Creating a Modern PhoneGap Plugin

Presented by

- Kerri Shotts (@kerrishotts)
- Jesse MacFadyen (@purplecabbage)

Slides

- PDF: https://goo.gl/fl2nh0
- Based on PGDay EU 2016 plugin workshop by Jesse

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 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

Photo by skeeze (https://pixabay.com/en/users/skeeze-272447/), courtesy of Pixabay.com

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.

Photo by Pexels (https://pixabay.com/en/users/Pexels-2286921/), courtesy of Pixabay.com

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 🎆

Photo by PublicDomainPictures (https://pixabay.com/en/users/PublicDomainPictures-14/), courtesy of Pixabay.com

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

Devoloped 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 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-
│ 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
cordova-plugin-*your-plugin*/	Plugin root
package.json	npm metadata
plugin.xml	Plugin metadata and configuration
README.md	English documentation
doc/*locale*	Documentation other than English
tests/	*Please add tests!*
types/	TypeScript typings
src/*platform*	Platform-specific native code
android/	Native Android code
YourPlugin.java	
ios/	Native iOS code
CDV*YourPlugin*.h	
CDV*YourPlugin*.m	
www/	JavaScript code & web assets
yourPlugin.js	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 57

Example Metadata (plugin.xml)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<plugin xmlns="http://apache.org/cordova/ns/plugins/1.0"
  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://git-wip-us.apache.org/repos/asf/
    cordova-plugin-device.git</repo>
  <issue>https://issues.apache.org/jira/browse/CB/
    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`

```xml
<config-file target="AndroidManifest.xml" parent="/*">
  <uses-permission android:name=
    "android.permission.WRITE_EXTERNAL_STORAGE" />
</config-file>
<config-file target="*-Info.plist"
  parent="NSLocationWhenInUseUsageDescription">
  <string>$GEOLOCATION_USAGE_DESCRIPTION</string>
</config-file>
```

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

Manifest Modifications (2)

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

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

Dependencies (< cordova@6.1.0)

Before `cordova@6.1.0`, `plugin.xml` managed dependencies:

- plugin dependencies[1] docs

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

- platform & tool dependencies[2] 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
```

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");
    }
    cordova.exec(successFn, failureFn, "YourPluginName",
                 "add", [arg1, arg2]);
  }
}
```

Native Code

Cordova Interface

Plugin Code

Receive request
Process request
Return result
Throw errors

Your Native Code (iOS)

```
// src/ios/CDVYourPlugin.h
#import <Cordova/CDVPlugin.h>
@interface YourPlugin : CDVPlugin
- (void) add:(CDVInvokedUrlCommand*)command;
@end
// src/ios/CDVYourPlugin.m
#import <YourPlugin.h>
#import <Cordova/CDVPlugin.h>
@implementation YourPlugin
- (void) add:(CDVInvokedUrlCommand*)command {
  CDVPluginResult* result = nil;
  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.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)) {
      this.add(args.getDouble(0), args.getDouble(1), callbackContext);
      return true;
    }
    return false;
  }
  private void add(double a, double b, CallbackContext callbackContext) {
    callbackContext.success(a + b);
  }
}
```

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`?

```
cordova.exec(successFn, failureFn, "StatusBar", "styleDefault", [args])
```

plugin.xml

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

CDVStatusBar.h

```
@interface CDVStatusBar : CDVPlugin

- (void) styleDefault:(CDVInvokedUrlCommand*)command

@end
```

StatusBar Plugin Class Mapping (Android)

```
cordova.exec(successFn, failureFn, "StatusBar", "show", [args])
```

plugin.xml

```
<feature name="StatusBar">
  <param name="android-package" value="org.apache.cordova.statusbar.StatusBar" />
</feature>
```

StatusBar.java

```
package org.apache.cordova.statusbar;

public class StatusBar extends CordovaPlugin {

  public boolean execute(final String action, ...) {

    if ("show".equals(action)) {...}

  }

}
```

Returning data 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

Crossing the bridges

Photo by kaboompics (https://pixabay.com/en/users/kaboompics-1013994/), courtesy of Pixabay.com

Know your Bridges

Allows communication between native code and web view contexts.

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

Cordova iOS Bridge (abridged)

A cool plugin demo

Photo by PeteLinforth (https://pixabay.com/en/users/PeteLinforth-202249/), courtesy of Pixabay.com

Testing your plugins

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

```
$ 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>
```

Debugging & Iterating

Photo by ROverhate (https://pixabay.com/en/users/ROverhate-1759589/), courtesy of Pixabay.com

Debugging & Iterating

- Be sure to `--link` your plugin for easier development
- Xcode (macOS) / Safari
  - But not concurrently!
- Android Studio / Google Chrome
- Visual Studio (Windows)

Photo by ROverhate (https://pixabay.com/en/users/ROverhate-1759589/), courtesy of Pixabay.com

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 `--link`s (CB-12597)

Photo by bernswaelz (https://pixabay.com/en/users/bernswaelz-1728198/), courtesy of Pixabay.com

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

Photo by Tama66 (https://pixabay.com/en/users/Tama66-1032521/), courtesy of Pixabay.com

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 .
$ npm publish
```

- Don't panic if the repo doesn't immediately show your plugin
  - wait a while — the underlying index has to catch up
  - (TODO: check; this is getting fixed)

Photo by Tama66 (https://pixabay.com/en/users/Tama66-1032521/), courtesy of Pixabay.com

Tips & Tricks

Photo by Pexels (https://pixabay.com/en/users/Pexels-2286921/), courtesy of Pixabay.com

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

Photo by Pexels (https://pixabay.com/en/users/Pexels-2286921/), courtesy of Pixabay.com

Native

- Return useful error information
- Use background threads for processing
- Avoid init at app startup unless necessary

```
<param name="onload" value="false" />
```

- Use `onReset` method to clean up when web view navigates[1]
- Android
  - use `initialize` for plugin initialization logic[2]
  - use `onResume`, `onDestroy`, etc. to handle life-cycle
- iOS
  - `pluginInitialize` for plugin initialization logic[3]

Photo by Pexels (https://pixabay.com/en/users/Pexels-2286921/), courtesy of Pixabay.com

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

Photo by Pexels (https://pixabay.com/en/users/Pexels-2286921/), courtesy of Pixabay.com

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!

Photo by geralt (https://pixabay.com/en/users/geralt-9301/), courtesy of Pixabay.com

Some more cool plugin ideas

- Game controller support
- Apple Pencil / Stylus support (pressure, tilt)
- Cloud storage support (Dropbox, Google Drive, etc.)
- Audio/video processing
- Faster computation than JavaScript

Photo by Pexels (https://pixabay.com/en/users/Pexels-2286921/), courtesy of Pixabay.com

Questions?

Thanks!

Jesse (@purplecabbage) • Kerri (@kerrishotts)

Slides available at: https://goo.gl/fl2nh0

Based in part on Jesse's PG Day 2016 EU plugin workshop

This slide intentionally left blank