Subject: help for new users skylark - GUI
Posted by BetoValle on Mon, 27 Oct 2025 15:44:25 GMT
View Forum Message <> Reply to Message

It allows you to:

Quickly create web servers and RESTful routes inside your U++ applications.
Serve HTML pages, handle URL parameters, and manage HTTP requests/responses.
Integrate seamlessly with U++ GUI apps, including threaded applications.
Run multiple routes easily using simple macros like SKYLARK(RouteName, "pattern").
In short, Skylark makes it easy to add web capabilities to U++ applications without requiring heavy external web frameworks.

Introduction

As the future of software development increasingly focuses on web-based applications, learning how to integrate HTTP servers with graphical user interfaces and thread-safe operations is becoming essential.

This demo demonstrates a complete and safe way to run a Skylark HTTP server within a U++ GUI application, using threads and an atomic flag to manage server state.

I believe this post will be very helpful for new Ultimate++ users, as it provides a practical, working example of how to:

Start and stop a web server from a GUI.

Keep the GUI responsive using background threads.

Safely manage server state using std::atomic<bool>.

Handle web requests using Skylark routes.

By following this demo, beginners can quickly learn how to combine web functionality with desktop applications in U++, which is an essential skill for modern cross-platform development.

```
// ======================
// Title: Skylark + GUI + Threads + Atomic Flag Example
// Description: Demonstrates a thread-safe Skylark HTTP server control with GUI.
// Tags: Skylark, Threads, Atomic, U++, Ultimate++
// ======================

/* in the first icon of the layout file "Edit as text"

LAYOUT(theLay, 400, 200)
 ITEM(Upp::Button, para, SetLabel(t_("parar")).LeftPosZ(192, 40).TopPosZ(144, 28))
```

```
 ITEM(Upp::Button, inicia, SetLabel(t_("iniciar")).LeftPosZ(92, 40).TopPosZ(144, 28))
 ITEM(Upp::Label, status, LeftPosZ(168, 40).TopPosZ(60, 19))
 ITEM(Upp::Button, encerrar, SetLabel(t_("fechar")).LeftPosZ(308, 40).TopPosZ(144, 28))
END_LAYOUT

*/

#include <CtrlLib/CtrlLib.h>
#include <Skylark/Skylark.h>
#include <atomic>  // Needed for std::atomic

using namespace Upp;

#define LAYOUTFILE <skyGUI/the.lay>
#include <CtrlCore/lay.h>

// ======================
// GuiApp: main GUI class
// ======================
class GuiApp : public WiththeLay<TopWindow> {
public:
    typedef GuiApp CLASSNAME;

    GuiApp();

    void Message(const String& s) { Title(s); }  // updates window title
    void paralizar();  // stops the Skylark server
    void fecha();      // closes application
    void iniciar();    // starts the Skylark server in a separate thread

private:
    // Atomic flag to control server state safely between threads
    std::atomic<bool> servidorRodando{false};
};

GuiApp& GlobalGui();

// ======================
// Skylark route handlers
// ======================

SKYLARK(HomePage, "")
{
    // Example home page
    PostCallback(callback1(&GlobalGui(), &GuiApp::Message, "Homepage"));
    http << "<html><body>Hello world!</body></html>";
}
```

```
SKYLARK(Param, "*/param")
{
    PostCallback(callback1(&GlobalGui(), &GuiApp::Message, "Parameter: " + http[0]));
    http << "<html><body>Parameter: " << http[0] << "</html></body>";
}

SKYLARK(Params, "params/**")
{
    http << "<html><body>";
    String params("Parameters: ");
    for(int i = 0; i < http.GetParamCount(); i++) {
        params += http[i] + " ";
    }
    PostCallback(callback1(&GlobalGui(), &GuiApp::Message, params));
    http << params << "</html></body>";
}

SKYLARK(CatchAll, "**") { http.Redirect(HomePage); }

SKYLARK(Favicon, "/favicon.ico")
{
    http.ContentType("image/png") << LoadFile(GetDataFile("favicon.png"));
}

// ======================
// MyApp: Skylark server application
// ======================
class MyApp : public SkylarkApp {
    typedef MyApp CLASSNAME;

public:
    MyApp()
    {
        root = "";

#ifdef _DEBUG
        prefork = 0;
        use_caching = false;
        StdLogSetup(LOG_FILE | LOG_COUT);
        Ini::skylark_log = true;
#endif
    }

    ~MyApp()
    {
        Quit();
        LOG("quit");
    }
```

```
};

// ======================
// GuiApp: Thread-safe server control
// ======================

void GuiApp::paralizar()
{
    // Atomic read: check if server is running
    if (!servidorRodando.load()) {
        Message("Servidor já parado."); // protection
        return;
    }

    Message("Parando servidor Skylark...");
    SkylarkApp::TheApp().Quit(); // request Skylark server shutdown

    // Optional: wait for server thread to update the flag
    // while(servidorRodando.load()) Thread::Sleep(50);

    Message("Servidor parado.");
}

void GuiApp::fecha()
{
    Close();
    SkylarkApp::TheApp().Quit(); // stop Skylark server
    Thread::ShutdownThreads();   // shutdown U++ thread subsystem
    Exit(0);                     // exit application

/*
When it is safe to use Exit(0)
When you want to terminate the entire application immediately (GUI + server + threads).
When you have already ensured that all main resources have been cleaned up
manually (Quit(), ShutdownThreads(), etc.).
In situations where the main GUI loop (Run()) cannot naturally exit.
Typical above example!
In this case, everything has already been cleaned up before Exit.

When not to use Exit(0):
Inside normal functions, if the goal is just to return to the point
of GUI_APP_MAIN -- in that case, Close() is enough.
Inside destructors -- it may cause partial destruction.
If other threads are still running (without proper control).
*/


}
```

```
void GuiApp::iniciar()
{
    // Atomic read: check if server is already running
    if (servidorRodando.load()) {
        Message("Servidor já iniciado."); // protection
        return;
    }

    // Atomic write: mark server as running
    servidorRodando.store(true);

    // Start server in a separate thread
    Thread::Start([=] {
        // Update GUI to indicate server started
        PostCallback(callback1(this, &GuiApp::Message, "Servidor iniciado."));

        SkylarkApp::TheApp().Run(); // blocking server loop

        // Atomic write: mark server as stopped when loop ends
        servidorRodando.store(false);

        // Update GUI to indicate server stopped
        PostCallback(callback1(this, &GuiApp::Message, "Servidor parado."));
    });
}

// ======================
// GuiApp constructor: setup GUI and callbacks
// ======================
GuiApp::GuiApp()
{
    CtrlLayout(*this, "Skylark GUI Demo");

    LOG("GUI initialized");

    para << [=] { paralizar(); }; // stop button
    encerrar << [=] { fecha(); }; // close button
    inicia << [=] { iniciar(); };  // start button

    WhenClose = [=] { fecha(); }; // handle window X button
}

// ======================
// Singleton instance of GuiApp
// ======================
GuiApp& GlobalGui()
{
```

```
    static GuiApp app;
    return app;
}

// ======================
// Main entry point
// ======================
GUI_APP_MAIN
{
    MyApp app;
    GlobalGui().Run();

    // Shutdown U++ threads after GUI exits
    Thread::ShutdownThreads();
    LOG("Application shutdown");
}
```

