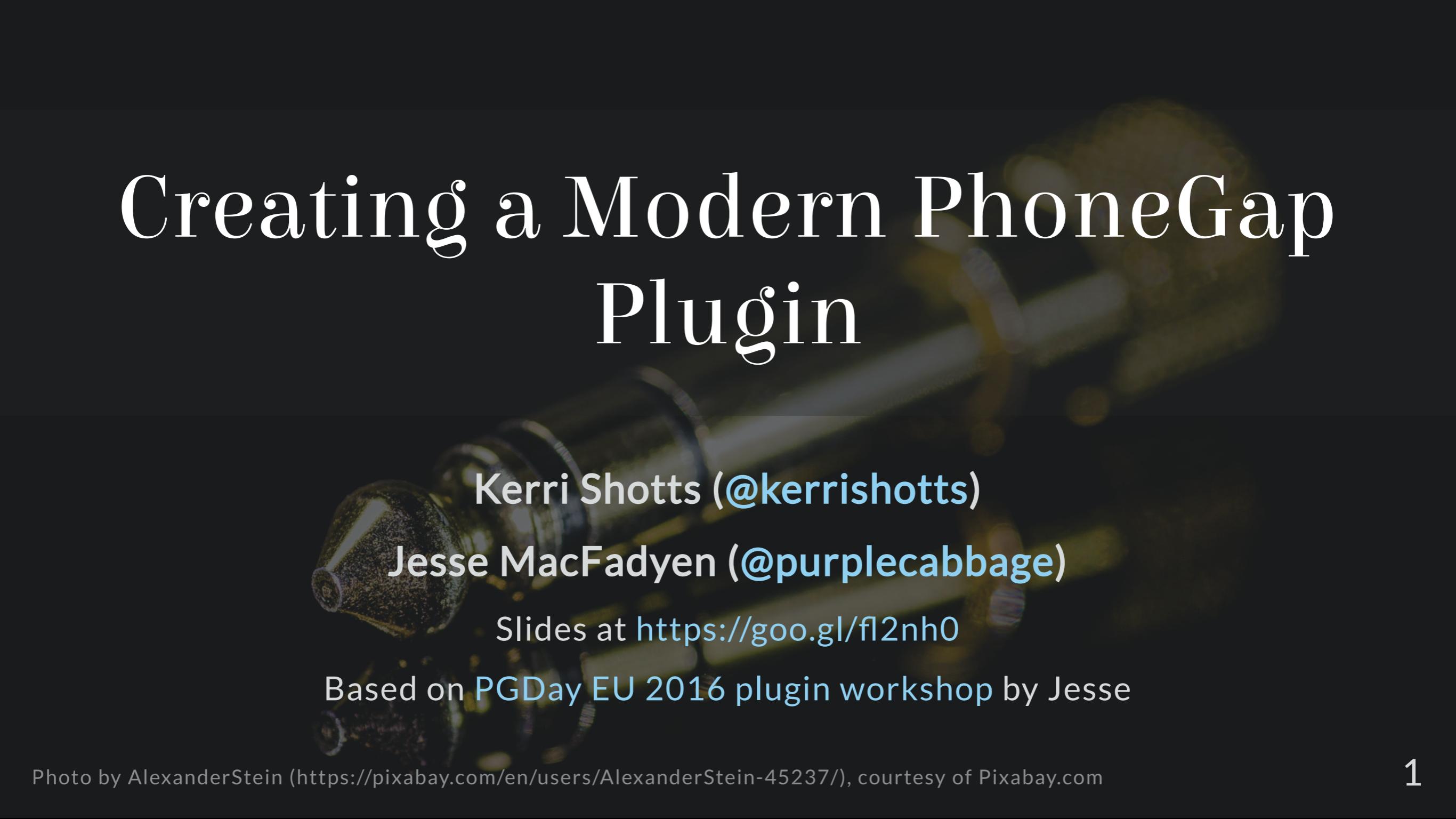


Creating a Modern PhoneGap Plugin



Kerri Shotts (@kerrishotts)

Jesse MacFadyen (@purplecabbage)

Slides at <https://goo.gl/fI2nh0>

Based on PGDay EU 2016 plugin workshop by Jesse

About Kerri

- Used PhoneGap for six+ years
- Author of five books about PhoneGap
- IT Consultant for eight years
- Apache Cordova committer
- One of several moderators:
 - Adobe PhoneGap Forums
 - Google Cordova Group
- @kerrishotts



About Jesse

- PhoneGap Developer since 2008
- Apache Cordova committer
- At Adobe for nearly 6 years now
- @purplecabbage



What is a Cordova Plugin?

noun A mystical collection of machine incantations which grant access to amazing and magical capabilities

ahem...

noun A module consisting of code and settings extending the essential functionality of Cordova with the goal of providing access to device capabilities, enhancing existing capabilities, or improving the developer's workflow

What can plugins do?

- Anything native code can do
- Active in the following contexts:
 - run time
 - build time
 - install time
- Two sources of Plugins
 - Core – used to be built in pre-3.x
 - Community – people like you!

Plugins at Run Time

Full access to the native SDK and device features. Some examples:

- Push Notifications: PhoneGap, Pushwoosh, AeroGear, OneSignal
- Storage Plugins: Native Storage, SQLite, SQLite 2
- Social Plugins: Email, X SocialSharing
- Audio Plugins: DBMeter, Native Audio, Media Picker
- Misc: Barcode Scanner, In App Purchase, Google Maps, Vuforia (AR), Microsoft ACE (native controls), Tesseract (OCR, iOS)
- Creative Cloud: Auth, Asset Browser, Image Editor, Send to Desktop

Plugins at Build Time

Full access to the build-time environment and Cordova project. Some examples:

- Transpile and Bundle ES2015+: Webpack & Transpiler plugin
- Pre-process CSS files (SASS, less, auto-prefixer)
- Check code quality (eslint, tslint, jshint)
- Etc.

