Android Libraries

I wish I knew when I started

github.com/ChrisGuzman/taasty

Chris Guzman

Developer Advocate

Nexmo

@speaktochris | chris-guzman.com


nexmo®
The Vonage® API Platform

Starting a new app?

Not sure what libraries to use?

A lot of apps do similar things

→ Manage multiple views

→ Load and display images

→ Fetch data from an API

→ Parse JSON

→ Start new activities with extras

→ Persist data to storage

These libraries prevent you reinventing the wheel with every project.

→ Butter Knife v8.5.1

→ Picasso v2.5.2

→ Gson v2.8.0

→ Retrofit v2.3.0

→ Realm v3.3.1

→ Dart & Henson v2.0.2

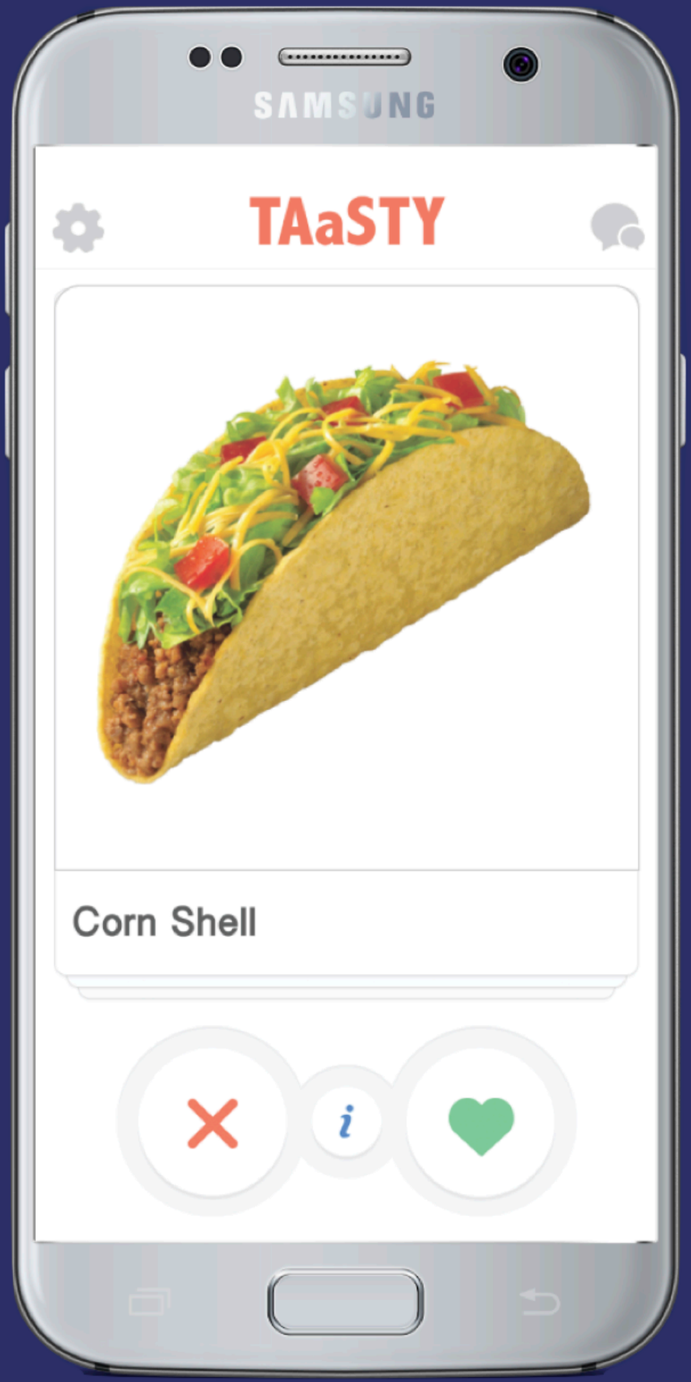
The 45 ~~Hour~~ Minute Hackathon

What should I make?

