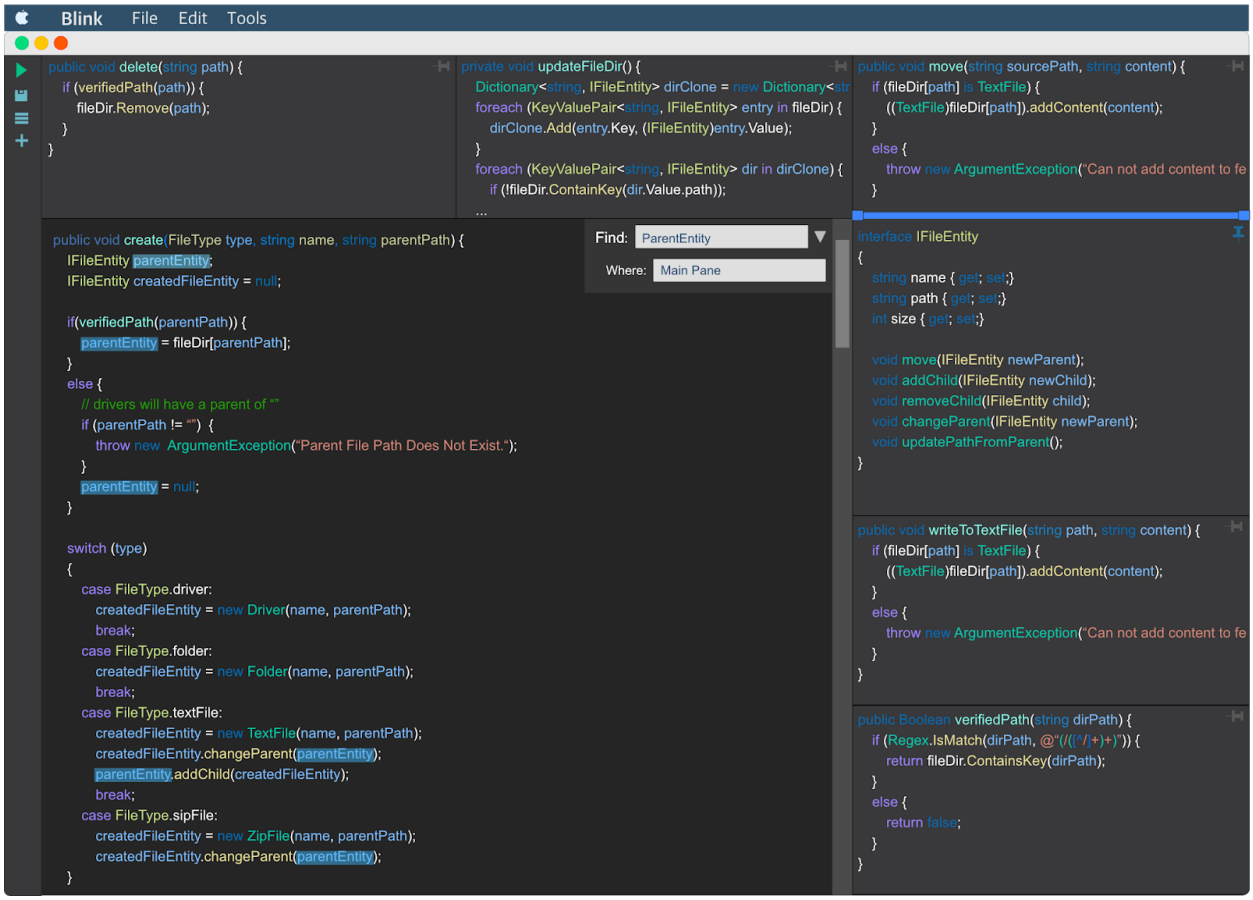


Figure 7 - Find View

