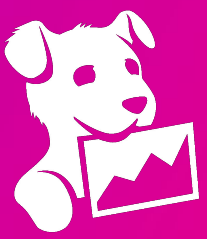
Binding Native Plugins with Flutter and Dart

Fluttercon USA 2025



DATADOG

Jeff Ward

Senior Software Engineer, RUM SDKs
Datadog

Plugin Development is all about working with native libraries.

Interacting with the Native Platform - Why?

The Flutter package ecosystem is great…

… but at some point, you will at some point run into something pub doesn't support.

Leverage new libraries.

New libraries come out all the time, and Flutter support is not their priority

Work with already cross platform C libraries.

How?

In the beginning…

All we had was Method Channels.

Using Method Channels - Dart

```dart
class _MethodChannelsScreenState extends State<MethodChannelsScreen> {
  static const methodChannel = MethodChannel('binding_demo');

  final rnmCharacters = <List<dynamic>>[ ... ];

  void callMethodChannel() async {
    final result = await methodChannel.invokeMethod('methodCallChannelTest', {
        'characters': rnmCharacters,
    });

    // Work with result
  }
```

Official Docs: https://docs.flutter.dev/platform-integration/platform-channels

Method Channels - Android

```
class MainActivity : FlutterActivity() {
  override fun configureFlutterEngine(flutterEngine: FlutterEngine) {
    MethodChannel(flutterEngine.dartExecutor.binaryMessenger, "binding_demo")
      .setMethodCallHandler( ::methodCallHandler)
  }

  fun methodCallHandler(call: MethodCall, result: MethodChannel.Result) {
    when (call.method) {
      "methodCallChannelTest" → {
        result.success("OK")
      }
    }
  }
}
```

Official Docs: https://docs.flutter.dev/platform-integration/platform-channels

Method Channels - iOS

```swift
@main
@objc class AppDelegate: FlutterAppDelegate {
  override func application(
      _ application: UIApplication,
      didFinishLaunchingWithOptions launchOptions [UIApplication.LaunchOptionsKey: Any]?
  ) → Bool {
    let controller = window?.rootViewController as! FlutterViewController
    let channel = FlutterMethodChannel(
      name: "binding_demo",
      binaryMessenger: controller.binaryMessenger
    )

    channel.setMethodCallHandler { (call: FlutterMethodCall, result: @escaping FlutterResult) in
      if call.method == "methodCallChannelTest" {
        result("OK")
      } else {
        result(FlutterMethodNotImplemented)
      }
  }
}
```

Official Docs: https://docs.flutter.dev/platform-integration/platform-channels

A Ridiculous Example

Sending a LOT of data over a Method Channel

Ridiculous Example

- Take the entire Rick and Morty Characters API response…
- … send it over a method channel
- 410KB of JSON data


```
    "https://rickandmortyapi.com/api/episode/48",
    "https://rickandmortyapi.com/api/episode/49",
    "https://rickandmortyapi.com/api/episode/50",
    "https://rickandmortyapi.com/api/episode/51"
  ],
  "url": "https://rickandmortyapi.com/api/character/5",
  "created": "2017-11-04T19:26:56.301Z"
},
{
  "id": 6,
  "name": "Abadango Cluster Princess",
  "status": "Alive",
  "species": "Alien",
  "type": "",
  "gender": "Female",
  "origin": {
    "name": "Abadango",
    "url": "https://rickandmortyapi.com/api/location/2"
  },
  "location": {
    "name": "Abadango",
    "url": "https://rickandmortyapi.com/api/location/2"
  },
  "image": "https://rickandmortyapi.com/api/character/avatar/6.jpeg
  "episode": ["https://rickandmortyapi.com/api/episode/27"],
  "url": "https://rickandmortyapi.com/api/character/6",
  "created": "2017-11-04T19:50:28.250Z"
},
{
  "id": 7,
  "name": "Abradolf Lincler",
  "status": "unknown",
  "species": "Human",
  "type": "Genetic experiment",
  "gender": "Male",
  "origin": {
    "name": "Earth (Replacement Dimension)",
    "url": "https://rickandmortyapi.com/api/location/20"
```

Performance

iOS Performance (iPhone 12)

	Min	Max	Average
invokeMethod	1.29ms	12.83ms	1.91ms
Full Response	10.71ms	46.51ms	12.51ms

Android Performance (Galaxy A31)