Plugins at Install Time

Full access to the Cordova project and environment at install time.
Some ideas:

- Configure the project environment
- Bundle other plugins
- Provide tests for another plugin...
 - `cordova-plugin-test-framework`

Plugin-ception 

The Core Plugins

Core Cordova Plugins (used to be built-in pre-3.x):

battery-status	camera	console
contacts	device	device-motion
device-orientation	dialogs	file
file-transfer	geolocation	globalization
inappbrowser	media	media-capture
network-information	splashscreen	statusbar
vibration	whitelist	

Community Plugins

Developed and supported by the community – like you!

Repository	Plugins
https://cordova.apache.org/plugins	~2,066 plugins & templates (excl. core)
http://www.plugreg.com	~1,592 plugins (excl. core)
http://plugins.telerik.com/cordova	~77 plugins

Managing Plugins

npm

Plugins are typically downloaded from npm:

```
$ cordova plugin add --save cordova-plugin-device
```

```
$ cordova plugin ls # or list  
cordova-plugin-device 1.1.1 "Device"
```

```
$ cordova plugin rm --save cordova-plugin-device # or remove
```

Note: --save persists the plugin to config.xml so that plugins can be easily restored (done at prepare -time)

7.0.0: --save will be the default action in cordova@7.0.0; --nosave will turn it off

Git

Plugins can also be installed from a Git repository.

```
$ cordova plugin add http://github.com/apache/cordova-plugin-device  
$ cordova plugin rm cordova-plugin-device
```

Specify a branch: (useful for testing pre-release/edge plugins):

```
$ cordova plugin add http://github.com/apache/cordova-plugin-device  
    #branch
```

Note: Use the plugin's identifier when removing – not the URL.

Local Filesystem

Or install from the local file system – very useful for plugin development.

```
$ | cordova plugin add --save [--link] /path/to/plugin  
$ | cordova plugin rm --save cordova-plugin-device
```

--link is useful when developing plugins

Tip: Adding a plugin to a child project (relative to the plugin) automatically symlinks the plugin

Note: Careful with parent plugins and child projects – easy to get circular references in the file system

Finding Plugins

- Cordova Plugin Search: <https://cordova.apache.org/plugins>
- npm: <https://www.npmjs.com/search?q=ecosystem:cordova>
- Or, if the CLI is more your thing:

```
$ npm install -g npms-cli  
$ npms search cordova-plugin device --size=5
```

Package

cordova-plugin-device • <https://github.com/apache/cordova-plugin-device>
Cordova Device Plugin
updated 2 months ago by shazron

Plugin X-ray

Photo by dcondrey (<https://pixabay.com/en/users/dcondrey-122249/>), courtesy of Pixabay.com

The Stuff Plugins are Made of

Ingredients	Ingredients
Metadata	Documentation ^s
Native Code *	JavaScript *
Tests ^s	Hooks *
Typings *	TLC

* Optional

^s Optional but highly suggested

Plugin Structure	Description
<code>cordova-plugin-your-plugin/</code>	Plugin root
<code>package.json</code>	npm metadata
<code>plugin.xml</code>	Plugin metadata and configuration
<code>README.md</code>	English documentation
<code>doc/locale</code>	Documentation other than English <i>Please add tests!</i>
<code>tests/</code>	
<code>types/</code>	TypeScript typings
<code>src/platform</code>	Platform-specific native code
<code>android/</code>	Native Android code
<code>YourPlugin.java</code>	
<code>ios/</code>	Native iOS code
<code>CDV YourPlugin.h</code>	
<code>CDV YourPlugin.m</code>	
<code>www/</code>	JavaScript code & web assets
<code>yourPlugin.js</code>	API for JavaScript consumers

(representational; not every file is included here); Ex: Device Plugin

Metadata

plugin.xml

id, version, author, license, name, description, repo, issue, keywords, platform (& assets), dependencies, engines, preferences, hooks, info, etc.

package.json

name, version, author, license, description, repository, issue, keywords, platforms, dependencies

Note: package.json can be generated by plugman ; see slide 60

Example Metadata (plugin.xml)

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <plugin xmlns="http://apache.org/cordova/ns/plugins/1.0"
3   xmlns:android="http://schemas.android.com/apk/res/android"
4     id="cordova-plugin-device" version="1.1.5-dev">
5   <name>Device</name>
6   <description>Cordova Device Plugin</description>
7   <license>Apache 2.0</license>
8   <keywords>cordova,device</keywords>
9   <repo>https://git-wip-us.apache.org/repos/asf/
10    cordova-plugin-device.git</repo>
11   <issue>https://issues.apache.org/jira/browse/CB/
12    component/12320648</issue>
```

Device Plugin's Metadata

npm Metadata Example (package.json)

```
1 { "name": "cordova-plugin-device",
2   "author": "Apache Software Foundation",
3   "license": "Apache-2.0",
4   "version": "1.1.5-dev",
5   "description": "Cordova Device Plugin",
6   "types": "./types/index.d.ts",
7   "cordova": {
8     "id": "cordova-plugin-device",
9     "platforms": ["android", "ios", "windows", "wp8", ... ] },
10  "repository": { "type": "git", "url": "https://..." },
11  "keywords": [ "cordova", "device", "ecosystem:cordova",
12                "cordova-ios", "cordova-android", ... ],
```

Device Plugin's package.json

JavaScript Modules

Automatically injected into your consumer's index.html . docs

```
<js-module src="www/device.js" name="device">  
  [<children/>]  
</js-module>
```

Children	Description
<clobbers target="device" />	overwrites window.device
<merges target="device" />	merges with window.device
<runs />	runs, but doesn't export

- Unless necessary, target cordova.plugins.yourPlugin

Indicate Platform Support

Use <platform> tags: [docs](#)

```
<platform name="android">  
  ...  
</platform>  
<platform name="ios">  
  ...  
</platform>
```

Note: Visible platform support on plugin repo is separately controlled (package.json keywords)

Specifying headers, frameworks, etc.

