جامعة الناصر

AL-NASSER UNIVERSITY

Aptech
COMPUTER EDUCATION
Unleash your potential

JavaFX eProject

FILE MANA - MODERN TEXT EDITOR

Developer Guide

TABLE OF CONTENTS

INTRODUCTION

Purpose of This Guide

This Developer Guide provides comprehensive technical documentation for the File Mana application, enabling developers to understand, maintain, and extend the codebase. It includes detailed module descriptions, API documentation, and architectural insights.

Target Audience

- **Developers:** Looking to understand or modify the codebase
- **Maintainers:** Responsible for ongoing application support
- **Contributors:** Planning to add new features or fix bugs
- **Students:** Learning from the implementation patterns and techniques

Technology Stack

Component	Technology	Version	Purpose
Language	Java	21	Core application development
UI Framework	JavaFX	22	Desktop user interface
Build Tool	Maven	3.6+	Dependency management and build
Styling	CSS	3	UI theming and responsive design
Testing	JUnit	5.8+	Unit and integration testing

Design Principles

- **Separation of Concerns:** Clear distinction between UI, business logic, and data access
- **Single Responsibility:** Each class has one primary responsibility
- **Open/Closed Principle:** Open for extension, closed for modification
- **Dependency Injection:** Loose coupling between components
- **Observer Pattern:** Event-driven communication

DEVELOPMENT ENVIRONMENT SETUP

Prerequisites

Required Software:

Java Development Kit 21
```
java -version
javac -version
```

Maven 3.6+
```
mvn -version
```

Git (for version control)
```
git --version
```

Recommended IDEs:

- IntelliJ IDEA (Community or Ultimate)
- Eclipse IDE with JavaFX plugin
- Visual Studio Code with Java extensions

Project Setup

Clone and Build:

Clone the repository
```
git clone <repository-url>
cd Aptech-JavaFX-Project
```

Build the project
```
mvn clean compile
```

Run tests
```
mvn test
```

Package application
```
mvn package
```

IDE Configuration:

```
<!-- Maven dependencies (pom.xml) -->
<dependencies>
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-controls</artifactId>
    <version>22</version>
  </dependency>
  <dependency>
    <groupId>org.openjfx</groupId>
    <artifactId>javafx-fxml</artifactId>
    <version>22</version>
  </dependency>
</dependencies>
```

Running the Application

Development Mode:

Run from Maven
mvn javafx:run

Run from IDE
Set main class: com.codemavriks.aptech.MainApp
Add VM options: --module-path /path/to/javafx/lib --add-modules javafx.controls,javafx.fxml

Production Mode:

Run packaged JAR
java -jar target/FileMana.jar

PROJECT STRUCTURE

Directory Layout

```
Aptech-JavaFX-Project/
├── src/main/java/com/codemavriks/aptech/
│   ├── MainApp.java                    # Application entry point
│   ├── EditorController.java           # Main MVC controller
│   ├── components/
│   │   ├── TextEditor.java            # Text editing component
│   │   ├── SidePanel.java             # Sidebar component
│   │   └── FileNavigator.java         # File tree navigator
│   └── services/
│       └── FileService.java           # File operations service
├── src/main/resources/com/codemavriks/aptech/
│   ├── styles/
│   │   └── modern-theme.css           # Application styling
│   └── File-Mana-Logo.png             # Application icon
├── src/test/java/                      # Test classes
├── target/                             # Compiled output
├── pom.xml                             # Maven configuration
└── README.md                           # Project documentation
```

Package Organization

com.codemavriks.aptech — Main application package containing entry point and controller
com.codemavriks.aptech.components — Reusable UI components and custom controls
com.codemavriks.aptech.services — Business logic and service layer classes
com.codemavriks.aptech.styles — CSS styling and theme resources

Naming Conventions

- **Classes:** PascalCase (e.g., FileService, TextEditor)
- **Methods:** camelCase (e.g., createFileSet, updateContent)
- **Variables:** camelCase (e.g., currentFileBaseName)
- **Constants:** UPPER_SNAKE_CASE

ARCHITECTURE OVERVIEW

Architectural Pattern

Model-View-Controller (MVC)

```
┌──────────────┐    ┌──────────────┐    ┌──────────────┐
│     VIEW     │ ◄► │  CONTROLLER  │ ◄► │    MODEL     │
└──────────────┘    └──────────────┘    └──────────────┘
```

