Fantastic Plugins &
How to Make Them

Kerri Shotts (@kerrishotts)

Jesse MacFadyen (@purplecabbage)

https://github.com/kerrishotts/pgday/2017/fantastic-plugins-and-how-to-make-them

Based in part on http://purplecabbage.github.io/slides/pgd16Plugins/index.html

Photo by AlexanderStein (https://pixabay.com/en/users/AlexanderStein-45237/), courtesy of Pixabay.com

About Kerri

- Used PhoneGap for six+ years
- Author of five books about PhoneGap
- IT Consultant for eight years
- Apache Cordova comitter
- One of many 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 at these times:
  - run time
  - build time
  - install time
- Two categories
  - Core — used to be built in
  - Community — people like you!

Plugins at run time

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

- Faster computations (compared to JS)
- Expose native device features
  - push notifications, native social network sharing
- Use native widgets
  - Microsoft ACE
- Quality assurance, logging, etc.
- Analytics

Plugins at build time

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

- Transpile ES2015+, TypeScript, etc. to ES5
- Bundle dependencies (webpack, browserify, jspm)
- Pre-process CSS files (SASS, less, auto-prefixer)
- Check code quality (eslint, tslint, jshint)
- Run tests, create code coverage reports

Plugins at install time

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

- Could bundle other plugins
- Could configure the project environment
- Or, could provide tests for another plugin…

Plugin-ception 🎇

The Core Plugins

Core Cordova features (used to be built-in)

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

Extensions provided by the community — like you!

Repository	Plugins
https://cordova.apache.org/plugins	~1,960 plugins (– core)
http://www.plugreg.com	~1,592 plugins (– core)
http://plugins.telerik.com/cordova	~77 plugins

Managing Plugins

or, finding fantastic plugins…

npm

Plugins are typically downloaded from npm:

```
[user@dev] $ | cordova plugin add --save cordova-plugin-device
             |
[user@dev] $ | cordova plugin ls                        # or list
             | cordova-plugin-device 1.1.1 "Device"
             |
[user@dev] $ | 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)

Github

Plugins can also be installed from a Github repository.

```
[user@dev] $ | cordova plugin add --save \
             |   http://github.com/apache/cordova-plugin-device
[user@dev] $ | cordova plugin rm --save cordova-plugin-device
```

Can specify a branch, too (useful for testing pre-release plugins):

```
[user@dev] $ | cordova plugin add --save \
             |   http://github.com/apache/cordova-plugin-device#branch
```

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

Local Filesystem

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

- Use `--link` when developing plugins
  - Changes are reflected automatically — no `rm` & `add` flow
  - Automatically symlinked if a parent (`../`)

Note: Careful with parent plugins and child projects — easy to get circular references in the file system (borks **cp**)

Finding Plugins

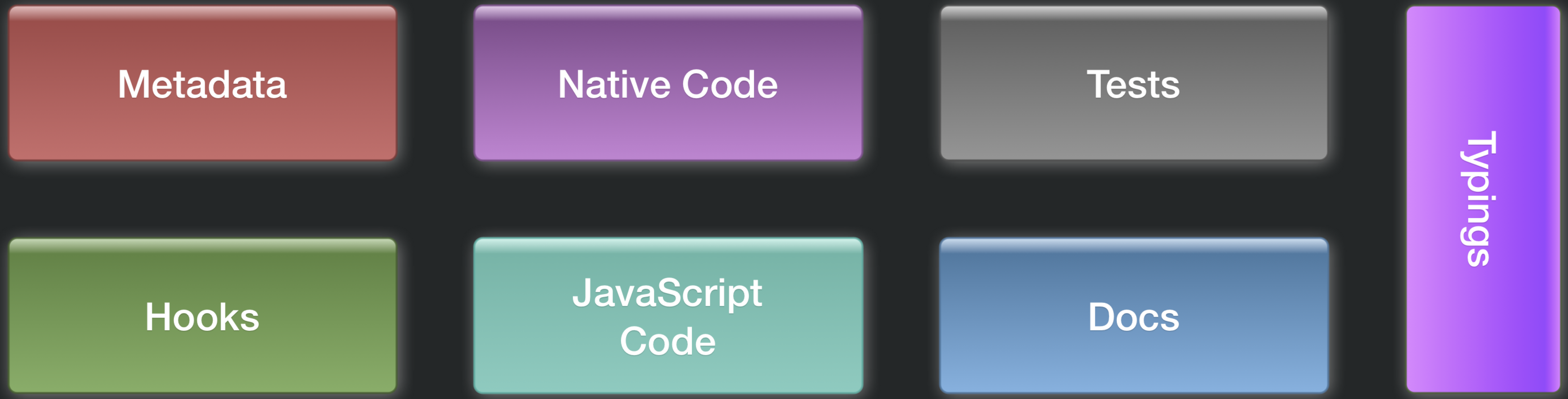
- Cordova Plugin Search: https://cordova.apache.org/plugins
- npm: https://www.npmjs.com/search?q=ecosystem:cordova
- Or, if the CLI is more your thing:

```
[user@dev] $ | npm install -g npms-cli
[user@dev] $ | npms search cordova-plugin device --size=5
             |  ┌───────────────────────────────────────────────────────────
             |  │ Package
             |  ├───────────────────────────────────────────────────────────
             |  │ cordova-plugin-device • https://github.com/apache/cordc
             |  │ Cordova Device Plugin
             |  │ updated 2 months ago by shazron
             |  ├───────────────────────────────────────────────────────────
```

Plugin Autopsy ⚰️

or, what's inside these things?

ref: cordova-plugin-device


Metadata
Native Code
Tests
Typings
Hooks
JavaScript Code
Docs

Plugin Structure

```
cordova-plugin-device/         # plugin root
    doc/<locale>               # documentation other than English (convention)
    src/<platform>             # Platform-specific native code
    |   android/
    |   +   Device.java        # Native Android code
    |   ios/
    |   |   CDVDevice.h        # Native iOS header
    |   +   CDVDevice.m        # Native iOS code
    |
    tests/                     # Please add tests!
    types/                     # Types for Typescript
    www/                       # Web assets
    +   device.js              # API for JavaScript consumers
    package.json               # npm metadata
    plugin.xml                 # plugin metadata and configuration
    README.md                  # English documentation
```

(representational only; not every file is included here)

Metadata

$$\text{plugin.xml} \xrightarrow{\text{plugman}} \text{package.json}$$

plugin.xml

ID, Author, Title, Author, Description, Keywords, Version #, Platforms, Dependencies, Permissions

package.json

Name, Author, Description, Repo, License, Platforms, Keywords, Dependencies

Example Metadata (plugin.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin xmlns="http://apache.org/cordova/ns/plugins/1.0"
    xmlns:rim="http://www.blackberry.com/ns/widgets"
    xmlns:android="http://schemas.android.com/apk/res/android"
    id="cordova-plugin-device"
    version="1.1.5-dev">
    <name>Device</name>
    <description>Cordova Device Plugin</description>
    <license>Apache 2.0</license>
    <keywords>cordova,device</keywords>
    <repo>https://link/to/git/repository.git</repo>
    <issue>https://link/to/issue/reporter.html</issue>
```

JavaScript API Entry

In `cordova-plugin-device`'s `plugin.xml`:

```
<js-module src="www/device.js" name="device">
    <clobbers target="device" />
</js-module>
```

Examples: Multiple `clobbers`[1], `runs`[2], `merges`[3]

1. clobbers, in app browser
2. runs, file transfer
3. merges, vibration

Indicate Platform Support

Using `<platform>` tags:

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

Specifying headers, frameworks, etc.

```
<platform name="android">
  <source-file src="src/android/Device.java"
               target-dir="src/org/apache/cordova/
</platform>
<platform name="ios">
    <header-file src="src/ios/CDVDevice.h" />
    <source-file src="src/ios/CDVDevice.m" />
    <framework src="libz.tbd" />
</platform>
```

Note: Can include third-party libraries too. iOS supports Cocoapods too!