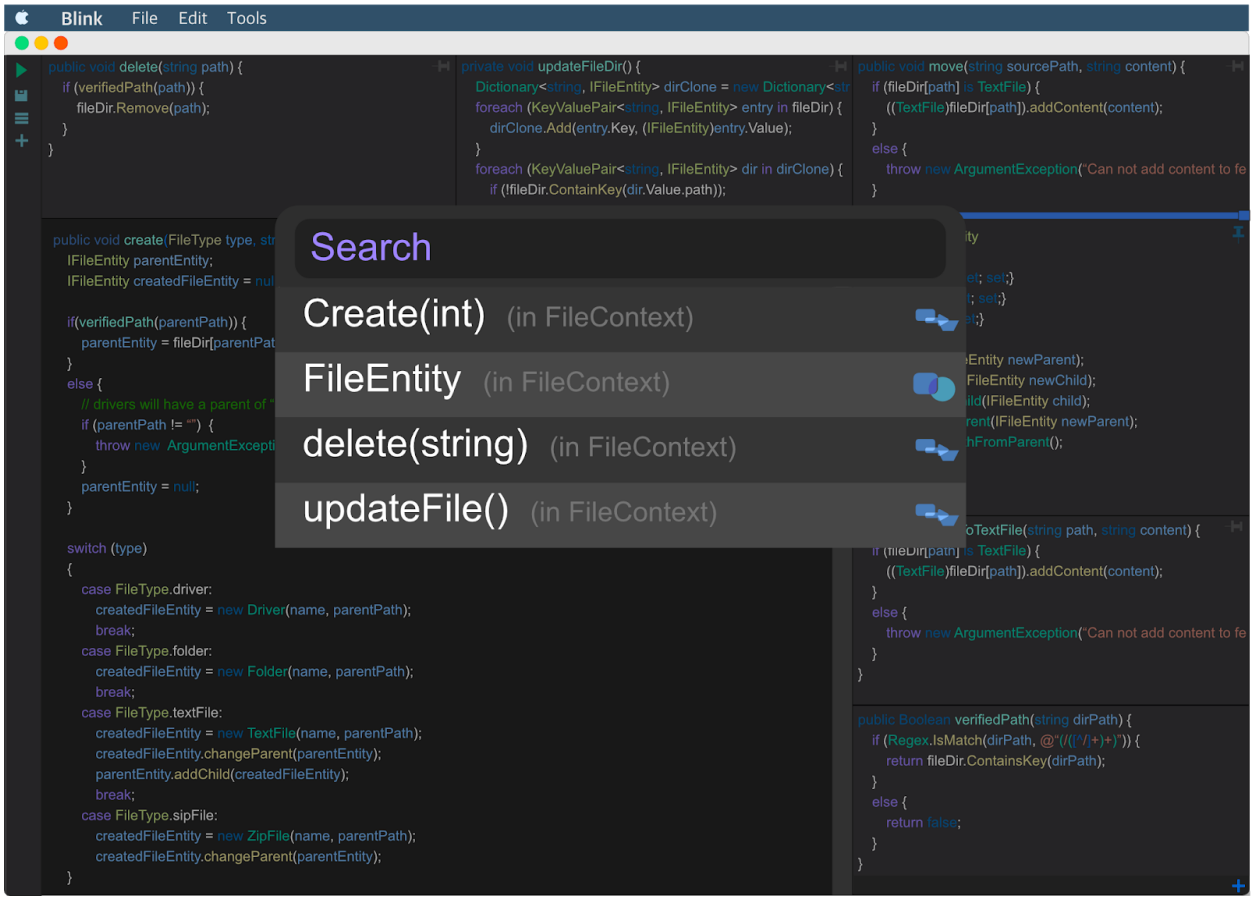


Figure 7 - Search View

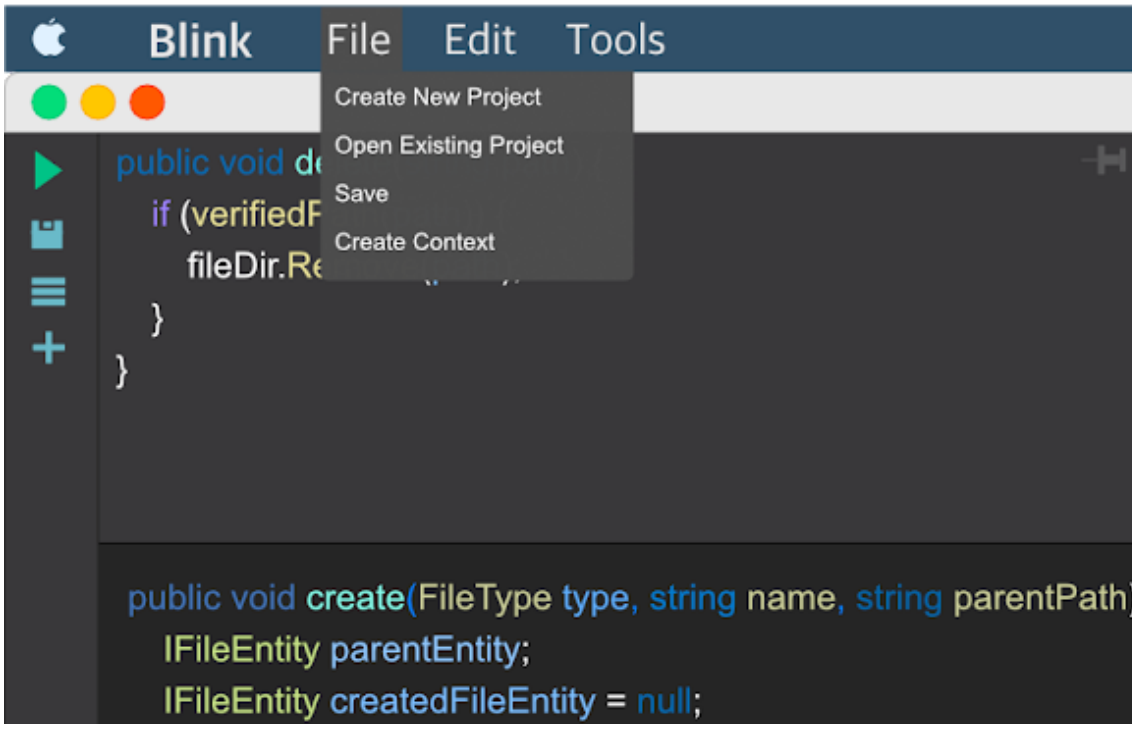


