```
// PureUltraCanvasDemo.cpp
// Demo using ONLY cross-platform UltraCanvas framework APIs
// Version: 7.0.0 - Pure cross-platform implementation
// Last Modified: 2025-07-16
// Author: UltraCanvas Framework

#include "UltraCanvasUI.h"
#include <iostream>
#include <thread>

using namespace UltraCanvas;

class CrossPlatformFrameworkWindow : public UltraCanvasWindow {
private:
    std::vector<std::shared_ptr<UltraCanvasElement>> uiElements;
    bool shouldClose = false;

public:
    virtual bool Create(const WindowConfig& config) override {
        if (UltraCanvasWindow::Create(config)) {
            CreateUserInterface();
            return true;
        }
        return false;
    }

    virtual void Destroy() override {
        if (!created_) {
            std::cout << "=== CrossPlatformFrameworkWindow already destroyed ===" << std::endl;
            return;
        }
        std::cout << "=== Destroying CrossPlatformFrameworkWindow ===" << std::endl;
        try {
            uiElements.clear();
            ClearElements();
        } catch (...) {
            std::cerr << "Exception during window destruction" << std::endl;
        }
        UltraCanvasWindow::Destroy();
    }

    void CreateUserInterface() {
        std::cout << "=== Creating Cross-Platform UI Elements ===" << std::endl;

        // Create buttons using ONLY UltraCanvas framework APIs
        auto copyButton = CreateButton("copy_btn", 1001, 50, 150, 180, 50, "Copy Text");
        copyButton->onClicked = [this]() {
            std::cout << "=== COPY BUTTON CLICKED! ===" << std::endl;
            std::cout << "Cross-platform button working perfectly!" << std::endl;
        };

        auto pasteButton = CreateButton("paste_btn", 1002, 250, 150, 180, 50, "Paste Text");
        pasteButton->onClicked = [this]() {
            std::cout << "=== PASTE BUTTON CLICKED! ===" << std::endl;
            std::cout << "Cross-platform button working perfectly!" << std::endl;
```

```
    };

    auto clearButton = CreateButton("clear_btn", 1003, 450, 150, 180, 50, "Clear All");
    clearButton->onClicked = [this]() {
        std::cout << "=== CLEAR BUTTON CLICKED! ===" << std::endl;
        std::cout << "Cross-platform button working perfectly!" << std::endl;
    };

    auto exitButton = CreateButton("exit_btn", 1004, 300, 450, 200, 60, "Exit Application");
    exitButton->onClicked = [this]() {
        std::cout << "=== EXIT BUTTON CLICKED! ===" << std::endl;
        std::cout << "Requesting application exit..." << std::endl;
        shouldClose = true;

        // Use the cross-platform way to request exit
        UltraCanvasApplication* app = UltraCanvasApplication::GetInstance();
        if (app) {
            app->Exit();
        }
    };

    // Add all elements to the window using framework methods
    AddElement(copyButton);
    AddElement(pasteButton);
    AddElement(clearButton);
    AddElement(exitButton);

    // Store references
    uiElements.push_back(copyButton);
    uiElements.push_back(pasteButton);
    uiElements.push_back(clearButton);
    uiElements.push_back(exitButton);

    std::cout << "Created " << uiElements.size() << " cross-platform UI elements" << std::endl;
}

bool ShouldClose() {
    return shouldClose || !IsVisible();
}

// Handle framework events using ONLY cross-platform event types
void OnEvent(const UCEvent& event) override {
    // Only log important events to reduce noise
    if (event.type == UCEventType::MouseDown || event.type == UCEventType::KeyDown) {
        std::cout << "CROSS-PLATFORM EVENT: type=" << static_cast<int>(event.type)
                  << " pos=(" << event.x << "," << event.y << ")" << std::endl;
    }

    // Handle window-level events using cross-platform approach
    if (event.type == UCEventType::KeyDown) {
        if (event.virtualKey == UCKeys::Escape) {