Manifest modifications

- `config-file`[1]
  - Adds elements to manifest
- `edit-config`[2]
  - Edits attributes of existing elements

1. android, file transfer; ios, geolocation; windows UAP, geolocation
2. TODO

npm Metadata Example

```
{
  "name": "cordova-plugin-device",
  "author": "Apache Software Foundation",
  "license": "Apache-2.0",
  "version": "1.1.5-dev",
  "description": "Cordova Device Plugin",
  "types": "./types/index.d.ts",
  "cordova": {
    "id": "cordova-plugin-device",
    "platforms": ["android", "ios", "windows", "wp8", ... ]
  },
  "repository": { "type": "git". "url": "https://..." },
  "keywords": ["cordova", "device",
               "ecosystem:cordova", "cordova-ios", "cordova-android",
               ... ],
```

Dependencies

```
<!-- plugin.xml -->
<dependency id="cordova-plugin-device" />
<dependency id="cordova-plugin-console" version="^1.0.0" />
// or in package.json
"engines": {
    "cordovaDependencies": {
        "2.0.0": { //plugin version (applies to any ver 2+)
            "cordova-plugin-console": "> 1.0.0",
        "cordova": "> 1.0.0" // cordova-cli above version 1
}   }   }
```

Note: don't forget about XML entities! So `< === lt;`
Ex 1: engine, in app browser
Ex 2: dependency, file transfer

Creating and Publishing Plugins

or, the art of crafting 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 can also *create* plugins:

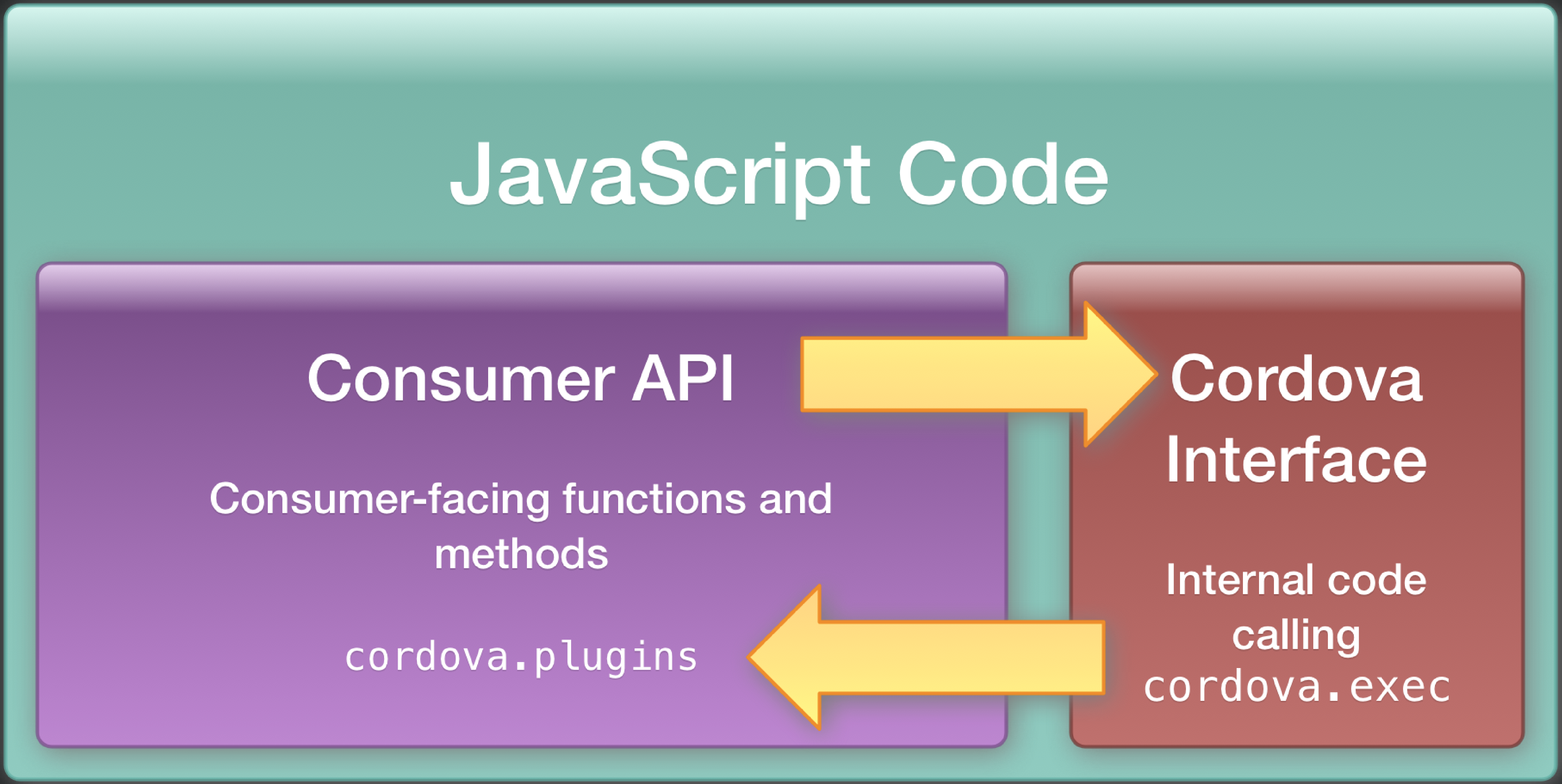
```
[user@dev] $ | npm install -g plugman
[user@dev] $ | plugman create --name Abracadabra \
             |               --plugin_id cordova-plugin-abracadabra \
             |               --plugin_version 0.0.1 \
             |               --path .
```