```
1 <platform name="android">
2   <source-file src="src/android/Device.java"
3     target-dir="src/org/apache/cordova/device" />
4 </platform>
5 <platform name="ios">
6   <header-file src="src/ios/CDVDevice.h" />
7   <source-file src="src/ios/CDVDevice.m" />
8   <framework src="libz.tbd" />
9 </platform>
```

Other asset tags: `asset` , `resource-file` , `lib-file`; [full docs](#)

Note: You can include third-party libraries; iOS supports Cocoapods, and Android supports AARs with Gradle.

Bug: On iOS hidden (dot) files may not be copied. See [CB-10135](#)

Plugin Class Mapping

- Android (Geolocation)

```
<config-file target="res/xml/config.xml" parent="/*">
  <feature name="Geolocation">
    <param name="android-package"
          value="org.apache.cordova.geolocation.Geolocation" />
  </feature>
</config-file>
```

- iOS (Geolocation)

```
<config-file target="config.xml" parent="/*">
  <feature name="Geolocation">
    <param name="ios-package" value="CDVLocation"/>
  </feature>
</config-file>
```

- Use `<param name="onload" value="true" />` to init at startup

Manifest Modifications

- config-file 1 docs
 - Adds elements to manifests / plist or platform config.xml

```
1 <config-file target="AndroidManifest.xml" parent="/*>
2   <uses-permission android:name=
3     "android.permission.WRITE_EXTERNAL_STORAGE" />
4 </config-file>
5 <config-file target="*-Info.plist"
6   parent="NSLocationWhenInUseUsageDescription">
7   <string>$GEOLOCATION_USAGE_DESCRIPTION</string>
8 </config-file>
```

1: android, file transfer; ios, geolocation; windows, geolocation

Manifest Modifications (2)

- edit-config 1 docs
 - Edits attributes of existing elements in manifests

```
1 <edit-config file="AndroidManifest.xml"
2   target="/manifest/application/activity"      \
3     [android:name='MainActivity']"
4   mode="merge">
5   <activity android:theme="@style/AppTheme" />
6 </edit-config>
```

Dependencies (< cordova@6.1.0)

Before cordova@6.1.0, plugin.xml managed dependencies:

- plugin dependencies¹ docs

```
<dependency id="cordova-plugin-file" version="^4.0.0" />
```

- platform & tool dependencies² docs

```
<engines>
  <engine name="cordova" version=">=3.1.0" />
</engines>
```

Note: don't forget about XML entities! So "<" becomes "<"

Dependencies (cordova@6.1.0+)

Now dependencies should be managed in package.json : docs

```
"engines": {  
  "cordovaDependencies": {  
    "2.0.0": { // plugin version (applies to any ver 2+)  
      "cordova-plugin-console": ">1.0.0",  
      "cordova": ">6.0.0" // cordova-cli above version 6  
    }  
  }  
}
```

Creating and Publishing Plugins

■ And getting rich, maybe? ■

Or maybe not...

plugman

plugman is a node library that manages plugins in your projects.
cordova-cli, phonegap-cli, etc., use plugman internally.

It is also used to create an initial plugin project:

```
$ | npm install -g plugman
$ | plugman create --name YourPlugin
$ |           --plugin_id cordova-plugin-your-plugin
$ |           --plugin_version 0.0.1
$ |           --path .
```

Pass --variable-name=value strings to supply extra config

phonegap-plugin-template

Or, use PhoneGap's plugin template to create a plugin:

<https://github.com/phonegap/phonegap-plugin-template>