Introducing TaaSTY

Tacos as a Service To You

Tinder for Tacos


SAMSUNG
TAaSTY
Corn Shell

Step 1

Set up our views

```
<LinearLayout ... android:orientation="vertical">
  <ImageView android:id="@+id/taco_img" .../>
  <TextView android:id="@+id/description" .../>
  <LinearLayout android:orientation="horizontal" .../>
    <Button android:id="@+id/reject" .../>
    <Button android:id="@+id/save" .../>
  </LinearLayout>
</LinearLayout>
```

Step 2

Use views

Butter Knife

Use annotations to write less boilerplate code

→ How many times have you wrote findViewById?

→ No additional cost at run-time - does not slow down your app at all!

→ Improved View lookups

→ Improved Listener attachments

→ Improved Resource lookups

Bind views in an activity

```
<TextView android:id="@+id/name"
    ...
    />

public class MainActivity extends Activity {
    @BindView(R.id.name) TextView name;

    @Override
    protected void onCreate(Bundle bundle) {
        ...
        ButterKnife.bind(this);
        name.setText("Tofu with Cheese on a tortilla");
    }
}
```

ButterKnife.bind(this)

Generates code that looks up views/resources and saves them as a property on the Activity.

```
public void bind(MainActivity activity) {
  activity.description =
   (android.widget.TextView) activity.findViewById(2130968577);
}
```

Bind and unbind views in a fragment

```
public class TacoFragment extends Fragment {
  @BindView(R.id.tags) EditText tags;

  @Override
  public View onCreateView(args...) {
    ...
    ButterKnife.bind(this, parentView);
    //Important!
    unbinder = ButterKnife.bind(this, parentView);
    tags.setHint("Add tags. Eg: Tasty!, Want to try")
    return view;
  }
 }
```

```
@Override
public void onDestroyView() {
  super.onDestroyView();
  //sets the views to null
  unbinder.unbind();
}
```

Event listeners

```
@OnClick(R.id.save)
public void saveTaco(Button button) {
  button.setText("Saved!");
}
```

Arguments to the listener method are optional

```
@OnClick(R.id.reject)
public void reject() {
  Log.d("RejectBtn", "onClick")
}
```

Butter Knife also supports:

- OnLongClick
- OnEditorAction
- OnFocusChange
- OnItemClick
- OnItemLongClick
- OnItemSelected
- OnPageChange
- OnTextChanged
- OnTouch
- OnCheckedChanged

Inject resources once, no need to save them as member variables!

```
class MainActivity extends Activity {
  @BindString(R.string.title) String title;
  @BindDrawable(R.drawable.star) Drawable star;
  // int or ColorStateList
  @BindColor(R.color.guac_green) int guacGreen;
  // int (in pixels) or float (for exact value)
  @BindDimen(R.dimen.spacer) Float spacer;
}
```

Group views together:

```
@OnClick({ R.id.save, R.id.reject})
public void saveOrReject(View view) {
  if (view.getId() == R.reject) {
    Toast.makeText(this, "Ew Gross!", LENGTH_SHORT).show();
  }
  else {
    Toast.makeText(this, "Yummy :)", LENGTH_SHORT).show();
  }
  //TODO: implement
  ButterKnife.apply(actionButtons, DISABLE);
  getNewTaco();
}
```

Act on list of views at once with ButterKnife.apply.

```
@BindViews({R.id.save, R.id.reject})
List<Button> actionButtons;

ButterKnife.apply(actionButtons, View.ALPHA, 0.0f);
```

Action and Setter interfaces allow specifying simple behavior.

```
ButterKnife.apply(actionButtons, DISABLE);
ButterKnife.apply(actionButtons, ENABLED, false);
```

```
static final ButterKnife.Action<View> DISABLE = new ButterKnife.Action<View>() {
  @Override public void apply(View view, int index) {
    view.setEnabled(false);
  }
};
static final ButterKnife.Setter<View, Boolean> ENABLED = new ButterKnife.Setter<View, Boolean>() {
  @Override public void set(View view, Boolean value, int index) {
    view.setEnabled(value);
  }
};
```

```
private void getNewTaco() {
  //TODO: implement
  setTacoImage();
}
```

Step 3

Add pictures of tasty tacos

Picasso

Download and display images with ease!

