Android Components Documentation

Release 1.0.0

Eddilbert Macharia

Feb 19, 2019

Contents:

Documentation for android components

Fig. 1: Latest version of components

To be able to use any of this components.

- Add the JitPack repository to your build file.

```
allprojects {
        repositories {
                ...
                maven { url 'https://jitpack.io' }
        }
}
```

- Add the dependency you would like

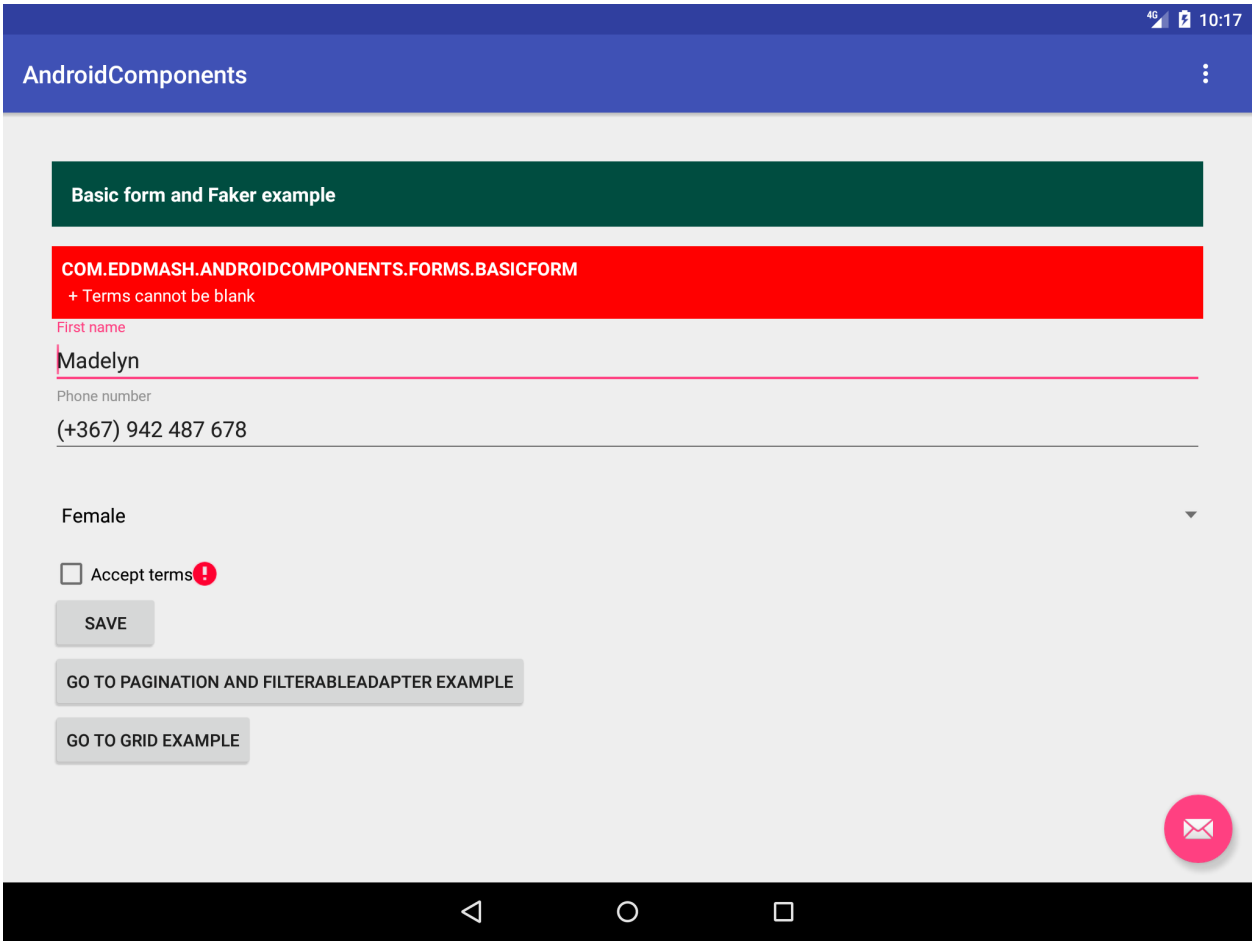
```
dependencies {
        compile 'com.github.eddmash.androidcomponents:activerecord:2.0.1'
        compile 'com.github.eddmash:androidcomponents:grid:2.0.1'
        compile 'com.github.eddmash:androidcomponents:form:2.0.1'
        compile 'com.github.eddmash:androidcomponents:validation:2.0.1'
        compile 'com.github.eddmash:androidcomponents:pagination:2.0.1'
        compile 'com.github.eddmash:androidcomponents:adapter:2.0.1'
        compile 'com.github.eddmash:androidcomponents:views:2.0.1'
}
```