- Can pass `--variable-name=value` pair string to define additional data like author, etc.

phonegap-plugin-template

Or, use PhoneGap's plugin template:
https://github.com/phonegap/phonegap-plugin-template

```
[user@dev] $ │ npm install -g \
             │   https://github.com/phonegap/phonegap-plugin-template
             │
             │ # phonegap-plugin-create path name plugin-id
[user@dev] $ │ phonegap-plugin-create ./abracadabra Abracadabra \
             │   cordova-plugin-abracadabra
```


JavaScript Code
Consumer API
Consumer-facing functions and methods
cordova.plugins
Cordova Interface
Internal code calling cordova.exec

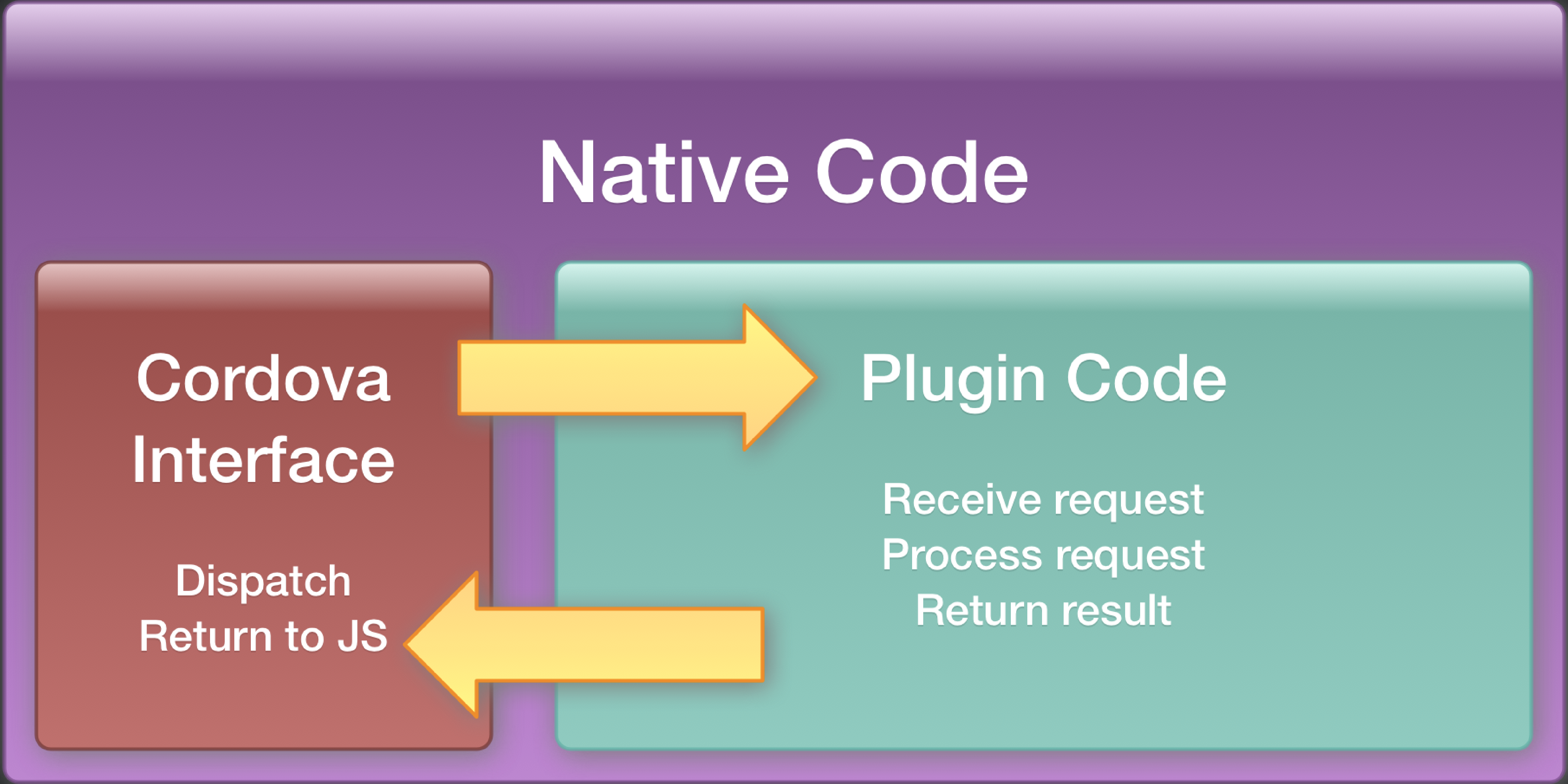
Wiring it all up…

- `www/<plugin>.js` (consumer API)

```
cordova.exec(successFn, failureFn, "PluginName",
             "pluginMethod", args<Array>);
```

- `plugin.xml` : (class mapping)

```
<feature name="PluginName">
    <param name="ios-package" value="CDV<PluginClass>"
    <param name="onload" value="true" />
</feature>
```


Native Code
Cordova Interface
Dispatch
Return to JS
Plugin Code
Receive request
Process request
Return result

Wiring it all up… (2)

📄 `src/ios/CDV<PluginClass>.m` (native code)

```
- (void) <pluginMethod>:(CDVInvokedUrlCommand*)command {
    // do something useful and optionally
    // return results across the "bridge"
}
```

StatusBar Example

www/statusbar.js (consumer API)

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

plugin.xml

```
<feature name="StatusBar">
    <param name="ios-package" value="CDVStatusBar" />
    <param name="onload" value="true" />
</feature>
```

StatusBar Example (2)

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

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

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

```
cordova.exec(null, null, "StatusBar", "styleDefault", []);
"StatusBar"     --> <feature name="StatusBar"> (plugin.xml)
                --> <param ... value="CDVStatusBar"/>
                --> src/ios/CDVStatusBar.m
"styleDefault"  --> -styleDefault:command (CDVStatusBar.m)
```

Returning data back to JavaScript

```
// 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];
}
```

Follow the yellow brick bridge?

