// 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\_METHOD\_CODEC\_H\_

#define FLUTTER\_SHELL\_PLATFORM\_COMMON\_CLIENT\_WRAPPER\_INCLUDE\_FLUTTER\_METHOD\_CODEC\_H\_

#include <memory>

#include <string>

#include <vector>

#include "method\_call.h"

#include "method\_result.h"

namespace flutter {

// Translates between a binary message and higher-level method call and

// response/error objects.

template <typename T>

class MethodCodec {

public:

MethodCodec() = default;

virtual ~MethodCodec() = default;

// Prevent copying.

MethodCodec(MethodCodec<T> const&) = delete;

MethodCodec& operator=(MethodCodec<T> const&) = delete;

// Returns the MethodCall encoded in |message|, or nullptr if it cannot be

// decoded.

std::unique\_ptr<MethodCall<T>> DecodeMethodCall(const uint8\_t\* message,

size\_t message\_size) const {

return std::move(DecodeMethodCallInternal(message, message\_size));

}

// Returns the MethodCall encoded in |message|, or nullptr if it cannot be

// decoded.

std::unique\_ptr<MethodCall<T>> DecodeMethodCall(

const std::vector<uint8\_t>& message) const {

size\_t size = message.size();

const uint8\_t\* data = size > 0 ? &message[0] : nullptr;

return std::move(DecodeMethodCallInternal(data, size));

}

// Returns a binary encoding of the given |method\_call|, or nullptr if the

// method call cannot be serialized by this codec.

std::unique\_ptr<std::vector<uint8\_t>> EncodeMethodCall(

const MethodCall<T>& method\_call) const {

return std::move(EncodeMethodCallInternal(method\_call));

}

// Returns a binary encoding of |result|. |result| must be a type supported

// by the codec.

std::unique\_ptr<std::vector<uint8\_t>> EncodeSuccessEnvelope(

const T\* result = nullptr) const {

return std::move(EncodeSuccessEnvelopeInternal(result));

}

// Returns a binary encoding of |error|. The |error\_details| must be a type

// supported by the codec.

std::unique\_ptr<std::vector<uint8\_t>> EncodeErrorEnvelope(

const std::string& error\_code,

const std::string& error\_message = "",

const T\* error\_details = nullptr) const {

return std::move(

EncodeErrorEnvelopeInternal(error\_code, error\_message, error\_details));

}

// Decodes the response envelope encoded in |response|, calling the

// appropriate method on |result|.

//

// Returns false if |response| cannot be decoded. In that case the caller is

// responsible for calling a |result| method.

bool DecodeAndProcessResponseEnvelope(const uint8\_t\* response,

size\_t response\_size,

MethodResult<T>\* result) const {

return DecodeAndProcessResponseEnvelopeInternal(response, response\_size,

result);

}

protected:

// Implementation of the public interface, to be provided by subclasses.

virtual std::unique\_ptr<MethodCall<T>> DecodeMethodCallInternal(

const uint8\_t\* message,

size\_t message\_size) const = 0;

// Implementation of the public interface, to be provided by subclasses.

virtual std::unique\_ptr<std::vector<uint8\_t>> EncodeMethodCallInternal(

const MethodCall<T>& method\_call) const = 0;

// Implementation of the public interface, to be provided by subclasses.

virtual std::unique\_ptr<std::vector<uint8\_t>> EncodeSuccessEnvelopeInternal(

const T\* result) const = 0;

// Implementation of the public interface, to be provided by subclasses.

virtual std::unique\_ptr<std::vector<uint8\_t>> EncodeErrorEnvelopeInternal(

const std::string& error\_code,

const std::string& error\_message,

const T\* error\_details) const = 0;

// Implementation of the public interface, to be provided by subclasses.

virtual bool DecodeAndProcessResponseEnvelopeInternal(

const uint8\_t\* response,

size\_t response\_size,

MethodResult<T>\* result) const = 0;

};

} // namespace flutter

#endif // FLUTTER\_SHELL\_PLATFORM\_COMMON\_CLIENT\_WRAPPER\_INCLUDE\_FLUTTER\_METHOD\_CODEC\_H\_