Design Patterns Used

- **Observer Pattern:** Event-driven communication
- **Service Layer Pattern:** Business logic separation
- **Component Pattern:** Modular UI components

Design Patterns Used

Observer Pattern

- Event-driven communication between components
- Property change listeners for UI updates
- File change notifications

Service Layer Pattern

- Centralized business logic in service classes
- Separation of concerns between UI and business logic
- Reusable service methods

Component Pattern

- Modular UI components with clear interfaces
- Encapsulated functionality and state
- Reusable across different contexts

MODULE DESCRIPTIONS

MainApp Module

File: `src/main/java/com/codemavriks/aptech/MainApp.java`

Purpose: Application entry point and window management

Key Responsibilities:

- JavaFX application initialization
- Primary stage setup and configuration
- Icon loading and window properties
- Application lifecycle management

Code Structure:

```
public class MainApp extends Application {
    @Override
    public void start(Stage primaryStage) throws Exception {
        // Load FXML and create scene
        // Set up window properties
        // Load application icons
        // Show primary stage
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Key Methods:

- start(Stage primaryStage): Main application startup
- loadIcons(): Loads application icons for title bar and taskbar
- setStageIcons(Stage stage): Applies icons to stage

Dependencies:

- JavaFX Application class
- EditorController for main UI
- Resource loading utilities

EditorController Module

File: `src/main/java/com/codemavriks/aptech/EditorController.java`

Purpose: Main MVC controller coordinating all application functionality

Key Responsibilities:

- UI component initialization and coordination
- Event handling and user interaction management
- Business logic coordination through services
- State management and data binding

Code Structure:

```
@Controller
public class EditorController {
    // FXML injected components
    @FXML private TextEditor textEditor;
    @FXML private SidePanel sidePanel;
    @FXML private FileNavigator fileNavigator;

    // Service dependencies
    private FileService fileService;

    // Initialization and event handlers
    public void initialize() { /* Setup logic */ }

    // Event handler methods
    @FXML private void handleCreateFiles() { /* File creation */ }
    @FXML private void handleReplaceWord() { /* Word replacement */ }
}
```

Key Methods:

- initialize(): Component initialization and setup
- handleCreateFiles(): File creation event handler
- handleReplaceWord(): Word replacement event handler
- updateFileNavigator(): Refreshes file tree display
- setupAutoSave(): Configures automatic saving

Event Handling:

- Button click events for file operations
- Text change events for auto-save triggering
- File selection events from navigator
- Keyboard shortcut handling

TextEditor Component

File: `src/main/java/com/codemavriks/aptech/components/TextEditor.java`

Purpose: Full-featured text editing component with advanced capabilities

Key Responsibilities:

- Text content display and editing
- Syntax highlighting and formatting
- Undo/redo functionality
- Find and replace operations
- Content change notifications

Code Structure:

```java
public class TextEditor extends TextArea {
    private UndoManager undoManager;
    private FindReplaceDialog findDialog;

    public TextEditor() {
        setupEditor();
        setupEventHandlers();
        setupKeyboardShortcuts();
    }

    private void setupEditor() {
        // Configure text area properties
        // Set up styling and appearance
        // Initialize undo/redo system
    }
}
```

Key Features:

- **Syntax Highlighting:** Color-coded text for better readability
- **Line Numbers:** Optional line numbering for navigation
- **Word Wrap:** Automatic line wrapping for long text
- **Undo/Redo:** Full undo/redo stack with Ctrl+Z/Ctrl+Y
- **Find/Replace:** Integrated search and replace functionality

API Methods:

```java
// Content management
public void setContent(String content)
public String getContent()
public void appendContent(String content)

// Editing operations
public void undo()
public void redo()
public boolean canUndo()
public boolean canRedo()

// Search functionality
public void showFindDialog()
public void findNext(String searchText)
public void replaceAll(String searchText, String replacement)
```

SidePanel Component

File: `src/main/java/com/codemavriks/aptech/components/SidePanel.java`

Purpose: Sidebar component containing file creation and word replacement controls

Key Responsibilities:

- File creation form management
- Word replacement input handling
- Status information display
- User input validation

Code Structure:

```java
public class SidePanel extends VBox {
    // UI components
    private TextField baseNameField;
    private TextArea contentArea;
    private TextField positionField;
    private TextField replacementField;
    private Label statusLabel;

    public SidePanel() {
        setupLayout();
        setupEventHandlers();
        setupValidation();
    }
}
```

Form Sections:

1. File Creation Section
   - Base name input field
   - Content text area
   - Create Files button
   - Clear form button

2. Word Replacement Section
   - Position input field (numeric)
   - Replacement text field
   - Replace Word button
   - Clear fields button

3. Status Section
   - Current operation status
   - File count display
   - Auto-save indicator

Validation Logic:

```java
// Input validation methods
private boolean validateBaseName(String baseName)
private boolean validatePosition(String position)
private boolean validateContent(String content)

// Error handling
private void showValidationError(String message)
private void clearValidationErrors()
```

FileNavigator Component

File: `src/main/java/com/codemavriks/aptech/components/FileNavigator.java`

Purpose: VSCode-like file tree navigator with context menu operations

Key Responsibilities:

- Hierarchical file structure display
- File and folder navigation
- Context menu operations
- File selection and opening

Code Structure:

```java
public class FileNavigator extends TreeView<File> {
    private ContextMenu contextMenu;
    private TreeItem<File> rootItem;

    public FileNavigator() {
        setupTreeView();
        setupContextMenu();
        setupEventHandlers();
    }

    private void buildFileTree() {
        // Recursively build tree structure
        // Add file and folder icons
        // Set up selection handling
    }
}
```

Tree Structure:

```
📁 Created files/
├── 📁 Ahmed/
│   ├── 📄 Ahmed-org.txt
│   ├── 📄 Ahmed-rev.txt
│   └── 📄 Ahmed-byte.txt
├── 📁 Document/
│   ├── 📄 Document-org.txt
│   ├── 📄 Document-rev.txt
│   └── 📄 Document-byte.txt
└── ...
```

Context Menu Operations:

- Open file in editor
- Delete file or folder
- Rename file or folder
- Refresh tree view
- Show file properties

FileService Module

File: `src/main/java/com/codemavriks/aptech/services/FileService.java`

Purpose: Centralized file operations and business logic service

Key Responsibilities:

- File creation and management
- Content processing (reversal, byte conversion)
- Auto-save functionality
- File system operations

Code Structure:

```java
@Service
public class FileService {
    private static final String BASE_DIRECTORY = "Created files";
    private Timer autoSaveTimer;
    private String currentFileBaseName;
    private String lastSavedContent;
    // Core file operations
    public void createFileSet(String baseName, String content)
    public void updateFileSet(String baseName, String content)
    public String loadFileContent(String filePath)

    // Content processing
    public String reverseContent(String content)
    public String convertToByteString(String content)
    public String replaceWordAtPosition(String content, int position, String
replacement)
}
```

File Operations API:

File set management

```
public void createFileSet(String baseName, String content) throws IOException
public void updateFileSet(String baseName, String content) throws IOException
public boolean fileSetExists(String baseName)
public void deleteFileSet(String baseName) throws IOException
```

Content processing

```
public String reverseContent(String content)
public String convertToByteString(String content)
public String replaceWordAtPosition(String content, int position, String
replacement)
```

Auto-save functionality

```
public void enableAutoSave(Runnable onSave, Runnable onError)
public void disableAutoSave()
public boolean isAutoSaveEnabled()
```

Auto-Save Implementation:

```java
private void setupAutoSave() {
    autoSaveTimer = new Timer(true);
    autoSaveTimer.scheduleAtFixedRate(new TimerTask() {
        @Override
        public void run() {
            if (hasContentChanged()) {
                try {
                    saveCurrentContent();
                    Platform.runLater(() -> notifyAutoSaveSuccess());
                } catch (IOException e) {
                    Platform.runLater(() -> notifyAutoSaveError(e));
                }
            }
        }
    },
}
```

API DOCUMENTATION

Core APIs

FileService API

```java
public class FileService {
    /**
     * Creates a new file set with three synchronized files
     * @param baseName The base name for the file set
     * @param content The initial content
     * @throws IOException If file creation fails
     */
    public void createFileSet(String baseName, String content) throws IOException

    /**
     * Updates an existing file set with new content
     * @param baseName The base name of the file set
     * @param content The new content
     * @throws IOException If file update fails
     */
    public void updateFileSet(String baseName, String content) throws IOException

    /**
     * Reverses the character order of the input string
     * @param content The string to reverse
     * @return The reversed string
     */
    public String reverseContent(String content)

    /**
     * Converts text to UTF-8 byte representation
     * @param content The text to convert
     * @return Space-separated byte values
     */
    public String convertToByteString(String content)
}
```

TextEditor API

