UI TESTING FOR ANDROID

espresso

Chris Davis

Independent Developer

Android and ChromeOS

github.com/rpgobjects
plus.google.com/+ChrisDavis0
linkedin.com/in/rpgobjects

rpgobjects.com
(nerds level 10 only!)

demo app:
github.com/rpgobjects/DevFestEspresso

...and I’m a recovering non-tester

Espresso Background

Android Ui Testing Library developed internally at Google

Open sourced October 2013

Espresso 2.0 released December 2015

Added to Support Library and Android Open Source Project (AOSP)

Documentation moved to d.android.com

Similar Libraries and Frameworks

Robotium

Selendroid

Appium

Calabash

Why  ?

So you don’t have to deal with:

```
// set up an ActivityMonitor
getInstrumentation().addMonitor(aMoniter);

// run something on the main thread
getInstrumentation().runOnMainSync(new Runnable() {
    @Override
    public void run() {
        ...
    }
});

// wait for the main thread to become idle
getInstrumentation().waitForIdleSync();

// yuck!
Thread.sleep(10000);
```

Espresso abstracts the instrumatation Api

Smaller simpler API

More reliable tests

Faster tests

Synchronization of Ui Events

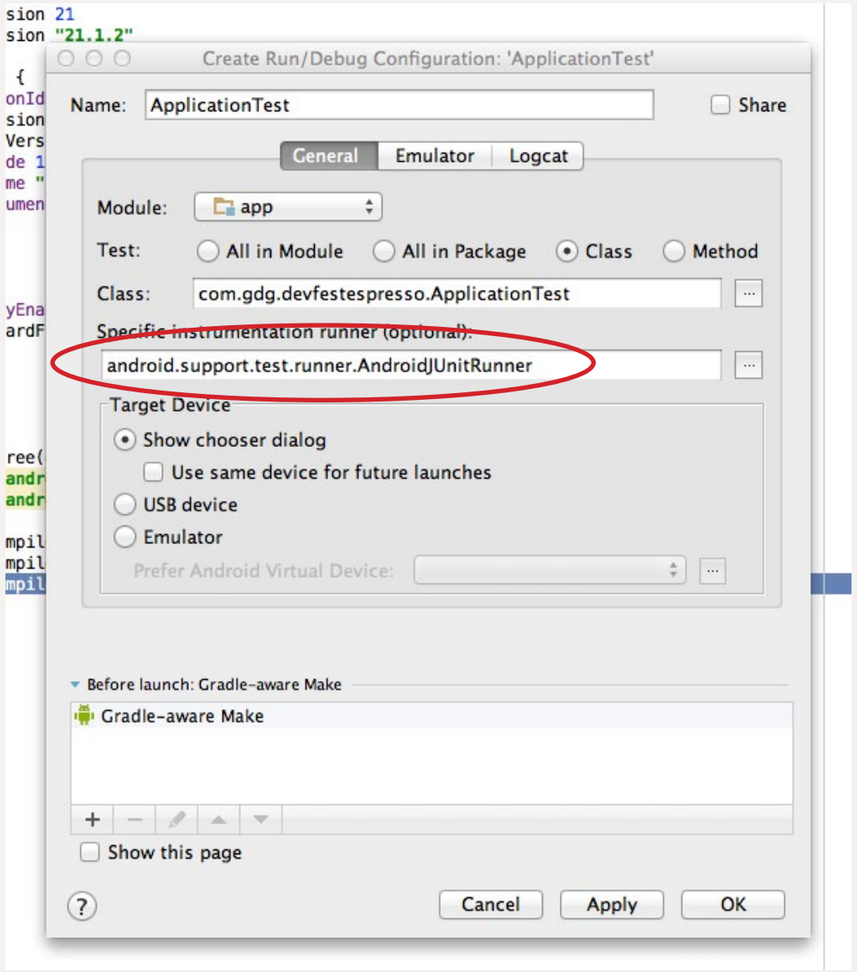
Very extendable

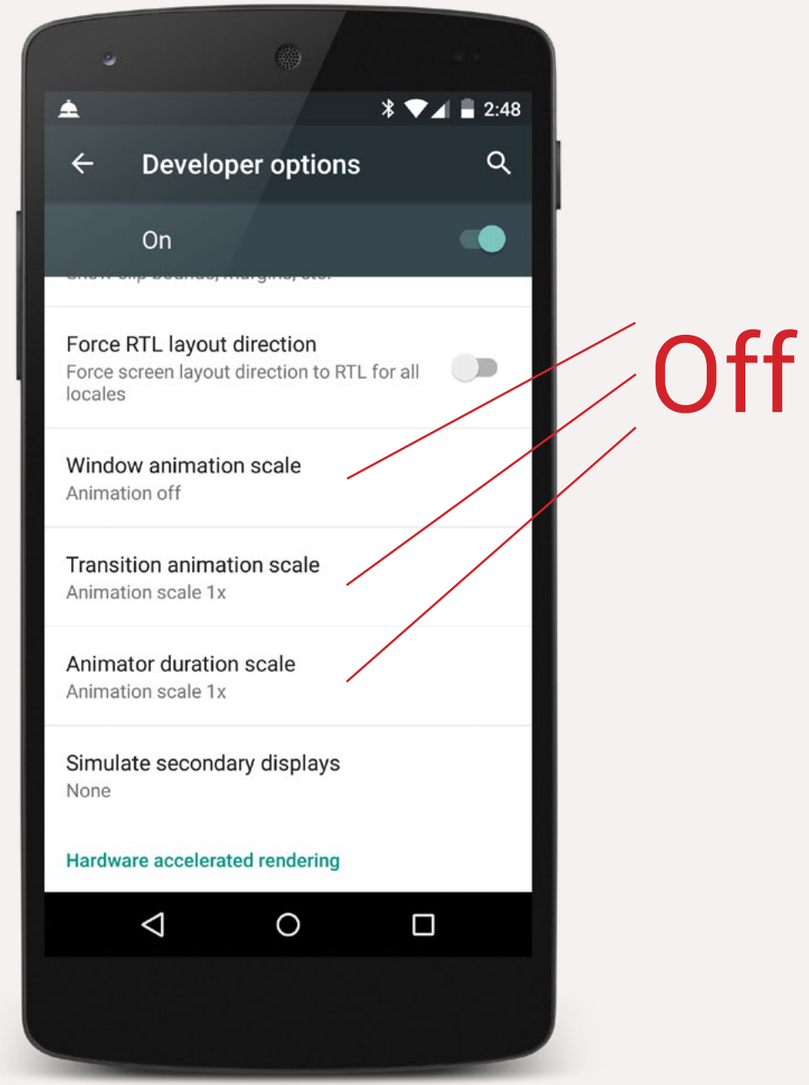
Setup

Gradle setup:

```
android {
    ...
    defaultConfig {
        targetSdkVersion 21
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }

    packagingOptions {
        exclude ‘LICENSE.txt’
    }
    ...
}
dependencies {
    // App's dependencies, including test
    compile 'com.android.support:support-annotations:21.+'

    // Testing-only dependencies
    androidTestCompile 'com.android.support.test:testing-support-lib:0.1'
    androidTestCompile 'com.android.support.test.espresso:espresso-core:2.+'
    // optional
    androidTestCompile 'com.android.support.test.espresso:espresso-contrib:+'
}
```

Run Configuration





Example Test:

```
class ActivityTest extends ActivityInstrumentationTestCase2<MainActivity>
{

    public MainActivityTest() {
        super(MainActivity.class);
    }

    @Override
    public void setUp() throws Exception {
        super.setUp();
        getActivity();
    }

    public void testHeader() {
        onView(withText(R.string.dev_fest)).check(matches(isDisplayed()));
    }
}
```

The Espresso API

```
// finding view
onView(withId(R.id.dev_fest))

// performing action
.perform(click())

// asserting state
.check(matches(isDisplayed()))
```