            std::cout << ">>> ESC KEY PRESSED - REQUESTING EXIT!" << std::endl;
            shouldClose = true;

            // Use cross-platform exit request
```

```
            UltraCanvasApplication* app = UltraCanvasApplication::GetInstance();
            if (app) {
                app->Exit();
            }
            return;
        }
    }

    if (event.type == UCEventType::WindowClose) {
        std::cout << ">>> WINDOW CLOSE EVENT - REQUESTING EXIT!" << std::endl;
        shouldClose = true;

        // Use cross-platform exit request
        UltraCanvasApplication* app = UltraCanvasApplication::GetInstance();
        if (app) {
            app->Exit();
        }
        return;
    }

    // Let the base window handle the event (cross-platform event dispatch)
    UltraCanvasWindow::OnEvent(event);
}

void Render() override {
    RenderWithDebugEnhanced();
}

// Custom rendering using ONLY cross-platform UltraCanvas rendering APIs
void Render1() {
    // CRITICAL: First call the base class render to set up the render context
    // This calls ULTRACANVAS_WINDOW_RENDER_SCOPE(this) internally
    UltraCanvasWindow::Render();

    // Now the render context is properly set up and GetRenderContext() will work
    // No need for ULTRACANVAS_RENDER_SCOPE() here since base class already set it up

    // Get window dimensions using cross-platform method
    int width, height;
    GetSize(width, height);

    // Draw window background using cross-platform rendering
    SetFillColor(Color(245, 248, 255, 255)); // Light blue background
    FillRectangle(Rect2D(0, 0, width, height));

    // Draw title using cross-platform text rendering
    SetTextColor(Color(0.1f, 0.2f, 0.4f, 1.0f));
    SetFont("Arial", 24);
    DrawText("UltraCanvas Cross-Platform Demo", Point2D(50, 50));

    // Draw subtitle
    SetTextColor(Color(0.3f, 0.4f, 0.6f, 1.0f));
    SetFont("Arial", 14);
    DrawText("100% Cross-Platform Implementation", Point2D(50, 75));
```

```
        // Draw instructions
        SetTextColor(Color(0.2f, 0.3f, 0.5f, 1.0f));
        SetFont("Arial", 12);
        DrawText("This demo uses ONLY cross-platform UltraCanvas APIs.", Point2D(50, 280));
        DrawText("No platform-specific code anywhere in the application!", Point2D(50, 300));
        DrawText("Same code works on Linux, Windows, macOS, and more.", Point2D(50, 320));
        DrawText("Press ESC or click Exit to close.", Point2D(50, 340));
    }

// Enhanced debug version of RenderWithDebug() method
    void RenderWithDebugEnhanced() {
        std::cout << "\n=== STARTING RENDER DEBUG ===" << std::endl;

        // Call base class render to set up the render context
        UltraCanvasWindow::Render();

        // Debug: Verify render context is available
        IRenderContext* ctx = GetRenderContext();
        if (!ctx) {
            std::cerr << "ERROR: No render context available after base class render!" << std::endl;
            return;
        }

        std::cout << "DEBUG: Render context is available: " << ctx << std::endl;

        // Get window dimensions using cross-platform method
        int width, height;
        GetSize(width, height);
        std::cout << "DEBUG: Window dimensions: " << width << "x" << height << std::endl;

        // Draw window background using cross-platform rendering
        std::cout << "DEBUG: Setting background color..." << std::endl;
        SetFillColor(Color(245, 248, 255, 255)); // Light blue background
        FillRectangle(Rect2D(0, 0, width, height));
        std::cout << "DEBUG: Background filled" << std::endl;

        // Test 1: Draw title using cross-platform text rendering
        std::cout << "\n--- Testing Title Text ---" << std::endl;

        // Set text color and verify
        Color titleColor(0.1f * 255, 0.2f * 255, 0.4f * 255, 1.0f * 255);
        std::cout << "DEBUG: Setting text color to: R=" << (int)titleColor.r
