// 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.

#include "include/flutter/flutter\_engine.h"

#include <algorithm>

#include <iostream>

#include "binary\_messenger\_impl.h"

namespace flutter {

FlutterEngine::FlutterEngine(const DartProject& project) {

FlutterDesktopEngineProperties c\_engine\_properties = {};

c\_engine\_properties.assets\_path = project.assets\_path().c\_str();

c\_engine\_properties.icu\_data\_path = project.icu\_data\_path().c\_str();

c\_engine\_properties.aot\_library\_path = project.aot\_library\_path().c\_str();

c\_engine\_properties.dart\_entrypoint = project.dart\_entrypoint().c\_str();

const std::vector<std::string>& entrypoint\_args =

project.dart\_entrypoint\_arguments();

std::vector<const char\*> entrypoint\_argv;

std::transform(

entrypoint\_args.begin(), entrypoint\_args.end(),

std::back\_inserter(entrypoint\_argv),

[](const std::string& arg) -> const char\* { return arg.c\_str(); });

c\_engine\_properties.dart\_entrypoint\_argc =

static\_cast<int>(entrypoint\_argv.size());

c\_engine\_properties.dart\_entrypoint\_argv =

entrypoint\_argv.empty() ? nullptr : entrypoint\_argv.data();

engine\_ = FlutterDesktopEngineCreate(&c\_engine\_properties);

auto core\_messenger = FlutterDesktopEngineGetMessenger(engine\_);

messenger\_ = std::make\_unique<BinaryMessengerImpl>(core\_messenger);

}

FlutterEngine::~FlutterEngine() {

ShutDown();

}

bool FlutterEngine::Run() {

return Run(nullptr);

}

bool FlutterEngine::Run(const char\* entry\_point) {

if (!engine\_) {

std::cerr << "Cannot run an engine that failed creation." << std::endl;

return false;

}

if (run\_succeeded\_) {

std::cerr << "Cannot run an engine more than once." << std::endl;

return false;

}

bool run\_succeeded = FlutterDesktopEngineRun(engine\_, entry\_point);

if (!run\_succeeded) {

std::cerr << "Failed to start engine." << std::endl;

}

run\_succeeded\_ = true;

return run\_succeeded;

}

void FlutterEngine::ShutDown() {

if (engine\_ && owns\_engine\_) {

FlutterDesktopEngineDestroy(engine\_);

}

engine\_ = nullptr;

}

std::chrono::nanoseconds FlutterEngine::ProcessMessages() {

return std::chrono::nanoseconds(FlutterDesktopEngineProcessMessages(engine\_));

}

void FlutterEngine::ReloadSystemFonts() {

FlutterDesktopEngineReloadSystemFonts(engine\_);

}

FlutterDesktopPluginRegistrarRef FlutterEngine::GetRegistrarForPlugin(

const std::string& plugin\_name) {

if (!engine\_) {

std::cerr << "Cannot get plugin registrar on an engine that isn't running; "

"call Run first."

<< std::endl;

return nullptr;

}

return FlutterDesktopEngineGetPluginRegistrar(engine\_, plugin\_name.c\_str());

}

void FlutterEngine::SetNextFrameCallback(std::function<void()> callback) {

next\_frame\_callback\_ = std::move(callback);

FlutterDesktopEngineSetNextFrameCallback(

engine\_,

[](void\* user\_data) {

FlutterEngine\* self = static\_cast<FlutterEngine\*>(user\_data);

self->next\_frame\_callback\_();

self->next\_frame\_callback\_ = nullptr;

},

this);

}

std::optional<LRESULT> FlutterEngine::ProcessExternalWindowMessage(

HWND hwnd,

UINT message,

WPARAM wparam,

LPARAM lparam) {

LRESULT result;

if (FlutterDesktopEngineProcessExternalWindowMessage(

engine\_, hwnd, message, wparam, lparam, &result)) {

return result;

}

return std::nullopt;

}

FlutterDesktopEngineRef FlutterEngine::RelinquishEngine() {

owns\_engine\_ = false;

return engine\_;

}

} // namespace flutter