// Copyright 2013 The Flutter Authors. All rights reserved.

// Use of this source code is governed by a BSD-style license that can be

// found in the LICENSE file.

#ifndef FLUTTER\_SHELL\_PLATFORM\_WINDOWS\_CLIENT\_WRAPPER\_INCLUDE\_FLUTTER\_PLUGIN\_REGISTRAR\_WINDOWS\_H\_

#define FLUTTER\_SHELL\_PLATFORM\_WINDOWS\_CLIENT\_WRAPPER\_INCLUDE\_FLUTTER\_PLUGIN\_REGISTRAR\_WINDOWS\_H\_

#include <flutter\_windows.h>

#include <windows.h>

#include <memory>

#include <optional>

#include "flutter\_view.h"

#include "plugin\_registrar.h"

namespace flutter {

// A delegate callback for WindowProc delegation.

//

// Implementations should return a value only if they have handled the message

// and want to stop all further handling.

using WindowProcDelegate = std::function<std::optional<

LRESULT>(HWND hwnd, UINT message, WPARAM wparam, LPARAM lparam)>;

// An extension to PluginRegistrar providing access to Windows-specific

// functionality.

class PluginRegistrarWindows : public PluginRegistrar {

public:

// Creates a new PluginRegistrar. |core\_registrar| and the messenger it

// provides must remain valid as long as this object exists.

explicit PluginRegistrarWindows(

FlutterDesktopPluginRegistrarRef core\_registrar)

: PluginRegistrar(core\_registrar) {

FlutterDesktopViewRef implicit\_view =

FlutterDesktopPluginRegistrarGetView(core\_registrar);

if (implicit\_view) {

implicit\_view\_ = std::make\_unique<FlutterView>(implicit\_view);

}

}

virtual ~PluginRegistrarWindows() {

// Must be the first call.

ClearPlugins();

// Explicitly cleared to facilitate destruction order testing.

implicit\_view\_.reset();

}

// Prevent copying.

PluginRegistrarWindows(PluginRegistrarWindows const&) = delete;

PluginRegistrarWindows& operator=(PluginRegistrarWindows const&) = delete;

// Returns the implicit view, or nullptr if there is no implicit view.

//

// See:

// https://api.flutter.dev/flutter/dart-ui/PlatformDispatcher/implicitView.html

//

// DEPRECATED: Use |GetViewById| instead.

FlutterView\* GetView() { return implicit\_view\_.get(); }

// Returns the view with the given ID, or nullptr if the view does not exist.

//

// Destroying the shared pointer destroys the reference to the view; it does

// not destroy the underlying view.

std::shared\_ptr<FlutterView> GetViewById(FlutterViewId view\_id) const {

FlutterDesktopViewRef view =

FlutterDesktopPluginRegistrarGetViewById(registrar(), view\_id);

if (!view) {

return nullptr;

}

return std::make\_shared<FlutterView>(view);

}

// Registers |delegate| to receive WindowProc callbacks for the top-level

// window containing this Flutter instance. Returns an ID that can be used to

// unregister the handler.

//

// Delegates are not guaranteed to be called:

// - The application may choose not to delegate WindowProc calls.

// - If multiple plugins are registered, the first one that returns a value

// from the delegate message will "win", and others will not be called.

// The order of delegate calls is not defined.

//

// Delegates should be implemented as narrowly as possible, only returning

// a value in cases where it's important that other delegates not run, to

// minimize the chances of conflicts between plugins.

int RegisterTopLevelWindowProcDelegate(WindowProcDelegate delegate) {

if (window\_proc\_delegates\_.empty()) {

FlutterDesktopPluginRegistrarRegisterTopLevelWindowProcDelegate(

registrar(), PluginRegistrarWindows::OnTopLevelWindowProc, this);

}

int delegate\_id = next\_window\_proc\_delegate\_id\_++;

window\_proc\_delegates\_.emplace(delegate\_id, std::move(delegate));

return delegate\_id;

}

// Unregisters a previously registered delegate.

void UnregisterTopLevelWindowProcDelegate(int proc\_id) {

window\_proc\_delegates\_.erase(proc\_id);

if (window\_proc\_delegates\_.empty()) {

FlutterDesktopPluginRegistrarUnregisterTopLevelWindowProcDelegate(

registrar(), PluginRegistrarWindows::OnTopLevelWindowProc);

}

}

private:

// A FlutterDesktopWindowProcCallback implementation that forwards back to

// a PluginRegistarWindows instance provided as |user\_data|.

static bool OnTopLevelWindowProc(HWND hwnd,

UINT message,

WPARAM wparam,

LPARAM lparam,

void\* user\_data,

LRESULT\* result) {

const auto\* registrar = static\_cast<PluginRegistrarWindows\*>(user\_data);

std::optional optional\_result = registrar->CallTopLevelWindowProcDelegates(

hwnd, message, wparam, lparam);

if (optional\_result) {

\*result = \*optional\_result;

}

return optional\_result.has\_value();

}

std::optional<LRESULT> CallTopLevelWindowProcDelegates(HWND hwnd,

UINT message,

WPARAM wparam,

LPARAM lparam) const {

std::optional<LRESULT> result;

for (const auto& pair : window\_proc\_delegates\_) {

result = pair.second(hwnd, message, wparam, lparam);

// Stop as soon as any delegate indicates that it has handled the message.

if (result) {

break;

}

}

return result;

}

// The associated FlutterView, if any.

std::unique\_ptr<FlutterView> implicit\_view\_;

// The next ID to return from RegisterWindowProcDelegate.

int next\_window\_proc\_delegate\_id\_ = 1;

std::map<int, WindowProcDelegate> window\_proc\_delegates\_;

};

} // namespace flutter

#endif // FLUTTER\_SHELL\_PLATFORM\_WINDOWS\_CLIENT\_WRAPPER\_INCLUDE\_FLUTTER\_PLUGIN\_REGISTRAR\_WINDOWS\_H\_