CHAPTER 1

Android validation library documentation!

A lightweight and extensible android validation library.

It uses simple, straightforward validation methods with a focus on readable and concise syntax.



1.1 Usage

Using this library boils down to this steps

- Create a validator object

```
Validator validator = new Validator();
```

- Add validation checks to the validator

```
// the editText objects to validate
EditText nameEditText = (EditText) editText.findViewById(R.id.name);
Spinner ageSpinner = (Spinner) editText.findViewById(R.id.spinner);

// ... using check objects
validator.addCheck(new NotEmptyCheck(nameEditText, "name cannot be blank");
validator.addCheck(new NotEmptyCheck(ageSpinner, "age cannot be blank");
```

Learn more about available checks

- Validate

To run the validations involve the validators `validate()` method.

```
validator.validate()
```

This method returns `true` if the validation passed or `false` if the validations failed.

1.2 Handle the errors

Incase of validation failure, the validation errors can be accessed via the *`getErrors()`* method.

This library comes with a convenience class *`ErrorRenderer`*, which can be used to easily display the validation errors.

- Displaying errors.

```
// the layout where we display any validation errors
LinearLayout errorSpace = (LinearLayout) findViewById(R.id.error_base);
errorSpace.removeAllViews();// clear space first

if (validator.validate()) {
    // .. code to perform if validation passes
} else {

    // show the errors if validation failed
    // we use the renderer class to handle the display
    ErrorRenderer errorRenderer = new ErrorRenderer(this, validator);
    errorRenderer.render(errorSpace);
}
```

- Using ValidationListener to handle errors.

This version of *`validate()`* accepts a *`ValidationListener`* which has *`onValidationSuccess`* invoked when validation is a success. *`onValidationFailed`* invoked when validation fails methods.

```
// the layout where we display any validation errors
LinearLayout errorSpace = (LinearLayout) findViewById(R.id.error_base);
errorSpace.removeAllViews();// clear space first

validator.validate(new ValidationListener() {
    @Override
    public void onValidationSuccess(ValidatorInterface validatorInterface) {
        // on success code
    }

    @Override
    public void onValidationFailed(ValidatorInterface validatorInterface) {
        // show the errors if validation failed
        // we use the renderer class to handle the display
        ErrorRenderer errorRenderer = new ErrorRenderer(MainActivity.this,
                validatorInterface);
        errorRenderer.render(errorSpace);
    }
});
```

1.3 Checks

Contents

1.3.1 Introduction

A Check is a class that implements the *CheckInterface*. The interface defines a few methods.

- *run()* This is where all the logic of validation is placed. This method should return `true` if validation succeeds or `false` on failure.
- *getErrorMsg()* this should the error message to be used incase validation fails.
- *setError()* This method is invoked to with the error message above as an argument. This method should be used to set the error message on the editText.
- *clearError()* This method is invoked just be validation is run. it should be used to clear any previous error displayed on the validator.

1.3.2 Creating custom validations

You can create a custom checks by implementing *CheckInterface*, or by extending one of the *built-in checks*.

