Subject: example proposal particularly how the RichEdit is working
Posted by dodobar on Sun, 15 Jun 2025 07:51:46 GMT
View Forum Message <> Reply to Message

```
/**
 * @file main.cpp
 * @brief A minimal, self-contained example of a custom LogView control in U++.
 *
 * This application demonstrates how to create a specialized "Facade" class by inheriting
 * from RichEdit to produce a clean, easy-to-use, and reusable logging component.
 * It's designed to be a clear, practical example for community forums, showing how
 * to abstract away complex setup and formatting.
 */
#include <CtrlLib/CtrlLib.h>
#include <RichEdit/RichEdit.h>

using namespace Upp;

//================================================================================
/**
 * @class LogView
 * @brief A specialized RichEdit control designed for displaying formatted log output.
 *
 * This class encapsulates all the setup and QTF formatting logic required to create
 * a clean, readable, read-only log display. It provides a simple API for appending
 * styled text, abstracting away the complexities of the underlying RichEdit control.
 */
//================================================================================
struct LogView : RichEdit {
public:
    /// Defines the visual style for a log entry.
    enum LogStyle { LOG_NORMAL, LOG_HEADER, LOG_SUCCESS, LOG_ERROR };

    /**
     * @brief Constructor that applies sensible defaults for a log control.
     * Sets the control to be read-only, hides the ruler and non-printing characters,
     * and initializes the font.
     */
    LogView() {
        SetReadOnly();
        NoRuler();
        ShowCodes(Null);
        qtf_log_buffer = "[A1] "; // Initialize with 8pt Arial font.
    }
```

```
    /**
     * @brief Appends a styled line of text to the log.
     * @param text The string to be logged.
     * @param style The visual style to apply (e.g., HEADER, SUCCESS).
     * @param newline If true, a newline character is appended.
     */
    void Log(const String& text, LogStyle style = LOG_NORMAL, bool newline = true) {
        String qtf_text = text;

        switch(style) {
            case LOG_HEADER:  qtf_text = "[* " + qtf_text + "]"; break;
            case LOG_SUCCESS: qtf_text = "[@g " + qtf_text + "]"; break;
            case LOG_ERROR:   qtf_text = "[*@r " + qtf_text + "]"; break;
            case LOG_NORMAL:
            default:
                break;
        }

        qtf_log_buffer.Cat(qtf_text);
        if(newline) qtf_log_buffer.Cat("&");

        SetQTF(qtf_log_buffer);
        Move(GetLength());
        Ctrl::ProcessEvents(); // Keep UI responsive during long operations.
    }

    /// @brief Adds a horizontal rule to the log for visual separation.
    void AddSeparator() {
        Log("[--]", LOG_NORMAL);
    }

private:
    /// The internal QTF string that acts as the data model for the control.
    String qtf_log_buffer;
};


//================================================================================
======================
/**
 * @class ExampleRunner
 * @brief The main application window that hosts and demonstrates the LogView.
 */
//================================================================================
======================
struct ExampleRunner : TopWindow {
    typedef ExampleRunner CLASSNAME;
```

```
Splitter      splitter;
LogView       logDisplay;     ///< Our custom control for displaying detailed output.
Button        actionButton;   ///< The main button, used for cancelling or closing.

bool          tasks_cancelled = false; ///< Flag to gracefully stop the process.

/// @brief Logs a simple message to the standard console output.
void ConsoleLog(const String& text) {
    Cout() << text;
}

// --- Example Tasks ---
// These simple functions demonstrate using the LogView.

void Test_TaskA() {
    logDisplay.Log("Running: Task A", LogView::LOG_HEADER);
    logDisplay.Log("  Performing some work...");
    Sleep(500); // Simulate work
    logDisplay.Log("  -> PASSED", LogView::LOG_SUCCESS);
}

void Test_TaskB() {
    logDisplay.Log("Running: Task B", LogView::LOG_HEADER);
    logDisplay.Log("  This one is much faster.");
    Sleep(100); // Simulate work
    logDisplay.Log("  -> PASSED", LogView::LOG_SUCCESS);
}

void Test_AFailingTask() {
    logDisplay.Log("Running: A Failing Task", LogView::LOG_HEADER);
    logDisplay.Log("  Oh no, something went wrong.");
    Sleep(300); // Simulate work
    logDisplay.Log("  -> FAILED", LogView::LOG_ERROR);
}

/// @brief A macro to simplify running a task and logging its console status.
#define RUN_TASK(TASK_NAME) \
    if(tasks_cancelled) break; \
    ConsoleLog("Running: " #TASK_NAME "... "); \
    TASK_NAME(); \
    if(tasks_cancelled) break; \
    ConsoleLog("DONE\n"); \
    logDisplay.AddSeparator();

/**
 * @brief Executes all defined example tasks in sequence.
 */
void RunAllTasks() {
```

```
    ConsoleLog("=======================================\n");
    ConsoleLog("  Running Example Tasks\n");
    ConsoleLog("=======================================\n");
    logDisplay.Log("LogView Example Suite", LogView::LOG_HEADER);
    logDisplay.Log("");

    do {
        RUN_TASK(Test_TaskA);
        RUN_TASK(Test_TaskB);
        RUN_TASK(Test_AFailingTask);
    } while(0);

    logDisplay.Log("");
    if(tasks_cancelled) {
        ConsoleLog("\n** Tasks Cancelled By User **\n");
        logDisplay.Log("Tasks Cancelled By User", LogView::LOG_ERROR);
    } else {
        ConsoleLog("=======================================\n");
        ConsoleLog("  Tasks Finished\n");
        ConsoleLog("=======================================\n");
        logDisplay.Log("All tasks finished.", LogView::LOG_SUCCESS);
    }

    actionButton.SetLabel("Close");
    actionButton.Enable();
    actionButton <<= callback(this, &TopWindow::Close);
    Title("LogView Example - Finished");
}

/// @brief Sets the cancellation flag and updates the GUI to reflect it.
void CancelTasks() {
    tasks_cancelled = true;
    actionButton.Disable();
    actionButton.SetLabel("Cancelling...");
}

/// @brief Constructor for the main application window.
ExampleRunner() {
    Title("LogView Example - Running...").Sizeable().Zoomable();
    SetRect(0, 0, 700, 400);

    splitter.Vert(logDisplay, actionButton);
    splitter.SetPos(9000); // 90% for log, 10% for button
    Add(splitter.SizePos());

    actionButton.SetLabel("Cancel");
    actionButton <<= THISBACK(CancelTasks);
```

```
        // Schedule the tasks to run after the window is created and displayed.
        PostCallback(THISBACK(RunAllTasks));
    }
};

/**
 * @brief The main entry point for the U++ GUI application.
 */
GUI_APP_MAIN
{
    StdLogSetup(LOG_COUT);
    ExampleRunner().Run();
}
```