std::atomic<bool>:

Guarantees that servidorRodando is thread-safe.

No mutex required for simple flag check and update.

Threaded server (Thread::Start):

Skylark server runs in a background thread, GUI stays responsive.

Thread-safe GUI updates:

Use PostCallback to update GUI from server thread.

Safe shutdown:

paralizar() uses load() to check flag.

iniciar() uses store() to mark running/stopped.

1- Compile and Run the Demo

Open your project in Ultimate++ / TheIDE.

Copy the demo code provided earlier.

Compile and run it.

You'll see a GUI window with three buttons:

2-Testing the Server in a Browser

Make sure the server is running by clicking Start.

Open a web browser (Chrome, Firefox, etc.).

Access the test URLs:

URL Purpose
http://localhost:8001/ Server home page (HomePage)
http://localhost:8001/param/test Example route with a parameter (Param)
http://localhost:8001/params/a/b/c List of multiple parameters (Params)
http://localhost:8001/anything Any other route redirects to Home (CatchAll)

Observe that the GUI updates the window title showing messages such as:

"Server started."

"Server stopped."

"Homepage" or "Parameter: test" when visiting routes.

3- Testing Thread Safety

Click Start multiple times quickly:

With std::atomic<bool>, the flag prevents the server from starting more than once.

You'll see "Server already started." if you try to start it again.

Click Stop while the server is running:

The server safely shuts down.

"Server stopped." appears in the GUI.

The atomic flag ensures the state is never corrupted even with rapid clicks.

Notice that the GUI remains responsive while the server is running, thanks to the separate thread.

4- Testing Complete Shutdown

Click Close or the window's X button.

The fecha() function:

Stops the Skylark server (Quit()),

Shuts down U++ threads (Thread::ShutdownThreads()),

Closes the application (Exit(0)).

Check the log (LOG) to confirm shutdown:

Application shutdown

std::atomic<bool> is sufficient to control a simple server state, no mutex needed.

PostCallback ensures that messages from the server are safely updated in the GUI.

This demo demonstrates a best practice for beginners: threads + GUI + HTTP server working safely together.

If you want, I can also create a visual flow diagram in English showing how:

[GUI thread] <--atomic--> [Skylark server thread]

with flags servidorRodando = true/false and PostCallback updates.
It's perfect for explaining the flow to beginners on the forum.

Thanks!