	Min	Max	Average
invokeMethod	5.71ms	13.84ms	7.8ms
Full Response	45.39ms	103.83ms	62.73ms

- Get a bad wrap for being “slow”.
- Highly dependent on how much data you’re serializing
- Thread “thunks” / async are your biggest performance bottleneck for responses

Method Channels

Benefits

Easy to abstract to a single “platform” interface

Automatically serialize / deserialize almost any simple Dart type

Great when timing isn’t an issue

Drawbacks

Inherently asynchronous

Thread “thunks” can harm perceived performance

Writing native binding code with assumed contracts

Or rely on code generators like Pigeon

Flutter is looking to encourage more packages to drop them in favor of FFI equivalents

> “The fundamental challenge with method channels is that they can be time consuming to implement and maintain.”

Mariam Hasnany

Flutter Product Manager

Source: https://medium.com/flutter/flutters-path-towards-seamless-interop-4bf7d4579d9a

New Native Communication

Swift and Kotlin FFI generation are still experimental / unstable and part of an early access program

(https://medium.com/@mariam.hasnany/4bf7d4579d9a)

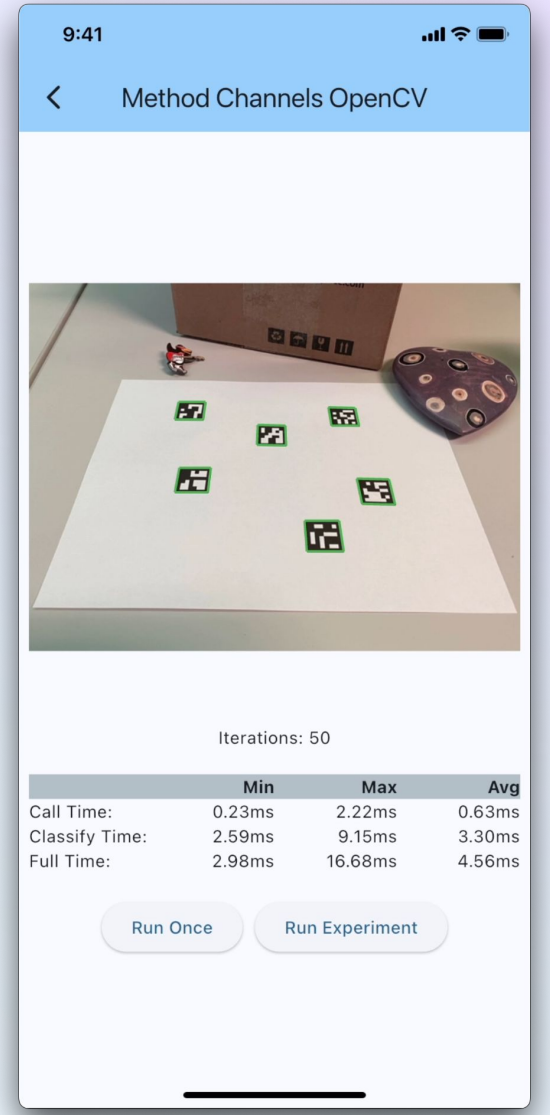
Practical Example

OpenCV Aruco Detector

OpenCV

Detect Arucos (small, AR optimized QR codes) in a static image.

Use a static image to achieve consistent results.



Practical Example

- Time something with an entirely native workload using
  - Method Channels
  - jnigen on Android
  - ffigen to a C wrapper on iOS and Android
- Time
  - `invokeMethod` call
  - Classification time
  - Total response time

Source Code: https://github.com/fuzzybinary/plugin_binding_demo

Caveats

- We’re going to do this the hard way - manually using `ffigen` and `jnigen`
- This is one of the areas that Flutter is evolving VERY quickly
  - `@Native` and native assets
  - New code gen / Pigeon functionality

For more info on some of this newer stuff, check out the talk that was earlier today:

-**Adopting Native Assets For Cross-Platform FFI Plugins** by Simon Binder

Using ffigen

- Add `ffi` to `dependencies`, `ffigen` to `dev_dependencies`
- Create `ffigen.yaml`
- Run ffigen (`dart run ffigen --config ffigen.yaml`)

Example ffigen.yaml

```yaml
name: 'OpenCv'
functions:
  leaf:
    include:
      - '.*'
output: lib/open_cv_c.dart
headers:
  entry-points:
    - 'c_module/opencv_wrapper.h'
```