```
$ npm i -g https://github.com/phonegap/phonegap-plugin-template  
  
$ phonegap-plugin-create ./your-plugin YourPlugin  
$ cordova-plugin-your-plugin      # parameters: path name plugin-id  
? license[MIT] [enter]
```

Creates `docs`, `src/android`, `src/ios`, `www`, `plugin.xml`,
`package.json`, and `README.md` (as well as some dot files)

Documentation

Documentation is critical; how else will you users know how to use your plugin?

- Location of documentation
 - English goes in `README.md` (plugin root)
 - Other languages in `docs/[locale]/README.md`
- Provide examples, constants, errors that can be thrown, etc.

JavaScript Code

Consumer API
Consumer-facing functions and methods
`cordova.plugins.yourPlugin`

Cordova Internals

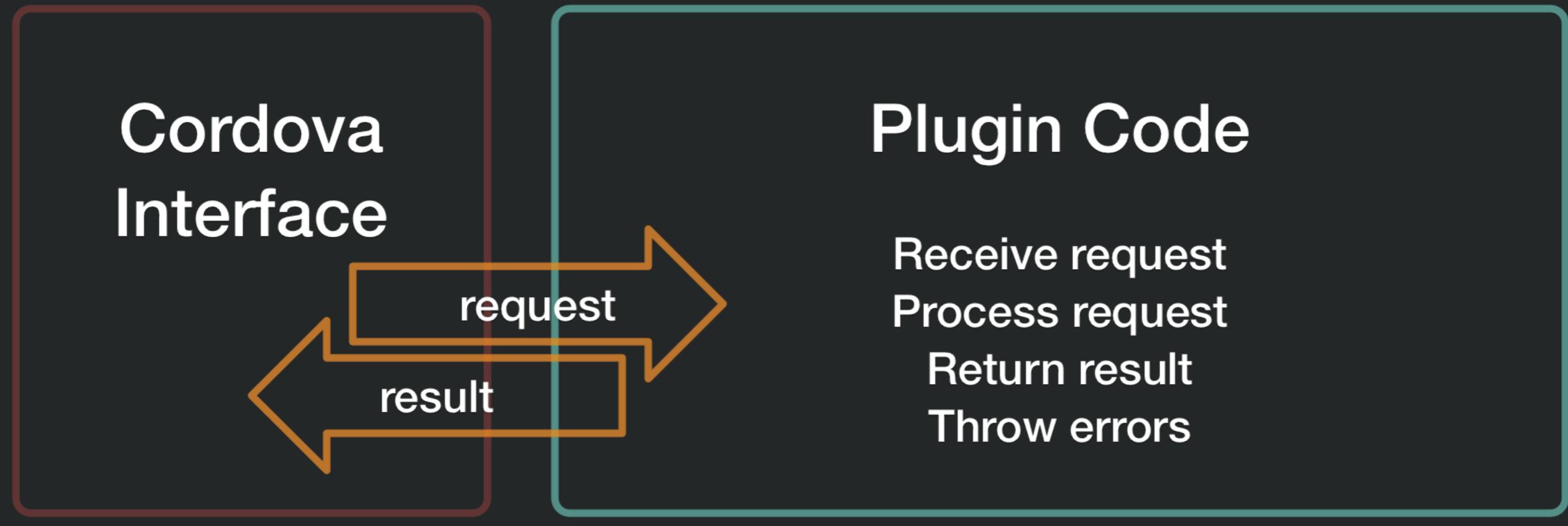
`cordova.exec`
`result`

Your Plugin's JS API

In www/yourPlugin.js (consumer API):

```
module.exports = {
  add: function(successFn, failureFn, arg1, arg2) {
    if (typeof successFn !== "function") {
      throw new Error("Success callback must be a function");
    }
    /* check remaining types */
    cordova.exec(successFn, failureFn, "YourPlugin",
      "add", [arg1, arg2]);
  }
}
```

Native Code



Your Native Code (iOS)

```
#import <Cordova/CDV.h>
@interface CDVYourPlugin : CDVPlugin
@end
@implementation CDVYourPlugin
- (void)add:(CDVInvokedUrlCommand*)command {
    double arg1 = [[command argumentAtIndex:1] doubleValue];
    double arg2 = [[command argumentAtIndex:2] doubleValue];
    CDVPluginResult* result = [CDVPluginResult
        resultWithStatus:CDVCommandStatus_OK messageAsDouble:arg1+arg2];
    [self.commandDelegate sendPluginResult:result
        callbackId:command.callbackId];
}
@end
```

Your Native Code (Android)

```
package com.example.yourplugin;

import org.apache.cordova.CordovaPlugin;
import org.apache.cordova.CallbackContext;
import org.apache.cordova.PluginResult;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public class YourPlugin extends CordovaPlugin {

    @Override
    public boolean execute(String action, JSONArray args,
        CallbackContext callbackContext) throws JSONException {
        if ("add".equals(action)) {
            callbackContext.success(args.getDouble(0) + args.getDouble(1));
        } else { return false; }
        return true;
    }
}
```

Your Native Code (Browser / Win)

```
//src/[browser|windows]/yourPluginProxy.js
function add(successFn, failureFn, args) {
    successFn(args[0] + args[1]);
}

module.exports = {
    add: add
};

require("cordova/exec/proxy").add("YourPlugin",
    module.exports);
```

StatusBar Example

In www/statusbar.js (consumer API)

```
function styleDefault() {
    cordova.exec(null, null, "StatusBar", "styleDefault", []);
}
```

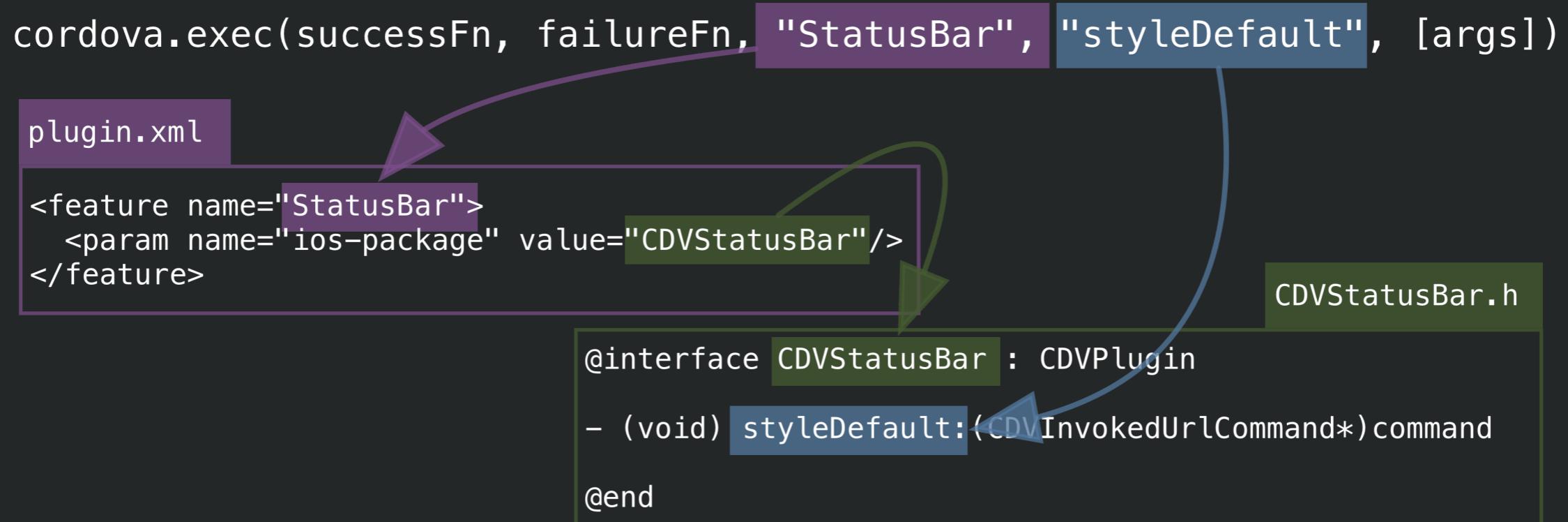
In src/ios/CDVStatusBar.m (native code)