As an example we create a simple check that checks if a string contains only alphanumeric characters.

```
class ContainsAlphaNumericCheck implements CheckInterface {
    private EditText editText;

    public ContainsAlphaNumericCheck(EditText editText) {
        this.editText = editText;
    }

    @Override
    public boolean run() {
        String value = editText.getText().toString();
        return Pattern.compile("^[a-zA-Z0-9]+$").matcher(value).matches();
    }

    @Override
    public String getErrorMsg() {
        return "String contains an illegal character: it can only contain letters or␣
↪numbers.";
    }

    @Override
    public void setError(String error) {
        editText.setError(error);
    }
```

```
    @Override
    public void clearError() {
        editText.setError(null);
    }
}
```

1.3.3 Built-in Checks

com.eddmash.validation.checks

AllCheck

public class **AllCheck** extends *CheckCompound*

This is an implementation of *CheckCompound*.

Ensures all are validation checks are valid.

If the no checks are provided i.e. checkList is empty, validation will always pass for this check.

Constructors

AllCheck

public **AllCheck** (String *errorMessage*)

Methods

run

public boolean **run** ()

AnyCheck

public class **AnyCheck** extends *CheckCompound*

This is an implementation of *CheckCompound*.

Checks if at least one of the checks passed validation.

If the no checks are provided i.e. checkList is empty, validation will always fail.

Constructors

AnyCheck

public **AnyCheck** (String *errorMessage*)

Methods

run

public boolean **run**()

CheckCompound

public abstract class **CheckCompound** extends *CheckSingle*

This Implementation of this class gets the ability to run multiple checks as a unit.

The `AllCheck` implementation ensures that all the checks added to it pass the validation if any one of the fails the whole check fails.

The `AnyCheck` implentation ensures that atleast one passed the check meaning this check will pass validatioin if one of the checks within passed.

Fields

checkList

protected List<*CheckInterface*> **checkList**

Constructors

CheckCompound

public **CheckCompound**(String *errorMessage*)

Methods

addCheck

public void **addCheck**(*CheckInterface checkInterface*)

addChecks

public void **addChecks**(List<*CheckInterface*> *validationChecks*)

clearError

public void **clearError**()

disableCheck

public void **disableCheck**(*CheckInterface checkInterface*)

disableChecks

public void **disableChecks**(List<*CheckInterface*> *validationChecks*)

getErrorMsg

public String **getErrorMsg**()

setError

public void **setError**(String *error*)

CheckInterface

public interface **CheckInterface**
Classes that impliment this interface can be used for validation checks.

Methods

clearError

void **clearError**()
This method is invoked before validation is run, this is to allow clearing any validation indicators on the View.

getErrorMsg

String **getErrorMsg**()
The error message to use for the failed validations.

Returns the error message

run

boolean **run**()
This is where all the validation logic is placed.

Returns true if validation was a success else return false.

setError

void **setError**(String *error*)
Set the error message on the View being validated.

This will invoked when the validation starts, To clear out any previous errors displayed on the View. This is done by passing null as the error message

Its again invoked incase validation fails and error message need to be added to the View.

Parameters

- **error** – the error message that needs to be set on the View being validated

CheckSingle

public abstract class **CheckSingle** implements *CheckInterface*

This is the base class most of the validation Checks that relate to a single view.

CheckCompound if you like to validate multiple checks as one unit.

Methods

clearError

public void **clearError**()

getValue

protected String **getValue**()

Gets the value to be validated.

Returns value of the view

getView

protected TextView **getView**()

Gets the view we are working on.This can be anything that is a child of TextView e.g. EditText, CompoundButton like Checkboxes

Incase of a spinner you return the selected view like so.

(TextView) spinner.getSelectedView();

Returns the View from which to get value to isValid and also on which to set error by invoking **view.setError()**

setError

public void **setError**(String *error*)

EqualCheck

public class **EqualCheck** extends *NotEmptyCheck*

Checks if the provided values is matches to what view has.