Espresso Entry Point

View Matchers

View Actions

View Assertions

View Matchers

Based on Hamcrest matchers (hamcrest.org)

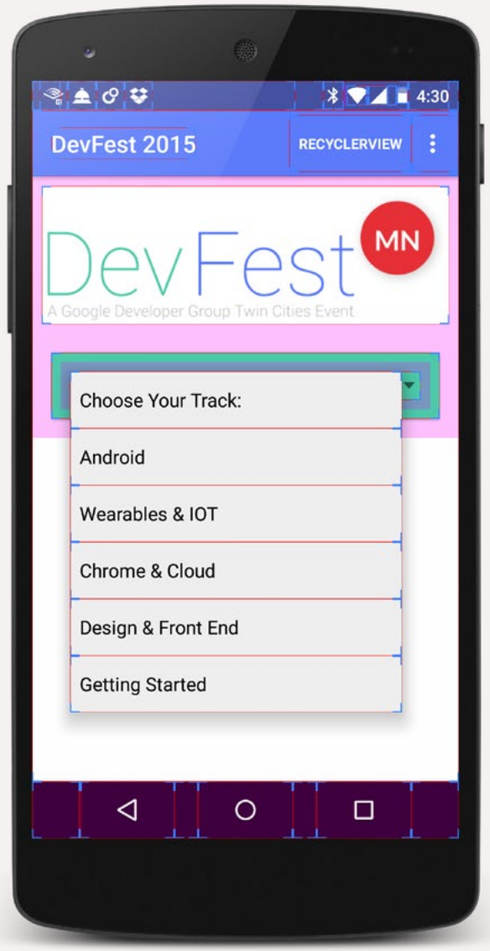
Filters views for finding single view (**onView**)

Filters views for assertions (**check**)

ViewMatchers class implements Android specific Matchers

Finding Your View

```
// Most of time you'll simply use an id
onView(withId(R.id.dev_fest))

// No id? Or maybe the id isn't unique
onView(withText(R.string.dev_fest))
onView(withHint(R.string.dev_fest))

// Better yet, be clever and accessible
onView(withContentDescription("Dev Fest Image"))

// Be tricky with combinations
onView(allOf(withId(R.id.dev_fest), withText(R.string.dev_fest)))
onView(allOf(withId(R.id.dev_fest), not(withText(R.string.dev_fest))))
// allOf and not via hamcrest
```



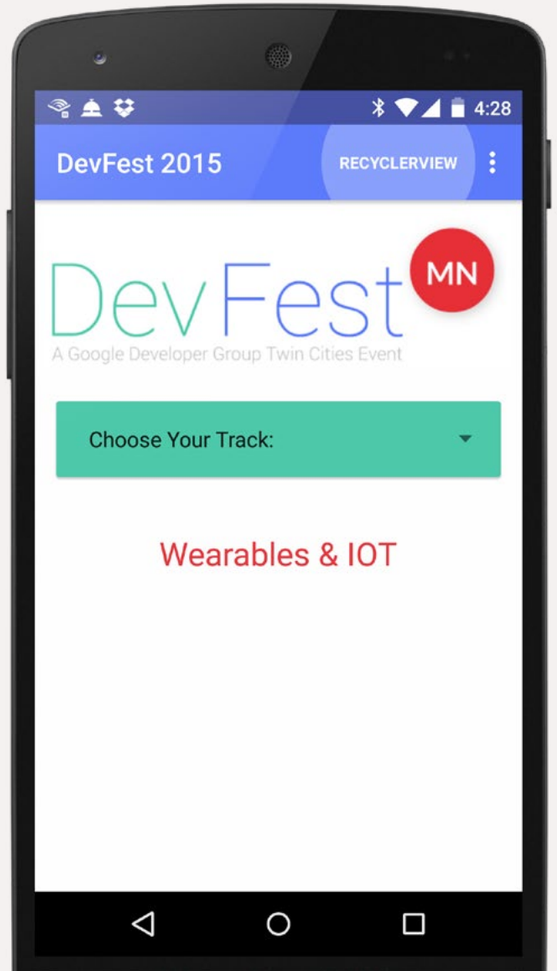
View Actions

```
onView(...).perform(final ViewAction... viewActions)
```