→ makes HTTP Requests

→ caches the images

→ easy resizing/cropping/centering/scaling

→ takes care of downloading off the main thread

→ properly recycles views in RecyclerView

Pop quiz

Which do you prefer?

```
private Bitmap DownloadImage(String url)
{
    Bitmap bitmap = null;
    InputStream in = null;

    try
    {
        in = OpenHttpGETConnection(url);
        bitmap = BitmapFactory.decodeStream(in); in.close();
    }
    catch (Exception e)
    {
        Log.d("DownloadImage", e.getLocalizedMessage());
    }

    return bitmap;
}
```

NO

Or...

```
Picasso.with(context)
        .load("http://placekitten.com/200/300")
        .into(imageView);
```

But wait there's more!

```
.placeholder(R.mipmap.loading) //can be a resource or a drawable
.error(R.drawable.sad_taco) //fallback image if error
.fit() //reduce the image size to the dimensions of imageView
.resize(imgWidth, imgHeight) //resizes the image in pixels
.centerCrop() //or .centerInside()
.rotate(90f) //or rotate(degrees, pivotX, pivotY)
.noFade() //don't fade all fancy-like
```

Not just for loading images from the web

```
Picasso.with(context).load(R.drawable.salsa).into(imageView1);
Picasso.with(context).load("file:///asset/salsa.png").into(imageView2);
Picasso.with(context).load(new File(...)).into(imageView3);
```

```
//Butter Knife!
@BindView(R.id.taco_img) ImageView tacoImg;

 private void setTacoImage() {
   Picasso.with(context)
          .load("http://tacoimages.com/random.jpg")
          .into(tacoImg);
 }

private void getNewTaco() {
  setTacoImage();
  //TODO: implement
  loadTacoDescription();
}
```


DETOUR

Step 4

Set up models

Get ready for models by setting up what JSON will look like

Gson

Convert JSON to a java object and vice versa!

→ Without requiring you place Java annotations in your classes

→ Super performant

→ Commonly used

Example time!

```
class Taco {
  private String name;
  private String url;
  //not included in JSON serialization or deserialization
  private transient String imageUrl;
  Taco(args...) {
    ...
  }
}
```

```
// Serialize to JSON
Taco breakfastTaco = new Taco(
    "Eggs with syrup on pancake",
    "tacofancy.com/breakfast",
    "imgur.com/123");
Gson gson = new Gson();
String json = gson.toJson(breakfastTaco);

json = {
  "name":"Eggs with syrup on pancake",
  "url": "tacofancy.com/breakfast"
 }

// Deserialize to POJO
Taco yummyTaco = gson.fromJson(json, Taco.class);
// ==> yummyTaco is just like breakfastTaco without imagUrl
```

Benefits

→ All fields in the current class (and from all super classes) are included by default

→ Supports multi-dimensional arrays

Gotchas

→ While serializing, a null field is skipped from the output

→ While deserializing, a missing entry in JSON results in setting the corresponding field in the object to null

Cool customization

```
//Set properties to null instead of ignoring them
Gson gson = new GsonBuilder().serializeNulls().create();

//Keep whitespace
Gson gson = new GsonBuilder().setPrettyPrinting().create();
```

Need to rename a variable from an API?

```
public class Taco {

    @SerializedName("serialized_labels")
    private String tags;

}
```

Or use a custom date format?

```
public String DATE_FORMAT = "yyyy-MM-dd";

GsonBuilder gsonBuilder = new GsonBuilder();
gsonBuilder.setDateFormat(DATE_FORMAT);
Gson gson = gsonBuilder.create();
```


END
DETOUR

```
private void getNextTaco() {
   setTacoImage();
   //TODO: implement
   loadTacoDescription();
}
```

Step 5

Get taco recipe from web API

Retrofit

Stop using AsyncTask

Please, just stop.