Figure 8 - File Menu

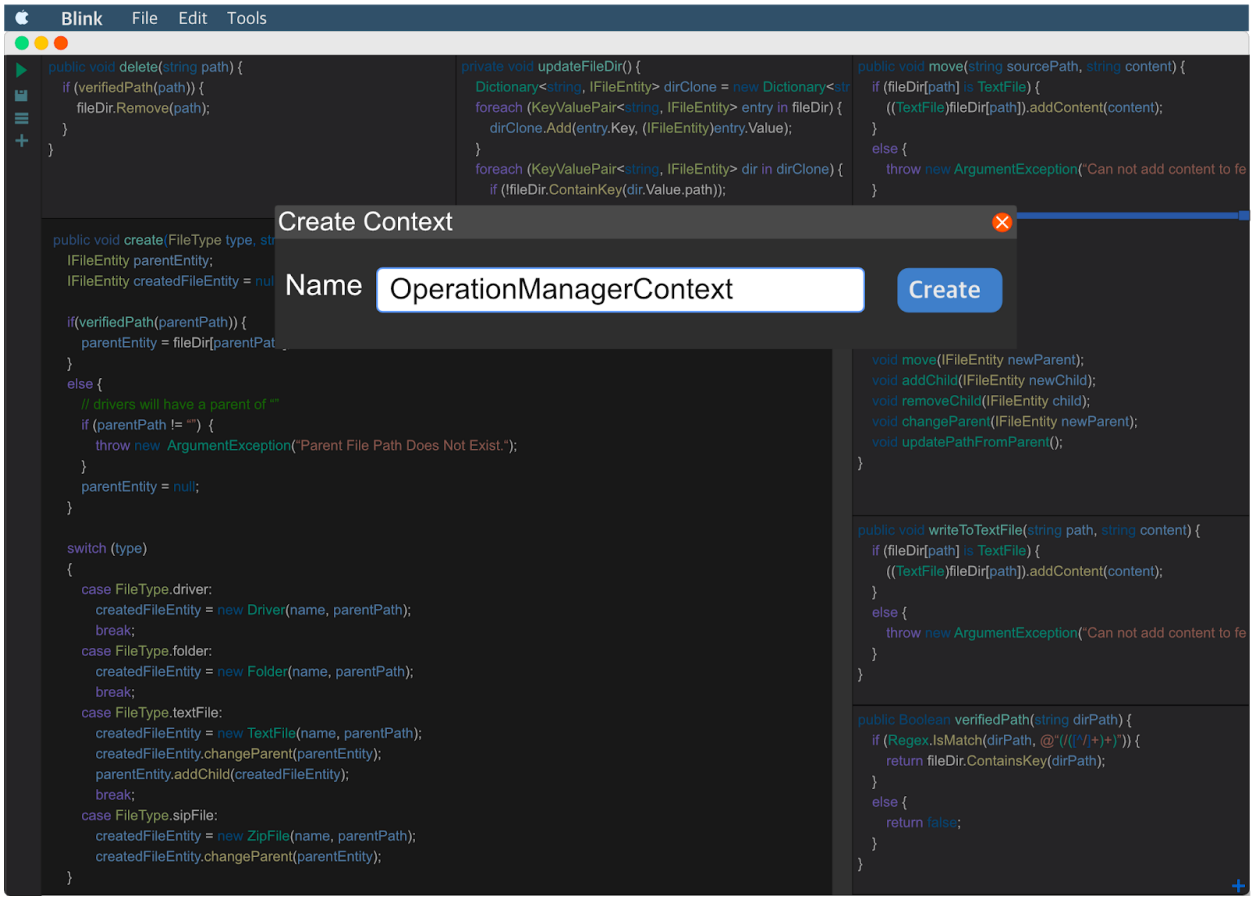


