(This wiki page applies to people migrating code written before March 2017. It is unlikely to still be relevant.)

Introduction

Prior to pull requests flutter/flutter#8837 and flutter/engine#3482, both merged on March 17 2017, communication between Flutter app components written in Dart and host platform app components written for Android or iOS was accomplished by sending and receiving string/JSON messages using methods defined in the following types:

- `PlatformMessages` (Dart)
- `FlutterView` (Android)
- `FlutterViewController` (iOS)

These methods have been removed, leaving only methods for sending unencoded binary messages. A new channel concept has been introduced in their place:

- `PlatformMessageChannel`, `PlatformMethodChannel` (Dart)
- `FlutterMessageChannel`, `FlutterMethodChannel` (Android, iOS)

The main goals of the new API are to achieve

- higher-level communication: asynchronous method calls and event streams
- less boilerplate and redundancy
- a more symmetric API across Dart, Android, and iOS

The following sections detail how to port code written against the old API.

Flutter side

With the new API you define in one place the name and type of the channel you need:

```
var fooChannel = new PlatformMessageChannel<String>('foo', const StringCodec());
var barChannel = new PlatformMethodChannel('bar', const JSONMethodCodec());
```

There are four codecs to choose from: binary, string, JSON, and standard. The standard codec employs efficient binary serialization of JSON-like values, supporting also buffers as leaf values (e.g. Dart `TypedData`, Java primitive arrays, Cocoa `NSData`).

Once you have a channel, you can use it in multiple places without repeating the information needed for its creation.

Sending messages to platform

Replace code like this

```
String reply = await PlatformMessages.sendString('foo', myString);
```

with code like this

```
String reply = await fooChannel.send(myString);
```

Receiving messages from platform

Replace code like this

```
PlatformMessages.setStringMessageHandler('foo', (String message) async {
  // do something, then
  return reply;
});
```

with code like this

```
fooChannel.setMessageHandler((String message) async {
  // do something, then
  return reply;
});
```

Invoking platform methods

Replace code like this

```
var arguments = { 'argA': 'hello', 'argB': 42 };
var message = {
  'method': 'someMethod',
  'args': <Map<String, dynamic>>[arguments],
};
dynamic reply = await PlatformMessages.sendJSON('bar', message);
// what about errors?
```

or this

```
var arguments = { 'argA': 'hello', 'argB': 42 };
dynamic reply = await PlatformMessages.invokeMethod(
  'bar',
  'someMethod',
  <Map<String, dynamic>>[arguments],
);
// what about errors?
```

with code like this

```
try {
  dynamic result = await barChannel.invokeMethod(
    'someMethod',
    { 'argA': 'hello', 'argB': 42 },
  );
  // use result
} on PlatformException catch(e) {
  // handle error
}
```

Receiving method calls from platform code

Replace code like this

```
PlatformMessages.setJSONMessageHandler('bar', (dynamic methodCall) async {
  String method = methodCall['method'];
  List arguments = methodCall['args'];
  // handle call then
  return result;
  // but what about errors?
});
```

with code like this

```
barChannel.setMethodCallHandler((MethodCall call) async {
  String method = call.method;
  dynamic arguments = call.arguments;
  // handle call then
  return result;
  // or
  throw new PlatformException(errorCode, anErrorMessage, someDetails);
});
```

See platform_channel for an example.

Android side

Similar to Flutter side, using `FlutterMessageChannel` and `FlutterMethodChannel` from `io.flutter.plugin.common`.

```
FlutterView view = ...
FlutterMessageChannel<String> fooChannel =
  new FlutterMessageChannel<>(view, "foo", StringCodec.INSTANCE);

fooChannel.send(myString);

// or if you need to handle a reply:

fooChannel.send(myString, new ReplyHandler<String>() {
  public void onReply(String reply) {
    // do something with reply
  }
});
```

API documentation. See platform_channel for another example.

iOS side

Similar to Flutter side, using `FlutterMessageChannel` and `FlutterMethodChannel` from `FlutterChannels.h`.

```
FlutterViewController controller = ...
FlutterMessageChannel* fooChannel =
  [FlutterMessageChannel messageChannelWithName:@"foo"
                                binaryMessenger:controller
```

```
                                          codec:[FlutterStringCodec sharedInstance]];

[fooChannel sendMessage:myString];

// or if you need to handle a reply:

[fooChannel sendMessage:myString replyHandler:^(id reply) {
  // do something with (NSString*)reply
}];
```

API documentation. See platform_channel for another example.