# Code Style Document

## 1. General Code Structure

- **Single Responsibility Principle**:
  - Each class and function should have a single, clear responsibility.
  - Avoid mixing logic in a single function; instead, separate concerns into multiple functions or classes where possible.
- **Code Blocks and Indentation**:
  - Use consistent indentation (4 spaces recommended).
  - Keep code within reasonable line length (ideally under 100 characters) to improve readability and avoid horizontal scrolling.

## 2. Naming Conventions

- **Classes and Structs**:
  - Use `PascalCase` for class names (e.g., `MouseEventHandler`, `CanvasRenderer`).
- **Functions and Methods**:
  - Use `camelCase` for functions and methods (e.g., `receiveMouseEvent`, `calculateScaleFactor`).
  - Method names should be verbs to clearly indicate actions.
- **Variables**:
  - Use `camelCase` for variable names (e.g., `currentScale`, `mousePosition`).
  - Boolean variables should clearly reflect the state they represent (e.g., `isVisible`, `isMouseDown`).

## 3. Commenting and Documentation

- **Function and Class Documentation**:
  - Provide a brief summary for each class and function using Doxygen-style comments.
  - Use `@param` and `@return` tags to explain function parameters and return values.
- **Inline Comments**:
  - Use inline comments sparingly and only for complex logic that may not be immediately clear.
  - Avoid redundant comments; code should generally be self-explanatory.
- **Header Files**:
  - Place class declarations and function prototypes in header files (`.h`), while implementation details should be in corresponding source files (`.cpp`).

## 4. Error Handling and Edge Cases

- **Error Checks**:
  - Validate inputs and handle potential errors gracefully. Use assertions or exceptions where appropriate.
- **Bounds and Limits**:
  - For values like scale or position, enforce limits to prevent invalid states (e.g., negative scaling factors).

## 5. Event Handling

- **Event Triggers**:
  - Functions handling events should only perform actions directly related to the event and avoid side effects.
  - Use flags or state variables (e.g., `isMouseDown`, `hasScrolled`) to track event states across function calls if needed.

## 6. Code Layout and Formatting

- **Spacing**:
  - Use blank lines to separate logical code blocks within functions for readability.
- **Consistent Style for Conditionals**:
  - Use consistent style for conditionals, with spaces around operators (e.g., `if (value > threshold)`).

## 7. File Organization

- **Header Guards**:
  - Use header guards in all header files (`#ifndef CLASSNAME_H`, `#define CLASSNAME_H`, `#endif`) to prevent multiple inclusions.
- **File Naming**:
  - Use `PascalCase` or `camelCase` for file names, consistent with class names (e.g., `MouseHandler.h`, `CanvasRenderer.cpp`).