FFI Example Code

```
class _CFfiScreenState extends State<CFfiScreen> {
  @override
  void initState() {
    super.initState();

    if (Platform.isAndroid) {
      final dylib = DynamicLibrary.open('libopencv_wrapper.so');
      opencv = OpenCv(dylib);
    } else {
      final dylib = DynamicLibrary.executable();
      opencv = OpenCv(dylib);
    }
    opencv.initializeOpenCV();
  }

  // ...
}
```

FFI Example Code

```
void _runTest() {
  using((arena) {
    final imageDataList = _byteData!.buffer.asUint8List();
    final imageData = arena.allocate<Uint8>(imageDataList.length);
    imageData
        .asTypedList(imageDataList.length)
        .setRange(0, imageDataList.length, imageDataList);

    final markerResult = opencv.decodeMarkers(
      imageData,
      _imageAsset!.width,
      _imageAsset!.height,
    );

    _setMarkerCorners(markerResult.ref)
    opencv.freeDecodeResult(markerResult);
  });
}
```

Using jnigen

- Add `jni` to `dependencies`, `jnigen` to `dev_dependencies`
- Create `jnigen.yaml`
- Build your APK (`flutter build apk`)
- Run jnigen (`dart run jnigen --config jnigen.yaml`)

Example jnigen.yaml

```yaml
android_sdk_config:
  add_gradle_deps: true

output:
  dart:
    path: lib/open_cv_jni.dart
    structure: single_file

source_path:
  - 'android/app'
classes:
  - 'org.opencv.core.CvType'
  - 'org.opencv.core.Mat'
  - 'org.opencv.core.Point'
  - 'org.opencv.objdetect.Dictionary'
  - 'org.opencv.objdetect.ArucoDetector'
  - 'org.opencv.objdetect.DetectorParameters'
  - 'org.opencv.objdetect.Objdetect'
```

JNI Example Code

```
Future<void> _runTest() async {
  using((arena) {
    JByteArray data = JByteArray.from(_byteData!.buffer.asUint8List())
      ..releasedBy(arena);

    final imageMat =
        cv.Mat.new$2(
            _imageAsset!.height,
            _imageAsset!.width,
            cv.CvType.CV_8UC4,
          )
          ..releasedBy(arena)
          ..put$8(0, 0, data);

    final markerCornerList = <cv.Mat?>[].toJList(cv.Mat.type)
      ..releasedBy(arena);
    final ids = cv.Mat.new$1()..releasedBy(arena);

    arucoDetector.detectMarkers$1(imageMat, markerCornerList, ids);
    if (markerCornerList.isEmpty) {
      status = 'Failed to find markers.';
      return;
    }

    _setMarkerCorners(markerCornerList, ids);
  });
}
```

Performance - iOS

iOS Performance (iPhone 12) Method Channels

	Min	Max	Average
invokeMethod	0.21ms	1.92ms	0.5ms
Classify Time	2.64ms	11.71ms	3.36ms
Full Response	3.03ms	15.56ms	4.53ms

iOS Performance (iPhone 12) FFI

	Min	Max	Average
Classify Time	2.61ms	7.88ms	3.23ms
Full Response	2.72ms	8.54ms	3.42ms

Performance - Android

Android Performance (Galaxy A31) Method Channel

	Min	Max	Average
invokeMethod	1.63ms	9.64ms	2.89ms
Classify Time	8.33ms	39.32ms	14.88ms
Full Response	13.41ms	59.68ms	24.44ms

Android Performance (Galaxy A31) C FFI

	Min	Max	Average
Classify Time	7.21ms	12.10ms	8.94ms
Full Response	8.52ms	14.07ms	10.25ms

Android Performance (Galaxy A31) jnigen

	Min	Max	Average
Classify Time	8.29ms	34.70ms	14.44ms
Full Response	10.49ms	49.54ms	18.65ms

FFI / JNI

Benefits

Tend to be faster (your mileage may vary)

Synchronous response

Drawbacks

Some Dart objects don’t translate well to C / Java constructs

- For complicated objects, you need your own protocol
- Calling overloaded methods requires $1 or similar.

Can require some very manual translation

C linker behavior can be confusing

LLVM Magic Attributes

- If you’re statically linking C functions, the linker may strip your code, or mangle it.
- Magic LLVM / GCC attributes to prevent it:

```
__attribute__((retain)) __attribute__((visibility("default")))
```

But…

- These all work great for async send and receive….
- What about more complicated scenarios?