Figure 9 - Create Context

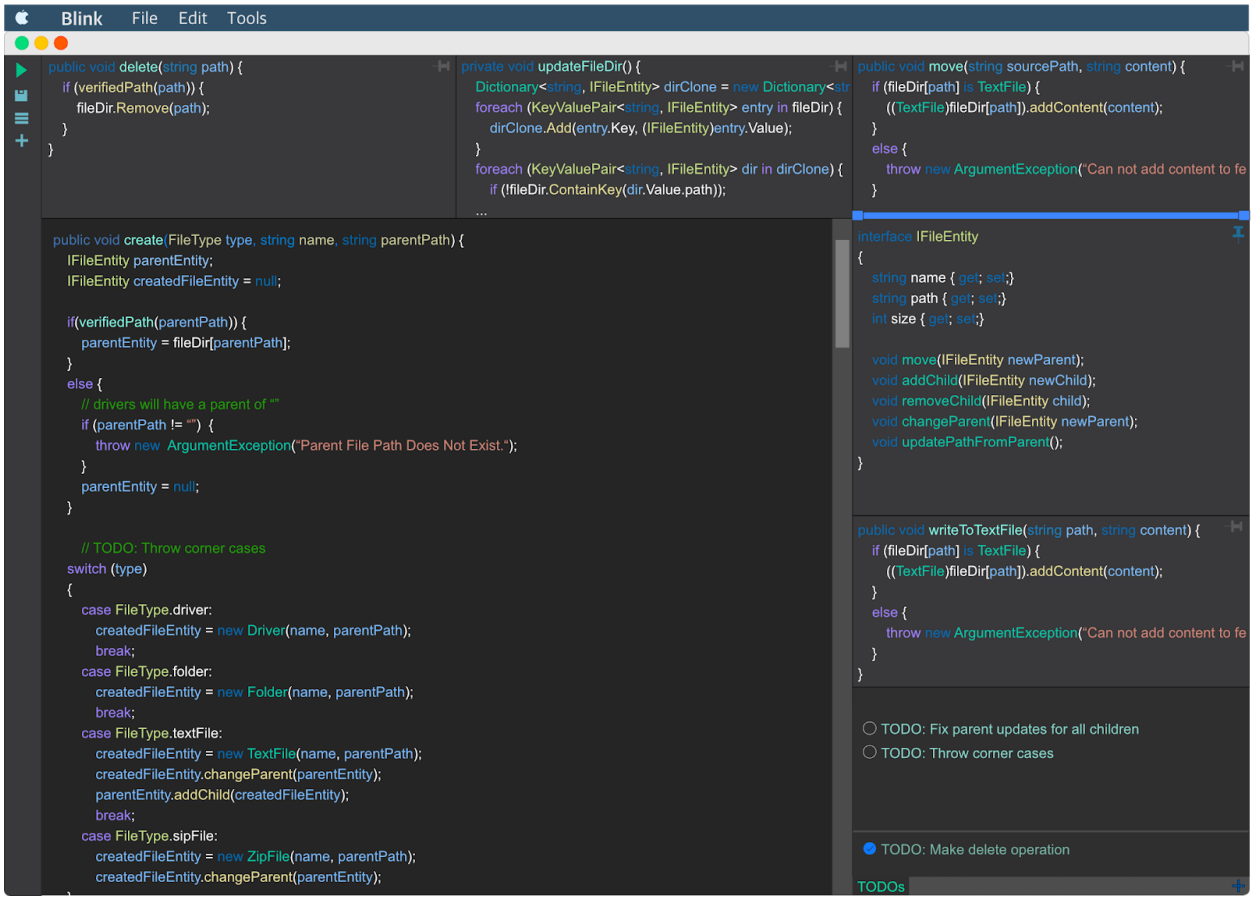


Figure 10 - TODO Pane