The better way

→ Typesafe

→ Built in support for:

→ Authentication

→ parse JSON to POJOs

→ Supports RxJava

→ Can be executed synchronously or asynchronously

API Endpoints

```
public interface TacoApi {
    // Request method and URL specified in the annotation
    // Callback for the parsed response is the last parameter

    @GET("random")
    Call<Taco> randomTaco(@Query("full-taco") boolean full);

    @GET("contributions/{name}")
    Call<Contributor> getContributors(@Path("name") String username);

    @POST("recipe/new")
    Call<Recipe> createRecipe(@Body Recipe recipe);

    @GET("contributions")
    Call<List<Contributor>> getContributors();
}
```

Getting JSON synchronously

```
Retrofit retrofit = new Retrofit.Builder()
    .baseUrl("http://taco-randomizer.herokuapp.com/")
    .addConverterFactory(GsonConverterFactory.create())
    .build();

// Create an instance of our TacoApi interface.
TacoApi tacoApi = retrofit.create(TacoApi.class);

// Create a call instance for a random taco
Call<Taco> call = tacoApi.randomTaco(false);

// Fetch a random taco
// Do this off the main thread
Taco taco = call.execute().body();
```

POSTing JSON Async

```
Recipe recipe = new Recipe();
Call<Recipe> call = tacoApi.createRecipe(recipe);
call.enqueue(new Callback<Recipe>() {
  @Override
  public void onResponse(Call<Recipe> call, Response<Recipe> response) {

  }

  @Override
  public void onFailure(Call<Recipe> call, Throwable t) {

  }
```

Annotations

→ `@Path`: variable substitution for the API endpoint.

→ `@Query`: Add a query parameter.

→ `@Body`: Payload for the POST call (serialized from a Java object to a JSON string)

→ `@Headers`: Add a header to the HTTP call

→ `@Url`: Pass in a dynamic url

Cool tricks

```
//Change the base url
@POST("http://taco-randomizer.herokuapp.com/v2/taco")
private Call<Taco> getFromNewAPI();

//Add headers
@Headers({"User-Agent: tacobot"})
@GET("contributions")
private Call<List<Contributor>> getContributors();
```

```
private void getNextTaco() {
  ...
  loadTacoDescription();
}

private void loadTacoDescription() {
  Call<Taco> call = tacoApi.randomTaco(true);
  call.enqueue(new Callback<Taco>() {
    @Override
    public void onResponse(Call<Taco> call, Response<Taco> response) {
      //Set description from response
      Taco taco = response.body;
      //TODO: implement
      saveTaco(taco);
    }

    @Override
    public void onFailure(Call<Taco> call, Throwable t) {
      //Show error
    }
}
```

Save taco recipe for later!

Realm

Replacement for sqlite

→ Works by extending your models

→ Made for mobile

→ Most queries in Realm are fast enough to be run synchronously

→ Can have multiple realm database in an app

```
public class Taco extends RealmObject {
  private String name;
  private String imageUrl;
  private String url;
  //getters and setters
}
```

Set-up Realm

```
Realm.init(this);
// Get a Realm instance for this thread
Realm realm = Realm.getDefaultInstance();
```

Persist to db

```
// Persist your data in a transaction
realm.beginTransaction();

// Persist unmanaged objects
final Taco managedTaco = realm.copyToRealm(unmanagedTaco);

// Create managed objects directly
Taco taco = realm.createObject(Taco.class);

realm.commitTransaction();
```

Accessing Data

```
//find all favorite tacos
final RealmResults<Taco> likedTacos =
 realm.where(Taco.class).equalTo("favorite", true).findAll();
```

Typical SQL relationships

→ One to One

→ One to Many

→ Many to One

→ Many to Many

Conditions

→ between()

→ greaterThan(), lessThan()

→ greaterThanOrEqualTo() & lessThanOrEqualTo()

→ equalTo() & notEqualTo()

→ contains()

→ beginsWith() & endsWith()