                  << " G=" << (int)titleColor.g << " B=" << (int)titleColor.b
                  << " A=" << (int)titleColor.a << std::endl;
        SetTextColor(titleColor);

        // Verify text style was set
        const TextStyle& currentTextStyle = ctx->GetTextStyle();
        std::cout << "DEBUG: Current text style after SetTextColor:" << std::endl;
        std::cout << "  - fontFamily: '" << currentTextStyle.fontFamily << "'" << std::endl;
        std::cout << "  - fontSize: " << currentTextStyle.fontSize << std::endl;
        std::cout << "  - textColor: R=" << (int)currentTextStyle.textColor.r
                  << " G=" << (int)currentTextStyle.textColor.g
                  << " B=" << (int)currentTextStyle.textColor.b
                  << " A=" << (int)currentTextStyle.textColor.a << std::endl;
```

```
    // Set font and verify
    std::cout << "DEBUG: Setting font to Arial, 24..." << std::endl;
    SetFont("Arial", 24);

    // Verify font was set
    const TextStyle& fontStyle = ctx->GetTextStyle();
    std::cout << "DEBUG: Current text style after SetFont:" << std::endl;
    std::cout << "  - fontFamily: '" << fontStyle.fontFamily << "'" << std::endl;
    std::cout << "  - fontSize: " << fontStyle.fontSize << std::endl;
    std::cout << "  - textColor: R=" << (int)fontStyle.textColor.r
              << " G=" << (int)fontStyle.textColor.g
              << " B=" << (int)fontStyle.textColor.b
              << " A=" << (int)fontStyle.textColor.a << std::endl;

    // Test text measurement first
    std::string titleText = "UltraCanvas Cross-Platform Demo";
    Point2D titleSize = ctx->MeasureText(titleText);
    std::cout << "DEBUG: Text '" << titleText << "' measures: "
              << titleSize.x << "x" << titleSize.y << " pixels" << std::endl;

    // Now draw the text
    Point2D titlePos(50, 50);
    std::cout << "DEBUG: About to draw title text at (" << titlePos.x << "," << titlePos.y << ")" <<
std::endl;
    DrawText(titleText, titlePos);
    std::cout << "DEBUG: Title text draw call completed" << std::endl;

    // Test 2: Try direct context call to bypass inline functions
    std::cout << "\n--- Testing Direct Context Call ---" << std::endl;

    // Set up text style manually
    TextStyle directStyle;
    directStyle.fontFamily = "Arial";
    directStyle.fontSize = 16;
    directStyle.textColor = Color(255, 0, 0, 255); // Red text
    ctx->SetTextStyle(directStyle);

    std::cout << "DEBUG: Set text style directly on context" << std::endl;
    std::cout << "DEBUG: About to call ctx->DrawText directly..." << std::endl;

    // Call DrawText directly on the context
    ctx->DrawText("DIRECT CONTEXT TEST", Point2D(50, 100));
    std::cout << "DEBUG: Direct context DrawText completed" << std::endl;

    // Test 3: Test with different colors
    std::cout << "\n--- Testing Different Colors ---" << std::endl;

    // Test with solid colors (no float multiplication)
    SetTextColor(Color(255, 0, 0, 255)); // Solid red
    SetFont("Arial", 14);
    std::cout << "DEBUG: About to draw RED text" << std::endl;
    DrawText("RED TEXT TEST", Point2D(50, 130));

    SetTextColor(Color(0, 255, 0, 255)); // Solid green
```

```
        std::cout << "DEBUG: About to draw GREEN text" << std::endl;
        DrawText("GREEN TEXT TEST", Point2D(50, 150));

        SetTextColor(Color(0, 0, 255, 255)); // Solid blue
        std::cout << "DEBUG: About to draw BLUE text" << std::endl;
        DrawText("BLUE TEXT TEST", Point2D(50, 170));

        // Force Cairo to flush all operations
        std::cout << "\n--- Flushing Render Context ---" << std::endl;
        ctx->Flush();
        std::cout << "DEBUG: Render context flushed" << std::endl;

        std::cout << "=== RENDER DEBUG COMPLETE ===\n" << std::endl;
    }

// Alternative test: Use the explicit scope method with more debug info
