#include "flutter\_window.h"

#include <optional>

#include "flutter/generated\_plugin\_registrant.h"

FlutterWindow::FlutterWindow(const flutter::DartProject& project)

: project\_(project) {}

FlutterWindow::~FlutterWindow() {}

bool FlutterWindow::OnCreate() {

if (!Win32Window::OnCreate()) {

return false;

}

RECT frame = GetClientArea();

// The size here must match the window dimensions to avoid unnecessary surface

// creation / destruction in the startup path.

flutter\_controller\_ = std::make\_unique<flutter::FlutterViewController>(

frame.right - frame.left, frame.bottom - frame.top, project\_);

// Ensure that basic setup of the controller was successful.

if (!flutter\_controller\_->engine() || !flutter\_controller\_->view()) {

return false;

}

RegisterPlugins(flutter\_controller\_->engine());

SetChildContent(flutter\_controller\_->view()->GetNativeWindow());

flutter\_controller\_->engine()->SetNextFrameCallback([&]() {

this->Show();

});

return true;

}

void FlutterWindow::OnDestroy() {

if (flutter\_controller\_) {

flutter\_controller\_ = nullptr;

}

Win32Window::OnDestroy();

}

LRESULT

FlutterWindow::MessageHandler(HWND hwnd, UINT const message,

WPARAM const wparam,

LPARAM const lparam) noexcept {

// Give Flutter, including plugins, an opportunity to handle window messages.

if (flutter\_controller\_) {

std::optional<LRESULT> result =

flutter\_controller\_->HandleTopLevelWindowProc(hwnd, message, wparam,

lparam);

if (result) {

return \*result;

}

}

switch (message) {

case WM\_FONTCHANGE:

flutter\_controller\_->engine()->ReloadSystemFonts();

break;

}

return Win32Window::MessageHandler(hwnd, message, wparam, lparam);

}