Writing data

```
//Transaction block
realm.executeTransaction(new Realm.Transaction() {
  @Override
  public void execute(Realm realm) {
    Taco taco = realm.createObject(Taco.class);
    taco.setName("Spaghetti Squash on Fresh Corn Tortillas");
    taco.setUrl("http://tacofancy.com/squash");
  }
});
```

```
//Async
realm.executeTransactionAsync(new Realm.Transaction() {
        public void execute(Realm bgRealm) {
            Taco taco = bgRealm.createObject(Taco.class);
            taco.name("Spaghetti Squash on Fresh Corn Tortillas");
            taco.setUrl("http://tacofancy.com/squash");
        }
    }, new Realm.Transaction.OnSuccess() {
        public void onSuccess() {
            // Transaction was a success.
        }
    }, new Realm.Transaction.OnError() {
        public void onError(Throwable error) {
            // Transaction failed and was automatically canceled.
        }
    });
```

Fact:

Tacos have many ingredients

```
public class Taco extends RealmObject {
   ...
   private List<Ingredient>
   ...
}

public class Ingredient extends RealmObject {
   private String name;
   private URL url;
}
```

```
RealmResults<Taco> limeTacos = realm.where(Taco.class)
                                    .equalTo("ingredients.name", "Lime")
                                    .findAll();
```

Delete results

```
// All changes to data must happen in a transaction
realm.executeTransaction(new Realm.Transaction() {
    @Override
    public void execute(Realm realm) {
        // remove single match
        limeTacos.deleteFirstFromRealm();
        //or limeTacos.deleteLastFromRealm();

        // remove a single object
        Taco fishTaco = limeTacos.get(1);
        fishTaco.deleteFromRealm();

        // Delete all matches
        limeTacos.deleteAllFromRealm();
    }
});
```

Data change listeners

Can be attached to `RealmObject` or `RealmResults`

```
limeTacos.addChangeListener(
  new RealmChangeListener<RealmResults<Taco>>() {
    @Override
    public void onChange(RealmResults<Taco> tacosConLimon) {
      //tacosConLimon.size() == limeTacos.size()

      // Query results are updated in real time
      Log.d("LimeTacos", "Now we have" + limeTacos.size() + " tacos");
    }
  }
);
```

Tips

→ `@PrimaryKey` allows the use of `copyToRealmOrUpdate()`

→ Works with `Gson` and `Retrofit` easily

Prevent memory leaks

```
@Override
protected void onDestroy() {

    // Remove the listener.
    realm.removeChangeListener(realmListener);
    //or realm.removeAllChangeListeners();

    // Close the Realm instance.
    realm.close();
    ...
}
```

Gotchas

→ No support for list of `String` or primitives

→ In the meantime add this Gson adapter

→ Need to save data to realm after getting response from `Retrofit`

→ Large datasets or complex queries should be run on background thread

```
//TODO: implement
goToTacoDetailActivity();
```

Dart + Henson

Inspired by Butter Knife

Inject intent extras as a property on an object

Benefits

→ readable DSL for passing extras to intent

→ stop wasting time and write less of this:

```
intent.putExtra("TACO_NAME", "Salsa Verde");
tacoName = getIntent().getExtras().getString("TACO_NAME");
```

```java
public class TacoDetailActivity extends Activity {
  @InjectExtra String name;
  @Nullable @InjectExtra String imageUrl;
  //default value if left null
  @Nullable @InjectExtra String tag = "taco";
  //Ingredient implements Parcelable
  @Nullable @InjectExtra Ingredient ingredient;

  @Override
  public void onCreate(Bundle bundle) {
    super.onCreate(bundle);
    Dart.inject(this);
    //TODO use member variables
    ...
  }
}
```

Generate intent builders

```
//Start intent for TacoDetailActivity
Intent intent = Henson.with(this)
    .gotoTacoDetailActivity()
    .name(taco.getName())
    .url(taco.getUrl())
    .imageUrl(taco.getImageUrl())
    .build();
// tag is null
startActivity(intent);
```

Want to use Henson for an activity without injectable extras?

Annotate the activity with `@HensonNavigable`

Is that it?

```
grep 'TODO: implement'
=> 0 results
```

→ Butter Knife - Manage multiple views

→ Picasso - Load and display images

→ Gson - Parse JSON

→ Retrofit - Fetch data from an API

→ Realm - Persist data to storage

→ Dart & Henson - Start new activities with extras

Questions?

@speaktochris

chris-guzman.com