    void RenderWithExplicitScopeDebug() {
        std::cout << "\n=== STARTING EXPLICIT SCOPE DEBUG ===" << std::endl;

        // Call base class render to draw UI elements first
        UltraCanvasWindow::Render();

        // Explicitly set up the render context scope
        {
            ULTRACANVAS_WINDOW_RENDER_SCOPE(this);

            IRenderContext* ctx = GetRenderContext();
            std::cout << "DEBUG: Context after explicit scope: " << ctx << std::endl;

            if (!ctx) {
                std::cerr << "ERROR: No context even after explicit scope!" << std::endl;
                return;
            }

            // Get window dimensions
            int width, height;
            GetSize(width, height);

            // Draw background
            std::cout << "DEBUG: Drawing background..." << std::endl;
            SetFillColor(Color(245, 248, 255, 255));
            FillRectangle(Rect2D(0, 0, width, height));

            // Draw text with debugging
            std::cout << "DEBUG: Drawing text with explicit scope..." << std::endl;
            SetTextColor(Color(0, 0, 0, 255)); // Black text
            SetFont("Arial", 24);

            const TextStyle& style = ctx->GetTextStyle();
            std::cout << "DEBUG: Final text style before drawing:" << std::endl;
            std::cout << "  - fontFamily: '" << style.fontFamily << "'" << std::endl;
            std::cout << "  - fontSize: " << style.fontSize << std::endl;
            std::cout << "  - textColor: R=" << (int)style.textColor.r
                      << " G=" << (int)style.textColor.g
                      << " B=" << (int)style.textColor.b
```

```
                  << " A=" << (int)style.textColor.a << std::endl;

            DrawText("EXPLICIT SCOPE TEST", Point2D(50, 50));

            ctx->Flush();
            std::cout << "DEBUG: Explicit scope render complete" << std::endl;
        }

        std::cout << "=== EXPLICIT SCOPE DEBUG COMPLETE ===\n" << std::endl;
    }};

class PureCrossPlatformApp {
private:
    std::unique_ptr<UltraCanvasApplication> theApp;
    std::unique_ptr<CrossPlatformFrameworkWindow> mainWindow;
    bool appInitialized = false;

public:
    PureCrossPlatformApp() {
        std::cout << "=== Initializing Pure Cross-Platform App ===" << std::endl;

        try {
            // Create the cross-platform application instance
            theApp = std::make_unique<UltraCanvasApplication>();

            if (!theApp->Initialize()) {
                throw std::runtime_error("Cross-platform application initialization failed");
            }
            appInitialized = true;
            std::cout << "UltraCanvas cross-platform application initialized successfully" << std::endl;

            CreateMainWindow();
            SetupGlobalEventHandling();

        } catch (const std::exception& e) {
            std::cerr << "Exception in constructor: " << e.what() << std::endl;
            throw;
        }
    }

    void CreateMainWindow() {
        std::cout << "=== Creating Cross-Platform Window ===" << std::endl;

        WindowConfig config;
        config.title = "UltraCanvas Pure Cross-Platform Demo";
        config.width = 800;
        config.height = 600;
        config.resizable = true;
        config.x = 100;
        config.y = 100;
        config.backgroundColor = Color(245, 248, 255, 255); // Light blue background

        try {

            mainWindow = std::make_unique<CrossPlatformFrameworkWindow>();
```

```
        mainWindow->Create(config);
        //theApp->RegisterWindow(mainWindow.get());

        std::cout << "Cross-platform window created successfully" << std::endl;

    } catch (const std::exception& e) {
        std::cerr << "Exception during window creation: " << e.what() << std::endl;
        throw;
    }
}

void SetupGlobalEventHandling() {
    std::cout << "=== Setting Up Cross-Platform Event Handling ===" << std::endl;