Performs natural actions on views

Runs on the Ui Thread

ViewActions class implements most common view actions



Performing Actions

```
// Click a view
onView(withId(R.id.dev_fest)).perform(click());

// You can chain them ...
onView(...).perform(click()).peform(swipeUp());
// ... but you also execute multiple actions with one call
onView(...).perform(click(),typeText(“android“),closeSoftKeyboard());

// inside ScrollView? preced actions with scrollTo
onView(...).perform(scrollTo(), click());
// scrollTo has no effect if view is already visible
```

View Assertions

```
onView(...).check(ViewAssertion viewAssert)
```

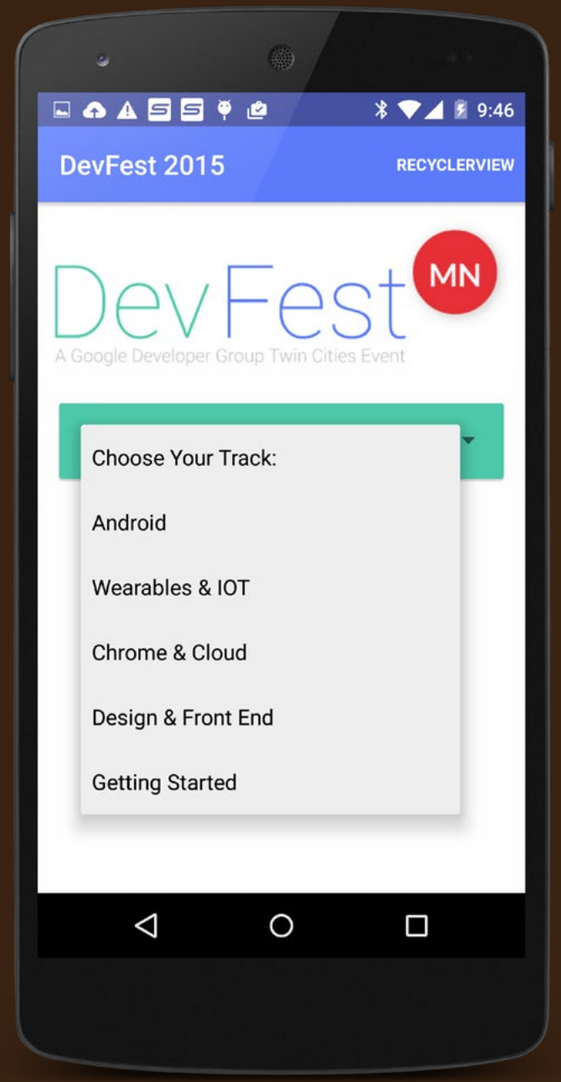
ViewAssertions class implements ***matches*** which is the ViewAssertion you will use most of the time

```
onView(...).check(matches(Matcher matcher));
```

Use **ViewMatchers** class again to assert that view's state

```
onView(...).check(matches(withText(R.string.dev_fest)));
```

What About AdapterViews?



AdapterViews

Only a subset of the AdapterView children may be loaded into the current view hierarchy.

If you know your view will be displayed, you can use onView.