```
- (void) styleDefault:(CDVInvokedUrlCommand*)command {
    [self setStyleForStatusBar:UIStatusBarStyleDefault];
}
```

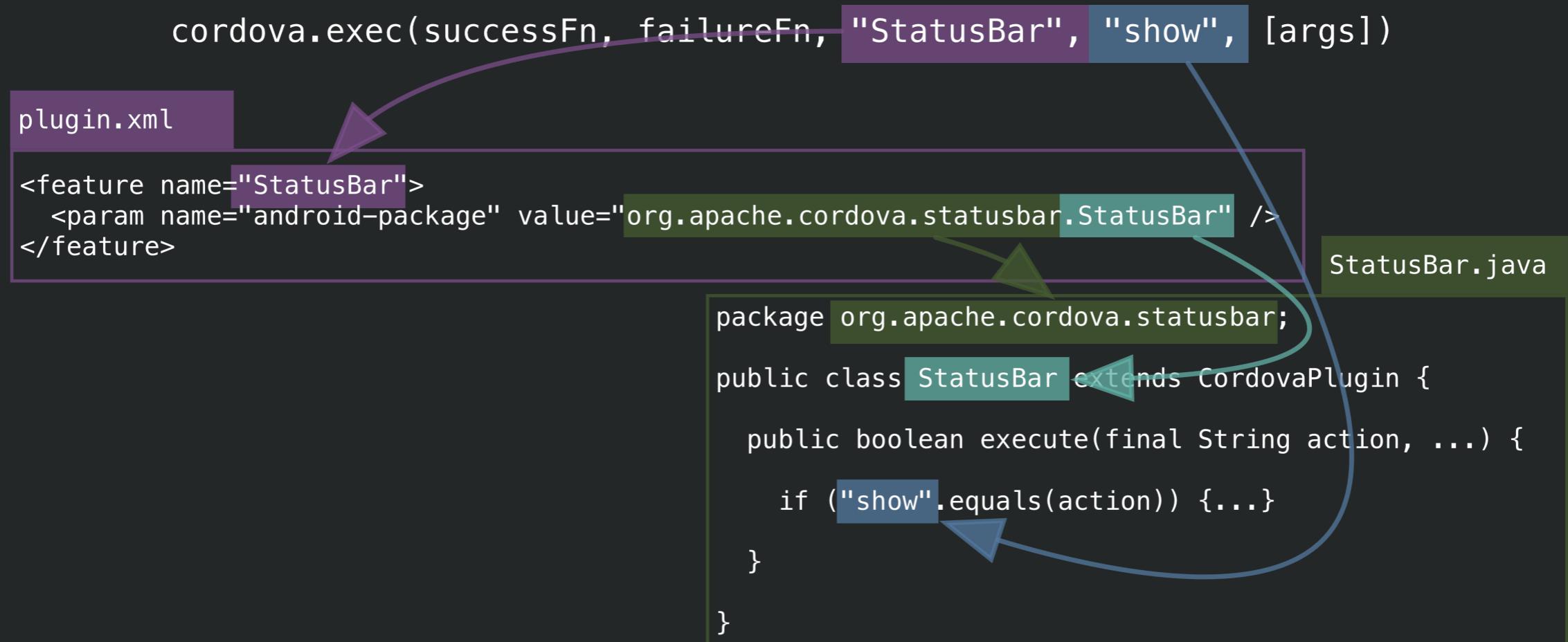
Refs: statusbar.js, plugin.xml, CDVStatusBar.m

StatusBar Plugin Class Mapping (iOS)

Remember the JS API's call to `cordova.exec` ?



StatusBar Plugin Class Mapping (Android)



Triggering callback more than once