    // Set up global event handler using cross-platform API
    // This works the same way on all platforms
    theApp->SetGlobalEventHandler([this](const UCEvent& event) -> bool {
        // Log important global events
        if (event.type == UCEventType::KeyDown && event.virtualKey == UCKeys::F4 && event.alt) {
            std::cout << "GLOBAL ALT+F4 - REQUESTING EXIT!" << std::endl;
            theApp->Exit();
            return true; // Consume the event
        }

        // Let other events pass through
        return false;
    });

    std::cout << "Cross-platform event handler configured" << std::endl;
}

void Run() {
    std::cout << "=== Starting Pure Cross-Platform Application ===" << std::endl;

    if (!appInitialized || !mainWindow) {
        std::cerr << "Cannot run: not properly initialized" << std::endl;
        return;
    }

    try {
        // Show the window using cross-platform API
        std::cout << "Showing window..." << std::endl;
        mainWindow->Show();

        std::cout << "========================================" << std::endl;
        std::cout << "=== CROSS-PLATFORM DEMO READY! ===" << std::endl;
        std::cout << "=== CLICK BUTTONS TO TEST! ===" << std::endl;
        std::cout << "=== PRESS ESC OR ALT+F4 TO EXIT! ===" << std::endl;
        std::cout << "========================================" << std::endl;

        // Use the cross-platform main loop - this handles ALL platform-specific details
        // No X11, Win32, or Cocoa code needed anywhere!
        std::cout << "Starting cross-platform main loop..." << std::endl;

        // This single call handles everything:
```

```
        // - Event processing (X11/Win32/Cocoa)
        // - Window management
        // - Rendering
        // - Frame rate control
        // - Platform-specific cleanup
        theApp->Run();

        std::cout << "Cross-platform main loop completed" << std::endl;

    } catch (const std::exception& e) {
        std::cerr << "Error during cross-platform application run: " << e.what() << std::endl;
    }

    std::cout << "=== Pure Cross-Platform Application Complete ===" << std::endl;
}

~PureCrossPlatformApp() {
    std::cout << "=== Cleaning up Pure Cross-Platform App ===" << std::endl;

    // Clean shutdown using cross-platform APIs
    if (mainWindow) {
        mainWindow->Close();
        mainWindow.reset();
    }

    if (theApp) {
        theApp->Exit();
        theApp.reset();
    }
}
};

// ===== MAIN FUNCTION - COMPLETELY CROSS-PLATFORM =====
int main() {
    std::cout << "=====================================================" << std::endl;
    std::cout << "===       UltraCanvas Pure Cross-Platform Demo    ===" << std::endl;
    std::cout << "===            NO PLATFORM CODE                 ===" << std::endl;
    std::cout << "===      Same Source Runs on Any Platform!      ===" << std::endl;
    std::cout << "=====================================================" << std::endl;

    try {
        // This entire block is completely cross-platform
        // Same code compiles and runs on Linux, Windows, macOS, etc.
        {
            PureCrossPlatformApp app;
            app.Run();

            std::cout << "Application completed successfully" << std::endl;
        }

        std::cout << "=====================================================" << std::endl;
        std::cout << "===    Cross-Platform Demo Completed Successfully ===" << std::endl;
        std::cout << "=====================================================" << std::endl;

        return 0;
```

```
    } catch (const std::exception& e) {
        std::cerr << "=== CROSS-PLATFORM APPLICATION ERROR ===" << std::endl;
        std::cerr << "Error: " << e.what() << std::endl;
        std::cerr << "========================================" << std::endl;
        return -1;
    }
}
```