What About Callbacks?

What do we mean by “callback”

- Asking to be updated by the library on a regular basis
- Or at least, not in 1:1 call / response, `async` / `await` way.

Callbacks - Method Channels

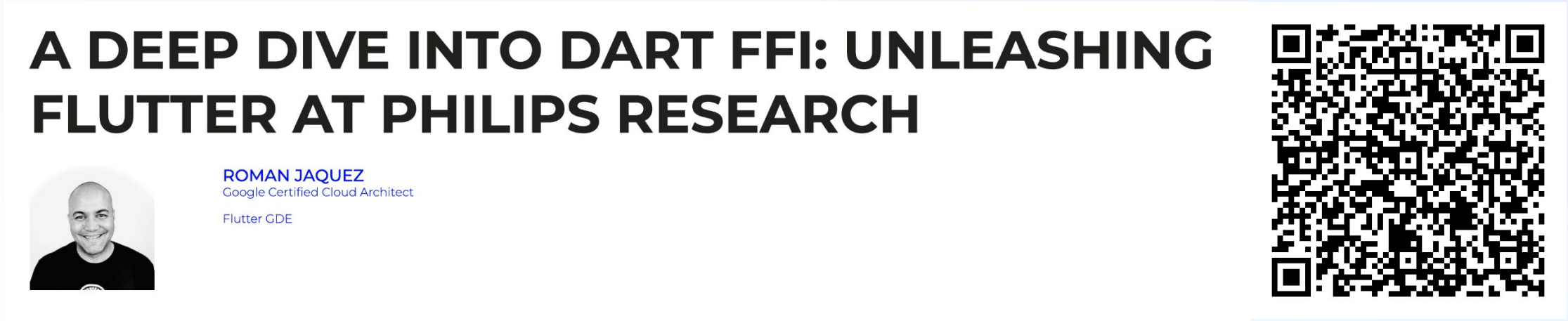
- `MethodChannel.invokeMethod`
  - Don’t have to worry about isolates!
  - Do have to worry about data serialization timing
  - Inherently asynchronous

- Major Drawback
  - Must be called from the main thread
  - Will always execute on your Root isolate

Send / Receive Ports

- An FFI way to perform callbacks
- More complicated to set up
- Can call from any thread into a background isolates no problem
- Inherently asynchronous
- Need to be used from C

Great Talk: A Deep Dive Into Dart FFI



Callbacks - FFI Methods

Dart FFI allows you to create a C Function Pointer for callback purposes

- `Pointer.fromFunction` - Only meant for synchronous callbacks (static functions only)
- `NativeCallable.isolateLocal` - Can only be called from the thread that created it
- `NativeCallable.listener` (since Dart 3.1) - Called from any thread
  - Uses Send / Receive ports behind the scenes
  - Async and can’t return data
- `NativeCallable.blocking` - Called from any thread but blocks waiting on a return value
  - Doesn’t exist yet. Dart SDK issue #54554

More Info: https://dart.dev/interop/objective-c-interop#callbacks-and-multithreading-limitations

Callbacks - Obj-C Delegates

- `ffigen` allows you to implement Objective-C protocols in Dart
- Add the protocol to your `ffigen.yaml`
- Generates 3 implementation functions:
  - `.implement` - Equivalent to `NativeCallable.isolateLocal`
  - `.implementListener` - Equivalent to `NativeCallable.listener`
  - `.implementBlocking` - Can be called from any thread, but blocks the calling thread until the callback is complete.

Callbacks - JNI Methods

jnigen allows you to implement interfaces in Dart

- Can use `.implement` to implement callback classes
- Tries to be smart - uses Send ports when necessary
- `.implement` can’t be used in all cases - need “Proxy Objects”