```
// in CDVStatusBar.m
- (void) fireTappedEvent {
    if (_eventsCallbackId == nil) { return; }
    NSDictionary* payload = @{@"type": @"tap"};
    CDVPluginResult* result = [CDVPluginResult
        resultWithStatus:CDVCommandStatus_OK
        messageAsDictionary:payload];
    [result setKeepCallbackAsBool:YES]; // default is NO
    [self.commandDelegate sendPluginResult:result
        callbackId:_eventsCallbackId];
}
```

Ref

A photograph of a small, rustic wooden bridge with a curved arch and a flat walkway, situated over a shallow stream or rocky path in a lush green garden. In the background, there's a colorful wooden playhouse and some large rocks. The foreground is filled with green grass and plants.

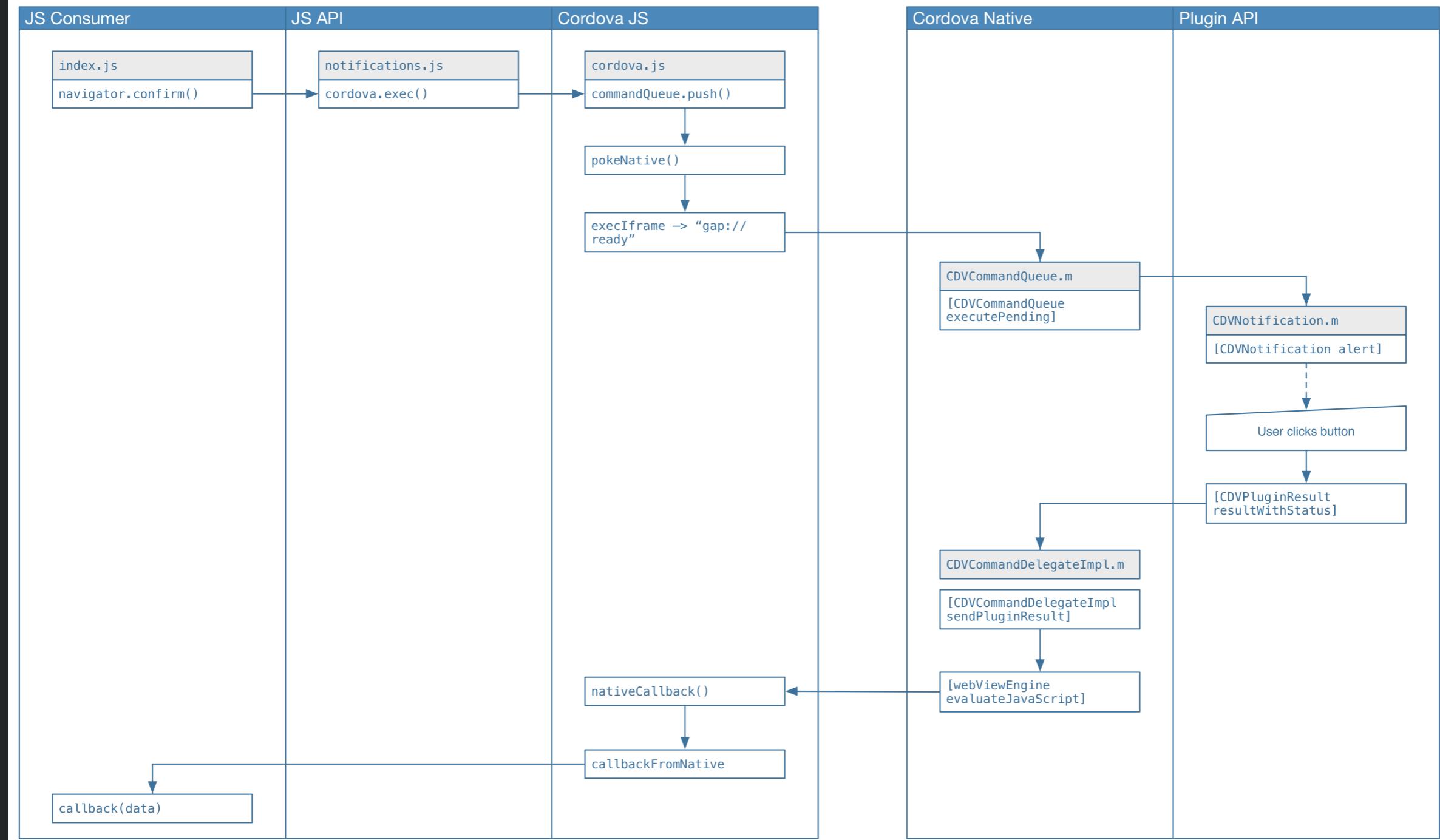
Crossing the bridges

Know your Bridges

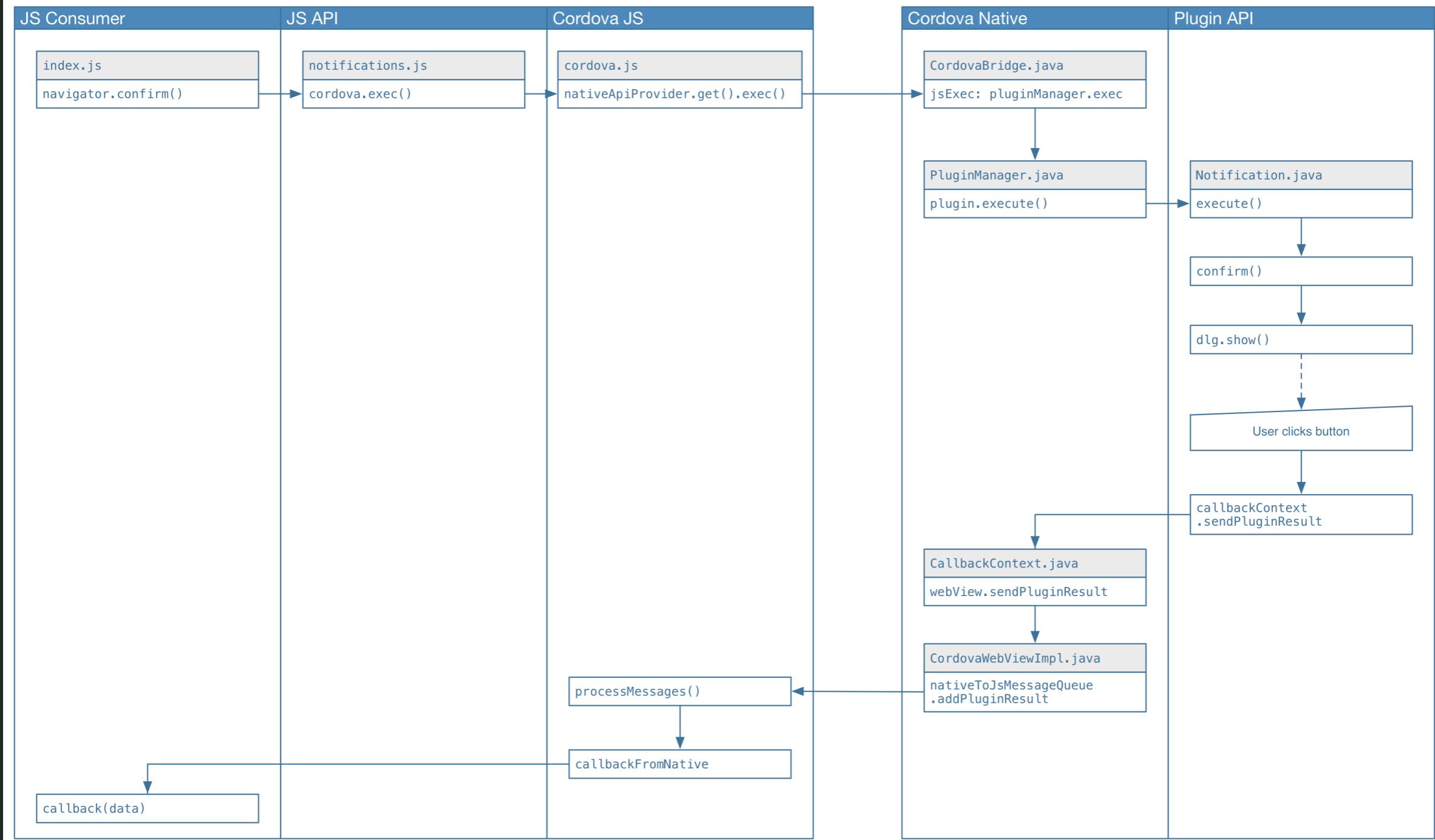
Allows communication between native code and web view contexts.

- iOS
- Android
- Browser/Windows is an exception...
 - Careful, the bridge is a **mirage!** 
 - JavaScript is **native** 
 - `cordova.exec` uses a proxy to keep things consistent
 - **full docs**

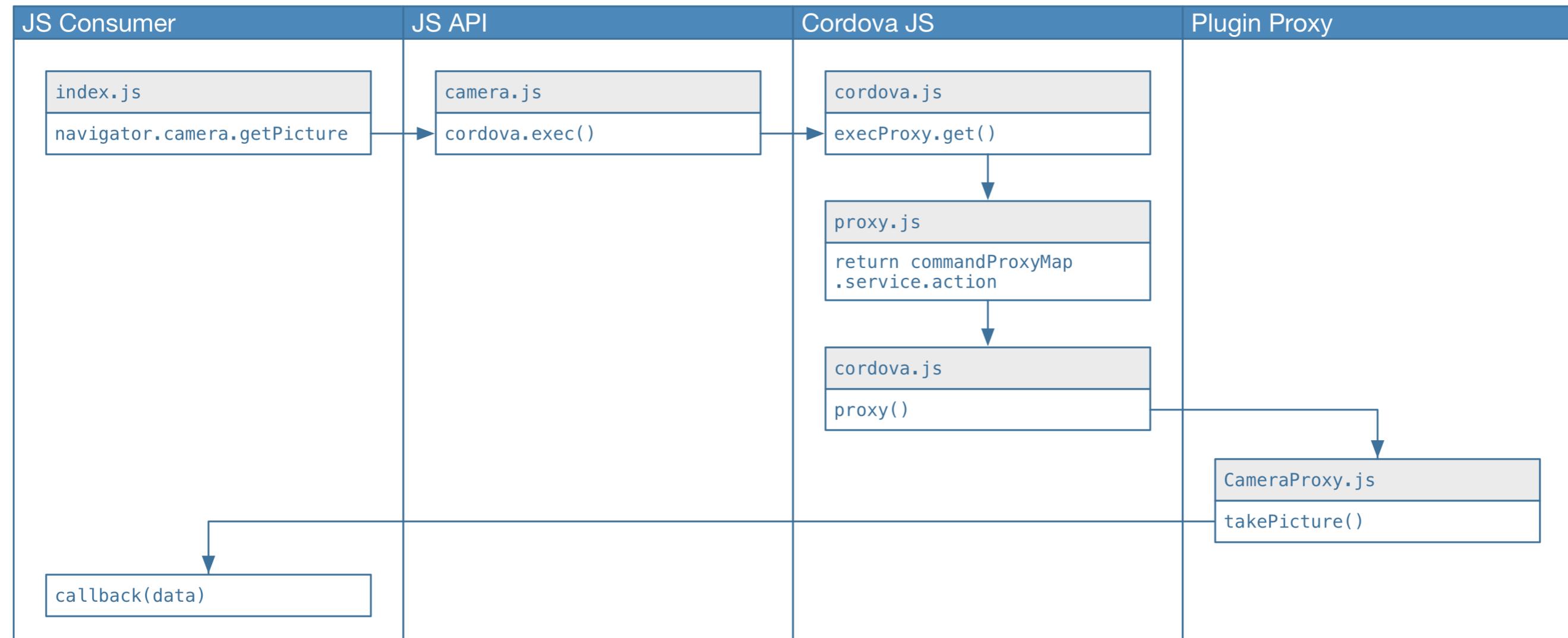
Cordova iOS Bridge (abridged)



Cordova Android Bridge (abridged)



Cordova Windows / Browser “Bridge” (abridged)



The background of the slide features a complex, three-dimensional geometric pattern. It is composed of numerous small, light-colored cubes or rectangular prisms arranged in a way that creates a dense, organic, and somewhat fractal-like texture. The pattern is darker in the center and lighter towards the edges, giving it a sense of depth and volume.

Demo Time

`cordova-plugin-example-isprime`

Tests

**Cordova Test
Harness**

cordova-paramedic
cordova-plugin-test-
framework

Test Cases

Your Jasmine tests
Automatic & Manual

Testing plugins

`cordova-medic` is a test tool designed to run all the core Cordova plugin tests as part of Cordova's continuous integration system

- Tests are written in Jasmine 2.0
- Tests run asynchronously
- Plugins have a dependent test plugin which is installed separately (usually in `/tests` by convention)
- Many of these pieces of `cordova-medic` are reusable, so Jesse spun them into another purpose-based tool...

cordova-paramedic

n. provides advanced levels of care at the point of illness or injury, including out-of-hospital treatment, and diagnostic services