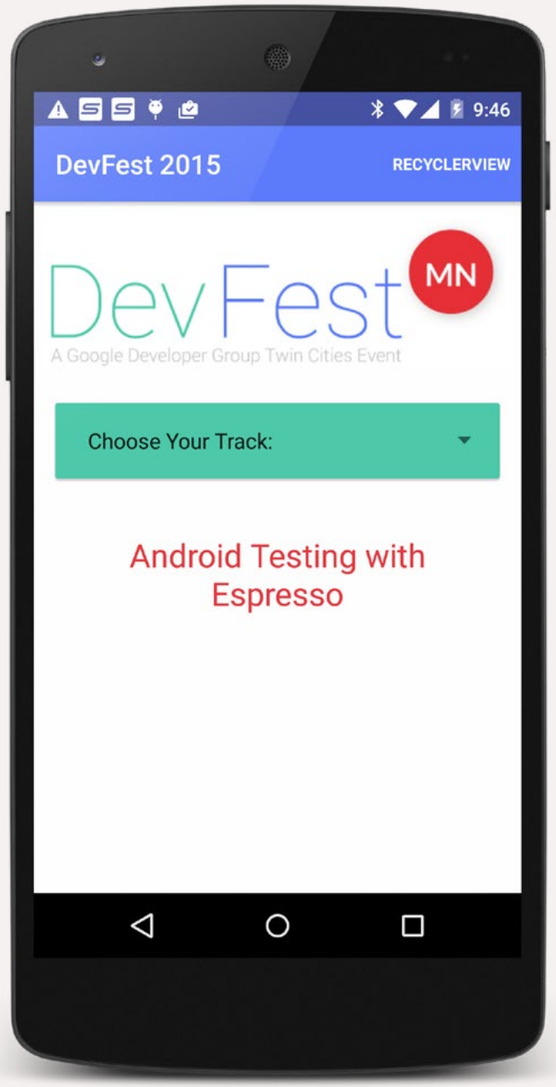
Or use onData to load the adapter item prior to operating on it

```
onData(allOf(is(instanceOf(String.class)), is(“GDG”))).perform(click());
```

RecyclerView is not an AdapterView

Example App

```
public class MainActivityTest extends ActivityInstrumentationTestCase2<MainActivity> {

    public MainActivityTest() {
        super(MainActivity.class);
    }

    @Override
    public void setUp() throws Exception {
        super.setUp();
        getActivity();
    }

    public void testHeader() {
        // test is image displayed
        String test = getActivity().getString(R.string.dev_fest_image);
        onView(withContentDescription(test)).check(matches(isDisplayed()));
    }

    public void testSpinner() {
         // click spinner
        onView(withId(R.id.spinner)).perform(click());
        // click view from choices now on screen
        onData(allOf(is(instanceOf(String.class)), is("Android"))).perform(click());
        // assert text has changed
        onView(withId(R.id.test_results)).check(matches(withText(R.string.android)));
    }
}
```



Synchronization

Main Ui Thread
Vs
Instrumentation Thread

Espresso Synchronization

Espresso waits for UI operations

Waits for the default AsyncTask thread pool

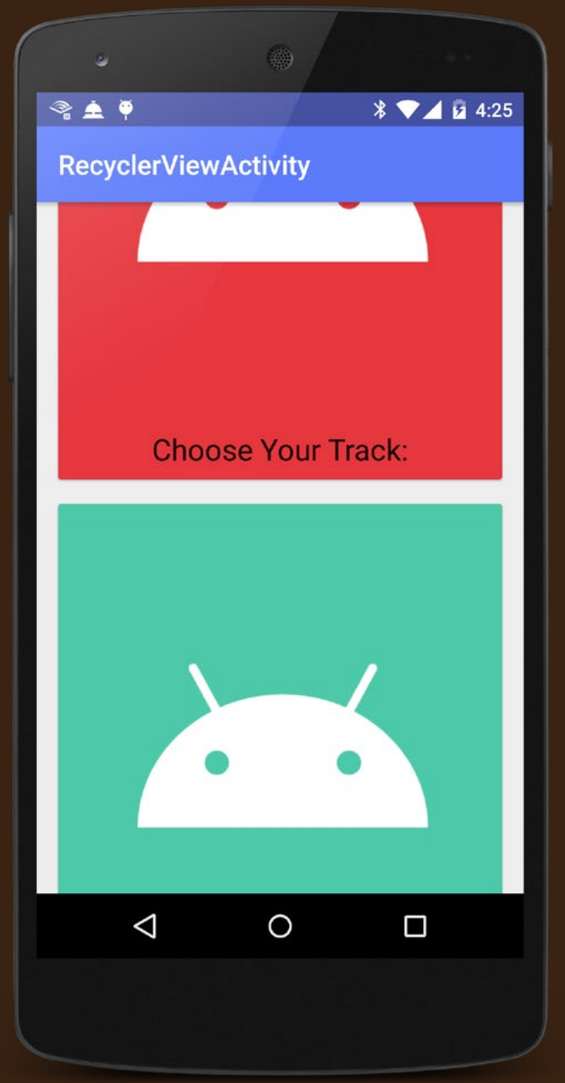
Implement the **IdlingResource** to handle all other background operations (i.e. IntentService)

```
registerIdlingResources(new DataIdlingResources(...));
```