```
public class TextEditor extends TextArea {
    /**
     * Sets the content of the text editor
     * @param content The text content to display
     */
    public void setContent(String content)

    /**
     * Gets the current content of the text editor
     * @return The current text content
     */
    public String getContent()

    /**
     * Undoes the last editing operation
     */
    public void undo()

    /**
     * Redoes the last undone operation
     */
    public void redo()
}
```

Event System

Event Types:

```
// File operation events
public class FileCreatedEvent extends Event
public class FileUpdatedEvent extends Event
public class FileDeletedEvent extends Event

// Content change events
public class ContentChangedEvent extends Event
public class WordReplacedEvent extends Event

// Auto-save events
public class AutoSaveSuccessEvent extends Event
public class AutoSaveErrorEvent extends Event
Event Handling:
```

```java
// Register event listeners
fileService.addFileOperationListener(event -> {
    // Handle file operation events
});

textEditor.addContentChangeListener(event -> {
    // Handle content changes
});
```

Configuration API

Application Configuration:

```java
public class AppConfig {
    // Auto-save settings
    public static final int AUTO_SAVE_INTERVAL = 30000; // 30 seconds
    public static final boolean AUTO_SAVE_ENABLED = true;

    // File settings
    public static final String BASE_DIRECTORY = "Created files";
    public static final String FILE_ENCODING = "UTF-8";

    // UI settings
    public static final double SIDEBAR_WIDTH_RATIO = 0.3;
    public static final double EDITOR_WIDTH_RATIO = 0.7;
}
```

DATABASE/FILE STRUCTURE

File System Organization

Directory Structure:

```
Application Root/
├── Created files/                    # Main data directory
│   ├── [BaseName1]/                 # File set folder
│   │   ├── [BaseName1]-org.txt      # Original content
│   │   ├── [BaseName1]-rev.txt      # Reversed content
│   │   └── [BaseName1]-byte.txt     # Byte representation
│   ├── [BaseName2]/
│   │   ├── [BaseName2]-org.txt
│   │   ├── [BaseName2]-rev.txt
│   │   └── [BaseName2]-byte.txt
│   └── ...
├── FileMana.jar                     # Application executable
└── logs/                            # Application logs (if enabled)
```

File Naming Convention

Pattern: [BaseName]-[Type].txt

File Types:

- org: Original content file
- rev: Reversed content file
- byte: Byte codes file

Naming Rules:

- Base names are sanitized to remove special characters
- Duplicate names get automatic suffixes (_1, _2, etc.)
- All files use UTF-8 encoding
- Extensions are always .txt

Data Persistence

File Content Format:

Original File (basename-org.txt):

Plain text content as entered by user

Reversed File (basename-rev.txt):

Character-reversed version of original content

Byte File (basename-byte.txt):

Space-separated UTF-8 byte values

Example: "72 101 108 108 111" for "Hello"

Metadata Storage:

- No separate metadata files
- File system timestamps used for modification tracking
- File sizes calculated dynamically
- Content integrity verified through comparison

BUILD AND DEPLOYMENT

Maven Build Configuration

pom.xml Structure:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.codemavriks</groupId>
    <artifactId>file-mana</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <properties>
        <maven.compiler.source>21</maven.compiler.source>
        <maven.compiler.target>21</maven.compiler.target>
        <javafx.version>22</javafx.version>
        <junit.version>5.8.2</junit.version>
    </properties>

    <dependencies>
        <!-- JavaFX dependencies -->
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-controls</artifactId>
            <version>${javafx.version}</version>
        </dependency>

        <!-- Testing dependencies -->
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter</artifactId>
            <version>${junit.version}</version>
            <scope>test</scope>
        </dependency>
    </dependencies>
</project>
```

Build Commands

Development Build:

```
bash
Clean and compile
mvn clean compile
```

Run tests

```
mvn test
```

Package without tests

```
mvn package -DskipTests
```

Run application

```
mvn javafx:run
```

Production Build:

```
bash
 Full build with tests
mvn clean package
```

Create executable JAR

```
mvn clean package shade:shade
```

Generate documentation

```
mvn javadoc:javadoc
```

Deployment Options

Standalone JAR:

bash
Create fat JAR with dependencies
mvn clean package

Run standalone

java -jar target/FileMana.jar

Platform-Specific Packages:

bash
Windows installer
jpackage --input target/ --name FileMana --main-jar FileMana.jar --type msi

macOS app bundle

jpackage --input target/ --name FileMana --main-jar FileMana.jar --type dmg

Linux package

jpackage --input target/ --name FileMana --main-jar FileMana.jar --type deb

EXTENDING THE APPLICATION

Adding New Features

Step 1: Define Requirements

- Identify the new functionality needed
- Determine impact on existing modules
- Plan integration points