More Info: https://github.com/dart-archive/jnigen/pull/326

jnigen Callback Proxy Object

```
class BluetoothGattCallbackProxy(val callback: CallbackInterface)
  : android.bluetooth.BluetoothGattCallback() {
    interface CallbackInterface {
        fun onConnectionStateChange(gatt: android.bluetooth.BluetoothGatt, status: Int, newState: Int)
        fun onServicesDiscovered(gatt: android.bluetooth.BluetoothGatt, status: Int)
        fun onPhyUpdate(gatt: BluetoothGatt?, txPhy: Int, rxPhy: Int, status: Int)
        fun onPhyRead(gatt: BluetoothGatt?, txPhy: Int, rxPhy: Int, status: Int)
        /// Etc....
    }

    override fun onPhyUpdate(gatt: BluetoothGatt?, txPhy: Int, rxPhy: Int, status: Int) {
        callback.onPhyUpdate(gatt, txPhy, rxPhy, status)
    }

    override fun onPhyRead(gatt: BluetoothGatt?, txPhy: Int, rxPhy: Int, status: Int) {
        callback.onPhyRead(gatt, txPhy, rxPhy, status)
    }


    override fun onConnectionStateChange(gatt: android.bluetooth.BluetoothGatt, status: Int, newState: Int) {
        callback.onConnectionStateChange(gatt, status, newState)
    }

    // Etc ...
}
```

Callback Example

Native Bluetooth in Dart

Objective-C ffigen.yaml

```yaml
name: CoreBluetooth
description: Bindings for iOS CoreBluetooth
language: objc
output:
  bindings: 'lib/core_bluetooth.dart'
  objc-bindings: 'ios/core_bluetooth.dart.m'
headers:
  entry-points:
    - '/Applications/Xcode-16.2.0.app/Contents/Developer/...../CoreBluetooth.h'
exclude-all-by-default: true
objc-interfaces:
  include:
    - CBCentralManager
    - CBPeripheral
    - CBService
    - CBUUID
    - CBCharacteristic
    - CBDescriptor
    - CBPeripheralManager
objc-protocols:
  include:
    - CBCentralManagerDelegate
    - CBPeripheralDelegate
```

iOS FFI Callback Code

```
void _startScan() {
  _centralDelegate = CBCentralManagerDelegate.implementAsListener(
    centralManagerDidUpdateState_: (centralManager) {
      switch (centralManager.state) {
        case CBManagerState.CBManagerStateUnknown:
          _setConnectionStatus('Manager is in unknown state');
          break;
        case CBManagerState.CBManagerStateResetting:
          _setConnectionStatus('Manager is currently resetting');
          break;
        /// Etc ...
      }
    },
    centralManager_didConnectPeripheral_: (central, peripheral) {
      _setConnectionStatus('Discovering services ... ');
      peripheral.delegate = _createPeripheralDelegate();
      peripheral.discoverServices(null);
    },
    /// Other delegate methods
  );

  _centralManager = CBCentralManager.alloc().initWithDelegate(
    _centralDelegate,
  );
}
```

jni Callback Object Code

```
void _connectDevice(BluetoothDevice device) {
    using((arena) {
      final context = Context.fromReference(Jni.getCachedApplicationContext())
        ..releasedBy(arena);
      final callback = BluetoothGattCallbackProxy$CallbackInterface.implement(
        $BluetoothGattCallbackProxy$CallbackInterface(
          onCharacteristicChanged: (
            bluetoothGatt,
            bluetoothGattCharacteristic,
            bs,
          ) {
            _gotCharacteristicChange(bs);
          },
          onConnectionStateChange: (bluetoothGatt, status, newState) {
            if (newState == BluetoothProfile.STATE_CONNECTED) {
              bluetoothGatt.requestMtu(517);
              _setConnectionStatus('Discovering services...');
              bluetoothGatt.discoverServices();
            }
          },
          /// ETC
        ),
      );

      _setConnectionStatus('Connecting GATT...');
      _connectedDevice = device;
      _connectedGatt = device.connectGatt(context, true, BluetoothGattCallbackProxy(callback));
    });
  }
}
```

So which should I use?

So Which Should I Use?

Method Channels are fine and aren't as slow as people think.

For a C library, using **FFI and ffigen** will give you better responsiveness.

New Bindings should probably try **fiigen and jnigen**.

Synchronous callbacks should definitely use **ffigen and jnigen**.

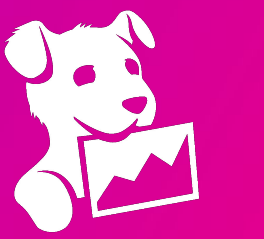
Questions?

github.com/fuzzybinary

fuzzybinary@mastodon.gamedev.place

@fuzzybinary.bsky.social

Source Code: https://github.com/fuzzybinary/plugin_binding_demo



Thank you

github.com/fuzzybinary
fuzzybinary@mastodon.gamedev.place
@fuzzybinary.bsky.social

Source Code: https://github.com/fuzzybinary/plugin_binding_demo