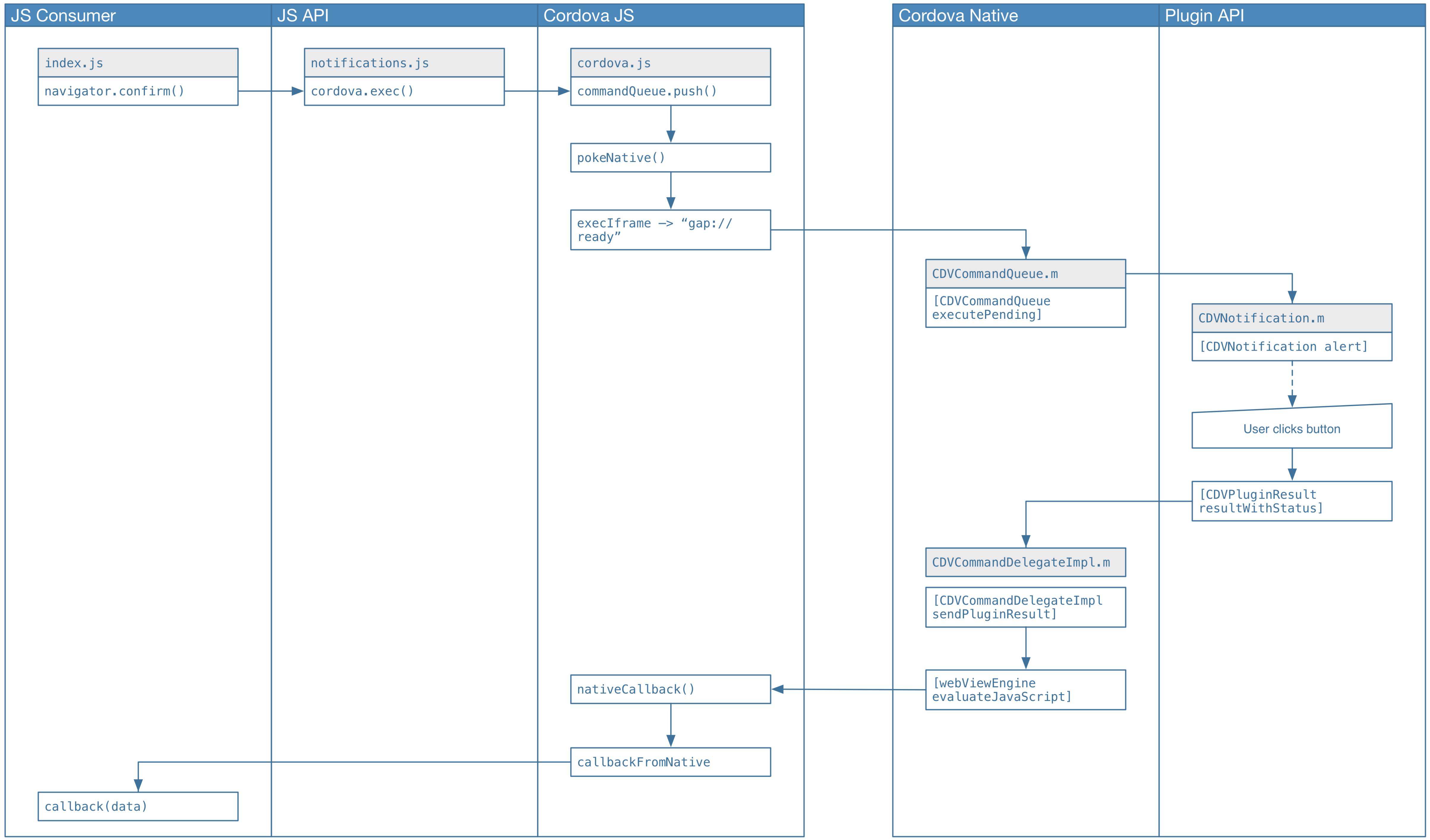
or, a look at the code behind the curtain!

