// 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\_BYTE\_BUFFER\_STREAMS\_H\_

#define FLUTTER\_SHELL\_PLATFORM\_COMMON\_CLIENT\_WRAPPER\_BYTE\_BUFFER\_STREAMS\_H\_

#include <cassert>

#include <cstdint>

#include <cstring>

#include <iostream>

#include <vector>

#include "include/flutter/byte\_streams.h"

namespace flutter {

// Implementation of ByteStreamReader base on a byte array.

class ByteBufferStreamReader : public ByteStreamReader {

public:

// Createa a reader reading from |bytes|, which must have a length of |size|.

// |bytes| must remain valid for the lifetime of this object.

explicit ByteBufferStreamReader(const uint8\_t\* bytes, size\_t size)

: bytes\_(bytes), size\_(size) {}

virtual ~ByteBufferStreamReader() = default;

// |ByteStreamReader|

uint8\_t ReadByte() override {

if (location\_ >= size\_) {

std::cerr << "Invalid read in StandardCodecByteStreamReader" << std::endl;

return 0;

}

return bytes\_[location\_++];

}

// |ByteStreamReader|

void ReadBytes(uint8\_t\* buffer, size\_t length) override {

if (location\_ + length > size\_) {

std::cerr << "Invalid read in StandardCodecByteStreamReader" << std::endl;

return;

}

std::memcpy(buffer, &bytes\_[location\_], length);

location\_ += length;

}

// |ByteStreamReader|

void ReadAlignment(uint8\_t alignment) override {

uint8\_t mod = location\_ % alignment;

if (mod) {

location\_ += alignment - mod;

}

}

private:

// The buffer to read from.

const uint8\_t\* bytes\_;

// The total size of the buffer.

size\_t size\_;

// The current read location.

size\_t location\_ = 0;

};

// Implementation of ByteStreamWriter based on a byte array.

class ByteBufferStreamWriter : public ByteStreamWriter {

public:

// Creates a writer that writes into |buffer|.

// |buffer| must remain valid for the lifetime of this object.

explicit ByteBufferStreamWriter(std::vector<uint8\_t>\* buffer)

: bytes\_(buffer) {

assert(buffer);

}

virtual ~ByteBufferStreamWriter() = default;

// |ByteStreamWriter|

void WriteByte(uint8\_t byte) { bytes\_->push\_back(byte); }

// |ByteStreamWriter|

void WriteBytes(const uint8\_t\* bytes, size\_t length) {

assert(length > 0);

bytes\_->insert(bytes\_->end(), bytes, bytes + length);

}

// |ByteStreamWriter|

void WriteAlignment(uint8\_t alignment) {

uint8\_t mod = bytes\_->size() % alignment;

if (mod) {

for (int i = 0; i < alignment - mod; ++i) {

WriteByte(0);

}

}

}

private:

// The buffer to write to.

std::vector<uint8\_t>\* bytes\_;

};

} // namespace flutter

#endif // FLUTTER\_SHELL\_PLATFORM\_COMMON\_CLIENT\_WRAPPER\_BYTE\_BUFFER\_STREAMS\_H\_