// 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\_COMMON\_CLIENT\_WRAPPER\_INCLUDE\_FLUTTER\_BASIC\_MESSAGE\_CHANNEL\_H\_

#define FLUTTER\_SHELL\_PLATFORM\_COMMON\_CLIENT\_WRAPPER\_INCLUDE\_FLUTTER\_BASIC\_MESSAGE\_CHANNEL\_H\_

#include <iostream>

#include <string>

#include <utility>

#include "binary\_messenger.h"

#include "message\_codec.h"

namespace flutter {

namespace internal {

// Internal helper functions used by BasicMessageChannel and MethodChannel.

// Adjusts the number of messages that will get buffered when sending messages

// to channels that aren't fully set up yet. For example, the engine isn't

// running yet or the channel's message handler isn't set up on the Dart side

// yet.

void ResizeChannel(BinaryMessenger\* messenger, std::string name, int new\_size);

// Defines whether the channel should show warning messages when discarding

// messages due to overflow.

//

// When |warns| is false, the channel is expected to overflow and warning

// messages will not be shown.

void SetChannelWarnsOnOverflow(BinaryMessenger\* messenger,

std::string name,

bool warns);

} // namespace internal

class EncodableValue;

// A message reply callback.

//

// Used for submitting a reply back to a Flutter message sender.

template <typename T>

using MessageReply = std::function<void(const T& reply)>;

// A handler for receiving a message from the Flutter engine.

//

// Implementations must asynchronously call reply exactly once with the reply

// to the message.

template <typename T>

using MessageHandler =

std::function<void(const T& message, const MessageReply<T>& reply)>;

// A channel for communicating with the Flutter engine by sending asynchronous

// messages.

template <typename T = EncodableValue>

class BasicMessageChannel {

public:

// Creates an instance that sends and receives method calls on the channel

// named |name|, encoded with |codec| and dispatched via |messenger|.

BasicMessageChannel(BinaryMessenger\* messenger,

const std::string& name,

const MessageCodec<T>\* codec)

: messenger\_(messenger), name\_(name), codec\_(codec) {}

~BasicMessageChannel() = default;

// Prevent copying.

BasicMessageChannel(BasicMessageChannel const&) = delete;

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

// Sends a message to the Flutter engine on this channel.

void Send(const T& message) {

std::unique\_ptr<std::vector<uint8\_t>> raw\_message =

codec\_->EncodeMessage(message);

messenger\_->Send(name\_, raw\_message->data(), raw\_message->size());

}

// Sends a message to the Flutter engine on this channel expecting a reply.

void Send(const T& message, BinaryReply reply) {

std::unique\_ptr<std::vector<uint8\_t>> raw\_message =

codec\_->EncodeMessage(message);

messenger\_->Send(name\_, raw\_message->data(), raw\_message->size(),

std::move(reply));

}

// Registers a handler that should be called any time a message is

// received on this channel. A null handler will remove any previous handler.

//

// Note that the BasicMessageChannel does not own the handler, and will not

// unregister it on destruction, so the caller is responsible for

// unregistering explicitly if it should no longer be called.

void SetMessageHandler(const MessageHandler<T>& handler) const {

if (!handler) {

messenger\_->SetMessageHandler(name\_, nullptr);

return;

}

const auto\* codec = codec\_;

std::string channel\_name = name\_;

BinaryMessageHandler binary\_handler = [handler, codec, channel\_name](

const uint8\_t\* binary\_message,

const size\_t binary\_message\_size,

const BinaryReply& binary\_reply) {

// Use this channel's codec to decode the message and build a reply

// handler.

std::unique\_ptr<T> message =

codec->DecodeMessage(binary\_message, binary\_message\_size);

if (!message) {

std::cerr << "Unable to decode message on channel " << channel\_name

<< std::endl;

binary\_reply(nullptr, 0);

return;

}

MessageReply<T> unencoded\_reply = [binary\_reply,

codec](const T& unencoded\_response) {

auto binary\_response = codec->EncodeMessage(unencoded\_response);

binary\_reply(binary\_response->data(), binary\_response->size());

};

handler(\*message, std::move(unencoded\_reply));

};

messenger\_->SetMessageHandler(name\_, std::move(binary\_handler));

}

// Adjusts the number of messages that will get buffered when sending messages

// to channels that aren't fully set up yet. For example, the engine isn't

// running yet or the channel's message handler isn't set up on the Dart side

// yet.

void Resize(int new\_size) {

internal::ResizeChannel(messenger\_, name\_, new\_size);

}

// Defines whether the channel should show warning messages when discarding

// messages due to overflow.

//

// When |warns| is false, the channel is expected to overflow and warning

// messages will not be shown.

void SetWarnsOnOverflow(bool warns) {

internal::SetChannelWarnsOnOverflow(messenger\_, name\_, warns);

}

private:

BinaryMessenger\* messenger\_;

std::string name\_;

const MessageCodec<T>\* codec\_;

};

} // namespace flutter

#endif // FLUTTER\_SHELL\_PLATFORM\_COMMON\_CLIENT\_WRAPPER\_INCLUDE\_FLUTTER\_BASIC\_MESSAGE\_CHANNEL\_H\_