Lots of bridges

A bridge is used to cross the gap between the native code context and the web view context.

- iOS
- Android
- Windows is an exception…
    - Careful, the bridge is a **mirage**! 🏜️
    - JavaScript is **native** 🎇
    - `cordova.exec` uses a proxy

Cordova iOS Bridge (abridged)



Publishing your plugin

- If you want to publish to `npm`, you'll need a `package.json`
- `plugman` can do that for you too!

```
[user@dev] $ plugman createpackagejson .
[user@dev] $ npm publish
```

- Don't panic if the repo doesn't immediately show your plugin
    - wait a while — the underlying index has to catch up

A cool plugin demo

Testing your plugins

or, the art of making sure it works like it should

and improving the lives of developers who use your plugin 😄

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 asynchonously
- 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

```
[user@dev] $ | npm install -g cordova-paramedic

[user@dev] $ | 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)

Debugging

or, mastering the dark art of reading your computer’s mind

Debugging

- Xcode (macOS) / Safari
    - But not concurrently!
- Android Studio / Google Chrome
- Visual Studio (Windows)

Documentation

README.md

English in plugin root (convention)

docs/<locale>/README.md

Other languages in docs/<locale>

Hooks

Before Prepare

Before Compile

After Plugin Install

etc.

Hooks

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

Want to see something cool?

Some more cool plugin ideas

- Optical Character Recognition using Tesseract
- Game controller support
- Apple Pencil, anyone?
- iOS Storage providers
- Audio/video processing

Tips & Tricks

or, wisdom from those who have gone before

and face-palmed for you in your stead…

JS API

- Promisify your API

```
function _promisifyMeMaybe(fn, thisArg) {
  if (typeof Promise === "undefined") { return fn.bind(thisArg); }
  return function _wrapper() {
    return new Promise(function (resolve, reject) {
      fn.apply(thisArg ? thisArg : this,
        [resolve, reject].concat([].slice.call(arguments, 2)));
    }
  }
}
function doSomething(successCB, errorCB, options) {
    return (_promisifyMeMaybe(cordova.exec, cordova)
        (successCB, errorCB, "Abracadabra", "doSomething",
        [arguments.length <= 1 ? successCB : options]));
}
```

JS API (2)

- Preprocess arguments in JavaScript
  - convert to appropriate types
  - throw type-mistmatch errors, etc.
- Transpile ES2015+ to ES5
  - not all targets understand native ES2015 yet
- Oh, and unless you're creating a polyfill, try sticking to the `cordova.plugins` namespace. `window` gets awfully crowded!

Native

- Return useful error information
- Use background threads!
- Be respectful of other plugins
- Lazy load?
- Init events?

Miscellany

- Don't forget the `browser` platform!
    - Useful when testing on the desktop
        - May need to mock results if no equivalent browser support
- Be kind when using hooks!
    - Your hook runs on your consumer's machine!
    - Don't be evil!
    - `before_prepare` hooks may not always be run when you expect; run the `cordova` command again

Miscellany (2)

- `events.emit("verbose", ...)` and `--verbose` are your friends when troubleshooting hooks
- Likewise, return useful error messages to error callbacks

Homework

- Create a new plugin and publish it to the Cordova plugin repo
- 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
        - Go for it: https://github.com/apache/cordova-plugin-globalization
- The sky's the limit!

Questions?

Thanks!

Jesse (@purplecabbage)

Kerri (@kerrishotts)

https://github.com/kerrishotts/pgday/2017/fantastic-plugins-and-how-to-make-them

Based in part on http://purplecabbage.github.io/slides/pgd16Plugins/index.html

This slide intentionally left blank