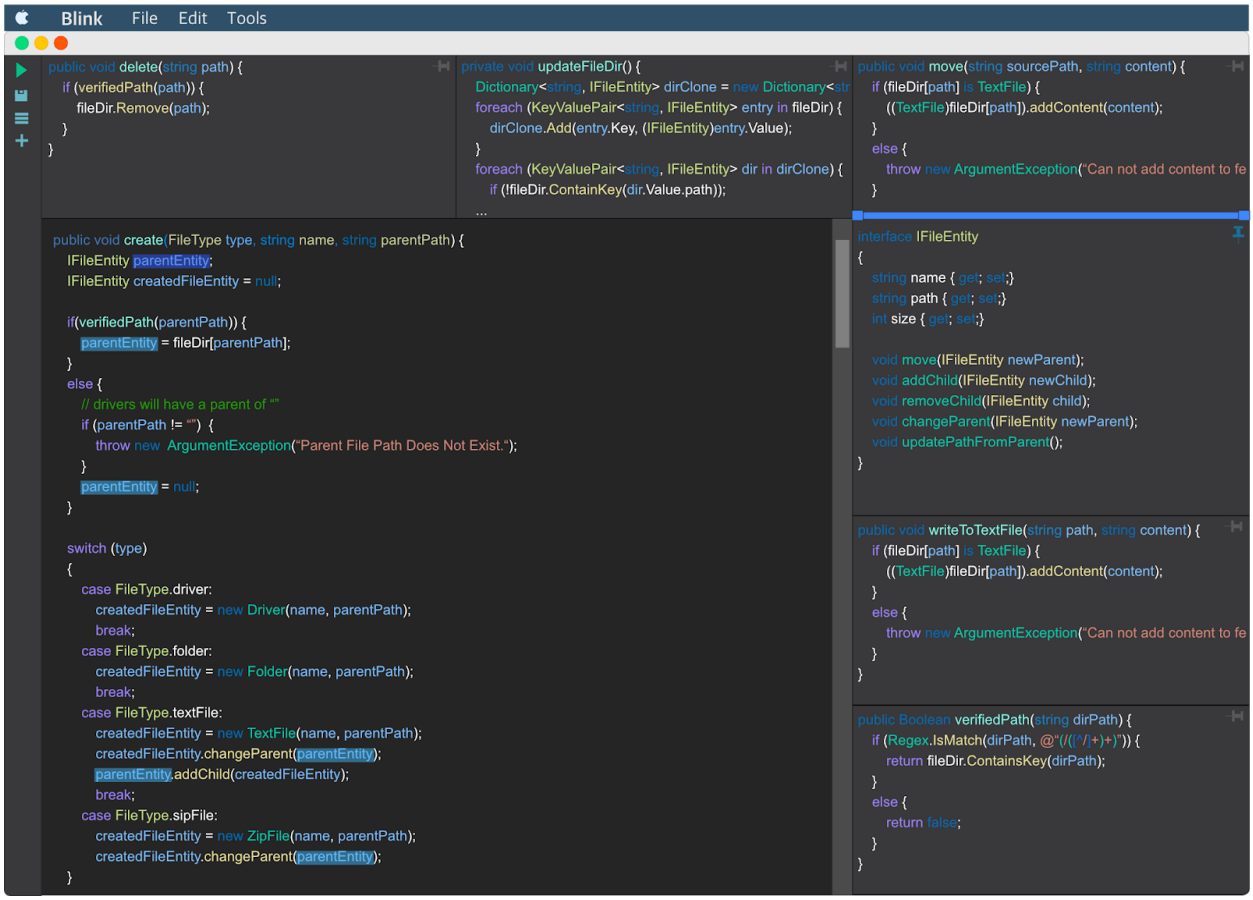


Figure 11 - Rename Variable/Function