Usage Guide

Getting Started

After installation, access AI Orchestrator commands through:

1. **Menu**: Editor > AI Orchestrator > [Command]
2. **Keyboard Shortcuts**: See SHORTCUTS.md (SHORTCUTS.md)
3. **Right-click Context Menu** (when available)

Commands

Fix Code Issues

Analyzes selected code (or entire file) for errors and applies AI-generated fixes.

How to use:
1. Select code with potential issues (or select nothing for entire file)
2. Choose **Editor > AI Orchestrator > Fix Code Issues**
3. Review the diff (if configured)
4. Fixes are applied automatically

Example:

```
// Before
func calculateArea(width: Int height: Int) -> Int {  // Missing comma
    return width * heigth  // Typo
}

// After AI fix
func calculateArea(width: Int, height: Int) -> Int {
    return width * height
}
```

Explain Code

Gets a detailed explanation of the selected code.

How to use:
1. Select the code you want explained
2. Choose **Editor > AI Orchestrator > Explain Code**
3. Explanation is inserted as comments above the selection

Example output:

```
/*
 * AI Orchestrator Code Explanation
 * ========================================
 * This function calculates the area of a rectangle.
 *
 * Parameters:
 * - width: The width of the rectangle
 * - height: The height of the rectangle
 *
 * Returns: The area (width × height)
 *
 * Time complexity: O(1)
 * Space complexity: O(1)
 */
func calculateArea(width: Int, height: Int) -> Int {
    return width * height
}
```

Generate Tests

Creates unit tests for selected functions or classes.

How to use:

1. Select a function, class, or struct
2. Choose **Editor > AI Orchestrator > Generate Tests**
3. Tests are inserted at the end of the file

Example output:

```
// MARK: - Generated Unit Tests
// Generated by AI Orchestrator for Xcode

import XCTest

class CalculatorTests: XCTestCase {

    func testCalculateArea_withPositiveValues_returnsCorrectArea() {
        let result = calculateArea(width: 5, height: 3)
        XCTAssertEqual(result, 15)
    }

    func testCalculateArea_withZeroWidth_returnsZero() {
        let result = calculateArea(width: 0, height: 10)
        XCTAssertEqual(result, 0)
    }

    func testCalculateArea_withZeroHeight_returnsZero() {
        let result = calculateArea(width: 10, height: 0)
        XCTAssertEqual(result, 0)
    }
}
```

Refactor Code

Improves code quality using AI-powered refactoring suggestions.

How to use:

1. Select code to refactor

2. Choose **Editor > AI Orchestrator > Refactor Code**
3. Review and accept the refactored code

Example:

```
// Before
func processItems(items: [String]) {
    var result = [String]()
    for i in 0..<items.count {
        let item = items[i]
        if item != "" {
            result.append(item.uppercased())
        }
    }
    print(result)
}

// After refactoring
func processItems(items: [String]) {
    let result = items
        .filter { !$0.isEmpty }
        .map { $0.uppercased() }
    print(result)
}
```

Generate Documentation

Creates Swift documentation comments for code.

How to use:
1. Select a function, class, or property
2. Choose **Editor > AI Orchestrator > Generate Documentation**
3. Documentation comments are inserted above declarations

Example output:

```
/// Calculates the area of a rectangle.
///
/// This method multiplies the width and height to compute the total area.
///
/// - Parameters:
///   - width: The width of the rectangle in units.
///   - height: The height of the rectangle in units.
/// - Returns: The calculated area as an integer.
/// - Complexity: O(1) time and space.
func calculateArea(width: Int, height: Int) -> Int {
    return width * height
}
```

Build and Fix

Builds your project and automatically fixes compilation errors.

How to use:
1. Ensure your project is open in Xcode
2. Choose **Editor > AI Orchestrator > Build and Fix**
3. The extension will:
- Trigger a build

- Capture any errors
- Send errors to AI for analysis
- Apply fixes automatically
- Optionally rebuild to verify

Workflow:

```
Build Project → Capture Errors → AI Analysis → Apply Fixes → Verify Build
```

Tips and Best Practices

1. Select Meaningful Code Blocks

For best results, select complete functions, classes, or logical blocks.

2. Review AI Suggestions

Enable “Show diff before applying” in settings to review changes.

3. Use Incremental Fixes

For large files, fix issues in smaller sections.

4. Combine Commands

Workflow example:
1. Write initial code
2. Use **Fix Code** to correct syntax
3. Use **Generate Tests** for testing
4. Use **Generate Docs** for documentation
5. Use **Refactor** for optimization

5. Configure Model Preferences

Different models excel at different tasks:
- **Claude**: Best for code generation and fixes
- **GPT-4**: Best for analysis and explanations
- **Gemini**: Good for test generation