The **CountingIdlingResource** (from Espresso contrib) can be used for simple counting resources

```java
public class DataIdlingResource implements IdlingResource {

    ResourceCallback resourceCallback;

    @Override
    public String getName() {
        // Returns the name of the resources (used for logging and idempotency of registration)
        return "DataIdlingResource";
    }

    @Override
    public boolean isIdleNow() {
        // add idle logic
        boolean idle = false;
        // when transitioning to idle
        if(idle) {
            resourceCallback.onTransitionToIdle();
        }
        return idle;
    }

    @Override
    public void registerIdleTransitionCallback(ResourceCallback resourceCallback) {
        this.resourceCallback = resourceCallback;
    }
}
```

RecyclerView, and...



RecyclerView

It’s not an adapterView so can’t use OnData

Use onView to match your RecyclerView and then use **RecyclerViewActions** from Espresso contrib to perform View Actions.

```
// click item at position 0
onView(withId(R.id.recyclerView))
   .perform(RecyclerViewActions.actionOnItemAtPosition(0, click()));

// scroll to item so you can onView(...) it
onView(withId(R.id.recyclerView))
   .perform(RecyclerViewActions.scrollToPosition(0));
```

DrawerLayout

Espresso contrib also provides Matchers and DrawerActions for DrawerLayouts

```
openDrawer(R.id.drawer_layout);

// The drawer should now be open.
onView(withId(R.id.drawer_layout)).check(matches(isOpen()));

closeDrawer(R.id.drawer_layout);

// Drawer should be closed again.
onView(withId(R.id.drawer_layout)).check(matches(isClosed()));
```

Actionbar

Actionbar items work with onView

```
// Click the item
onView(withId(R.id.action_item)).perform(click());

// Open the overflow menu
openActionBarOverflowOrOptionsMenu(getInstrumentation().getTargetContext());

// Click the item now displayed
onView(withText("Overflow 1")).perform(click());
```

Works with Toolbar as well!

Pickers

Use PickerActions class with onView from Espresso contrib

```
// Time Picker
onView(withId(R.id.time_pick)).perform(PickerActions.setTime(10,30));

// Date Picker
onView(withId(R.id.date_pick)).perform(PickerActions.setDate(1974, 1, 30));
```

Testing Tips

Model tests with an emphasis on time saving. Maximize savings by writing tests as soon as possible.

Test activities individually for quicker tests. Any activity can be the entry point.

```
public class RecyclerViewActivityTest extends
    ActivityInstrumentationTestCase2<RecyclerViewActivity> {...}
```

Use **AndroidTestCase** for testing that only requires access to an activity and resources. Data storage, Networking, etc..

Further Down the Rabbit Hole

***Spoon** - distributing instrumentation tests to all your Androids*
http://square.github.io/spoon/

***Mockito** - mocking framework for unit tests in Java*
http://mockito.org/

***Dagger** - dependency injector for Android and Java*
http://square.github.io/dagger/

***AndroidJUnitRunner** - new JUnit3 and JUnit4 test runner*
http://d.android.com/reference/android/support/test/runner/AndroidJUnitRunner.html