```
$ | npm install -g cordova-paramedic  
$ | cordova-paramedic --platform ios --plugin .
```

Repo & docs: <https://github.com/apache/cordova-paramedic>

Automates Jasmine Tests

- Creates a new project (in temporary location)
- Adds the platform specified (ios , android , windows , etc.)
- Installs the cordova-plugin-test-framework plugin
- Installs the plugin specified (in .) (current working directory)
- Installs the plugin's tests (in ./tests)
- Sets start page to cordova-plugin-test-framework 's test runner
- Creates a local server to listen for results
- Exits with success/fail based on results

Note: Only supports npm-published platforms

How to write tests

- Copy a core plugin's tests – we all do it!
- Create a `tests` folder in your plugin's repository
- Add a `plugin.xml` file (doesn't need to be complex) eg

```
<plugin xmlns="http://apache.org/cordova/ns/plugins/1.0"
  xmlns:android="http://schemas.android.com/apk/res/android"
  id="cordova-plugin-statusbar-tests" version="2.2.3-dev">
  <name>Cordova StatusBar Plugin Tests</name>
  <license>Apache 2.0</license>
  <js-module src="tests.js" name="tests"></js-module>
</plugin>
```

Testing Tips

- Automate as much as you can (`exports.defineAutoTests`)
- For tests that can't be automated, use manual tests (`exports.defineManualTests`)
- Don't forget to accept & call `done` in your `it` tests when working with callbacks and promises.
- If you've got similar tests, you can build them programmatically

Debugging & Iterating

Debugging & Iterating

- Be sure to `--link` your plugin for easier development
- Xcode (macOS) / Safari
 - Concurrently can be a little fussy
- Android Studio / Google Chrome
- Visual Studio (Windows)

What gets --linked?

- app's `plugins/<your-plugin>` is symlinked to your plugin
- Native code in app's `platforms/`

Exceptions & notes:

- `plugin.xml` changes require an `rm` & `add`
- `www` only propagates to `platforms/` at next `prepare`
- `platform rm` & `add` won't preserve `--links` (CB-12597)

Hook

noun A piece of code that hooks into a Cordova process in order to perform some action on behalf of the plugin; see dev guide.

Possibilities:

- Create entitlements as needed
- Transform code (transpile, version # replacement, etc.)
- Create launch images and icons
- Check plugin versions and warn if out-of-date
- **Note:** NOT supported by PhoneGap Build

Publishing your plugin

- If you want to publish to npm , you'll need a package.json
- plugman can generate it based on plugin.xml for you:
`$ | plugman createpackagejson .`
- If you used the PhoneGap Plugin Template, package.json is already there – you'll need to update it.
- Once package.json is correct, publish via:
`$ | npm publish`



Tips & Tricks

JavaScript API

- Promisify your API
- Preprocess arguments in JavaScript
 - convert to appropriate types
 - throw type-mismatch errors, etc.
- Transpile ES2015+ to ES5
- Stick to the `cordova.plugins` namespace
 - Unless creating a polyfill; `window` is crowded!
- Return useful error messages to error callbacks

Native

- Return useful error information
- Use background threads for processing
 - iOS documentation
 - Android documentation
- Avoid init at app startup unless necessary
`<param name="onload" value="false" />`
- Override `onReset` to clean up when web view navigates eg ios
android

Native (Android)

- Override `pluginInitialize` for plugin initialization logic [code](#)
- Runtime Permission Requests (Marshmallow) [docs](#)
 - `cordova.requestPermission()` [code](#)
 - `cordova.hasPermission()` [code](#)
 - Override `onRequestPermissionsResult` [code](#)
- Don't forget Android activity lifecycle [docs](#) [code](#)

Native (iOS)

- Use `pluginInitialize` for plugin initialization logic eg code
- If memory is getting low, `onMemoryWarning` is called code
- If app is going to be terminated, `onAppTerminate` is called code
- You can respond to `pause`, `resume`, etc. code, but you have to register for notifications in `pluginInitialize`
- If you need to handle URLs, override `handleOpenURL` code
- Never, ever call JavaScript that triggers blocking UI (e.g. `alert`)

Miscellaneous

- Don't forget the `browser` platform!
 - Useful when testing on the desktop
 - Mock results if no equivalent browser support
- Be kind when using hooks!
 - **Don't be evil!** Your hook executes on your user's machine!
 - `before_prepare` plugin hooks not run on discovery; run the `cordova` command again
 - `events.emit("verbose", ...)` and `--verbose` are your friends when troubleshooting

Homework

- Create a new plugin and add it to a Cordova project
- Extend and/or improve a plugin
 - For example, the globalization plugin's API is asynchronous, which is really irritating.
 - All the formatting / globalization information could be determined up-front instead
 - Try it: <https://github.com/apache/cordova-plugin-globalization>
- The sky's the limit!

Some more cool plugin ideas

- Game controller support
- Apple Pencil / Stylus support (pressure, tilt)
- Audio/video processing
- Faster computation (compared with JavaScript)

Questions?

Thanks!

Jesse (@purplecabbage) • Kerri (@kerrishotts)

Slides available at: <https://goo.gl/f12nh0>

Based on Jesse's PG Day 2016 EU plugin workshop

This slide intentionally left blank