Step 2: Design Implementation

- Choose appropriate design patterns
- Define new classes and interfaces
- Plan data flow and dependencies

Step 3: Implement Feature

- Create new modules following existing patterns
- Update existing modules as needed
- Add comprehensive tests

Example: Adding File Export Feature

New service method

```
public class FileService {
    public void exportFileSet(String baseName, String format) throws IOException
{
        switch (format.toLowerCase()) {
            case "pdf":
                exportToPdf(baseName);
                break;
            case "html":
                exportToHtml(baseName);
                break;
            default:
                throw new IllegalArgumentException("Unsupported format: " +
format);
        }
    }
}
```

UI integration

```
public class SidePanel {
    private void addExportSection() {
        ComboBox<String> formatCombo = new ComboBox<>();
        formatCombo.getItems().addAll("PDF", "HTML", "TXT");

        Button exportButton = new Button("Export");
        exportButton.setOnAction(e -> handleExport());
    }
}
```

Customizing UI Components

Creating Custom Components:

```
public class CustomTextEditor extends TextEditor {
    private LineNumberFactory lineNumbers;

    public CustomTextEditor() {
        super();
        setupLineNumbers();
        setupCustomStyling();
    }

    private void setupLineNumbers() {
        lineNumbers = new LineNumberFactory(this);
        setParagraphGraphicFactory(lineNumbers);
    }
}
```

Extending Existing Components:

```
public class EnhancedSidePanel extends SidePanel {
    private TabPane tabPane;

    public EnhancedSidePanel() {
        super();
        convertToTabbedInterface();
    }

    private void convertToTabbedInterface() {
        tabPane = new TabPane();
        tabPane.getTabs().addAll(
            new Tab("Files", createFileSection()),
            new Tab("Replace", createReplaceSection()),
            new Tab("Settings", createSettingsSection())
        );
    }
}
```

Plugin Architecture

Plugin Interface:

```
public interface FileManPlugin {
    String getName();
    String getVersion();
    void initialize(PluginContext context);
    void shutdown();
    List<MenuItem> getMenuItems();
}

public class PluginContext {
    private FileService fileService;
    private EditorController controller;

  Provide access to core functionality
    public FileService getFileService() { return fileService; }
    public EditorController getController() { return controller; }
}
```

Plugin Manager:

```java
public class PluginManager {
    private List<FileManPlugin> plugins = new ArrayList<>();

    public void loadPlugin(String pluginPath) throws Exception {
        // Load plugin JAR
        // Instantiate plugin class
        // Initialize plugin
        // Register with application
    }

    public void unloadPlugin(String pluginName) {
        // Find plugin
        // Shutdown plugin
        // Remove from registry
    }
}
```

Performance Optimization

Memory Optimization:

```java
// Lazy loading for large files
public class LazyFileLoader {
    private WeakReference<String> contentCache;

    public String getContent() {
        String content = contentCache != null ? contentCache.get() : null;
        if (content == null) {
            content = loadFromFile();
            contentCache = new WeakReference<>(content);
        }
        return content;
    }
}
```

Asynchronous operations

```java
public class AsyncFileService {
    private ExecutorService executor = Executors.newCachedThreadPool();

    public CompletableFuture<Void> createFileSetAsync(String baseName, String
content) {
        return CompletableFuture.runAsync(() -> {
            try {
                createFileSet(baseName, content);
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }, executor);
    }
}
```

CONCLUSION

This Developer Guide provides comprehensive documentation for understanding, maintaining, and extending the File Mana application. The modular architecture, clear separation of concerns, and well-defined APIs make it easy for developers to work with the codebase.

Key Development Principles:

- Maintainable Code: Clear structure and comprehensive documentation
- Extensible Design: Plugin-ready architecture for future enhancements
- Testable Components: Comprehensive test coverage and utilities
- Performance Conscious: Optimized for responsiveness and efficiency

Next Steps for Developers:

- Study the Modules: Understand each component's role and responsibilities
- Run the Tests: Verify your development environment setup
- Experiment: Try adding small features to understand the architecture
- Contribute: Follow the established patterns when adding new functionality