Constructors

EqualCheck

public **EqualCheck**(EditText *view*, String *errorMessage*, int *valToEquate*)

EqualCheck

public **EqualCheck** (EditText *view*, String *errorMessage*, double *valToEquate*)

EqualCheck

public **EqualCheck** (EditText *view*, String *errorMessage*, String *valToEquate*)

Methods

run

public boolean **run** ()

GTCheck

public class **GTCheck** extends *NotEmptyCheck*
Check if the value on the view is greater than the provided value.

Constructors

GTCheck

public **GTCheck** (EditText *view*, String *errorMessage*, int *max*)

GTCheck

public **GTCheck** (Spinner *view*, String *errorMessage*, int *max*)

GTCheck

public **GTCheck** (EditText *view*, String *errorMessage*, double *max*)

GTCheck

public **GTCheck** (Spinner *view*, String *errorMessage*, double *max*)

Methods

run

public boolean **run** ()

GTECheck

public class **GTECheck** extends *NotEmptyCheck*
Check if the value on the view is greater than or equal the provided value.

Constructors

GTECheck

public **GTECheck** (EditText *view*, String *errorMessage*, int *max*)

GTECheck

public **GTECheck** (Spinner *view*, String *errorMessage*, int *max*)

GTECheck

public **GTECheck** (EditText *view*, String *errorMessage*, double *max*)

GTECheck

public **GTECheck** (Spinner *view*, String *errorMessage*, double *max*)

Methods

run

public boolean **run** ()

IsCheckedCheck

public class **IsCheckedCheck** implements *CheckInterface*
Checks if the provided compound button is checked.

Compound buttons include checkbox, radios etc

Constructors

IsCheckedCheck

public **IsCheckedCheck** (CompoundButton *compoundButton*, String *errorMessage*)

IsCheckedCheck

public **IsCheckedCheck** (CompoundButton *compoundButton*)

Methods

clearError

public void **clearError**()

getErrorMsg

public String **getErrorMsg**()

run

public boolean **run**()

setError

public void **setError**(String *error*)

IsFloatCheck

public class **IsFloatCheck** extends *NotEmptyCheck*
Checks if the value is a float value

Constructors

IsFloatCheck

public **IsFloatCheck**(EditText *view*, String *errorMessage*)

IsFloatCheck

public **IsFloatCheck**(Spinner *spinner*, String *errorMessage*)

Methods

run

public boolean **run**()

IsIntegerCheck

public class **IsIntegerCheck** extends *NotEmptyCheck*
Check if value is an integer.

Constructors

IsIntegerCheck

public **IsIntegerCheck**(EditText *view*, String *errorMessage*)

IsIntegerCheck

public **IsIntegerCheck**(Spinner *spinner*, String *errorMessage*)

Methods

getErrorMsg

public String **getErrorMsg**()

run

public boolean **run**()

LTCheck

public class **LTCheck** extends *NotEmptyCheck*
Check if the value on the view is less than the provided value.

Constructors

LTCheck

public **LTCheck**(EditText *view*, String *errorMessage*, int *min*)

LTCheck

public **LTCheck**(Spinner *view*, String *errorMessage*, int *min*)

LTCheck

public **LTCheck**(EditText *view*, String *errorMessage*, double *min*)

LTCheck

public **LTCheck**(Spinner *view*, String *errorMessage*, double *min*)

Methods

run

public boolean **run**()

LTECheck

public class **LTECheck** extends *NotEmptyCheck*
Check if the value on the view is less than or equal the provided value.

Constructors

LTECheck

public **LTECheck**(EditText *view*, String *errorMessage*, int *min*)

LTECheck

public **LTECheck**(Spinner *view*, String *errorMessage*, int *min*)

LTECheck

public **LTECheck**(EditText *view*, String *errorMessage*, double *min*)

LTECheck

public **LTECheck**(Spinner *view*, String *errorMessage*, double *min*)

Methods

run

public boolean **run**()

NotEmptyCheck

public class **NotEmptyCheck** extends *CheckSingle*
Ensure the view if not blank.

Fields

errorMessage

protected String **errorMessage**

Constructors

NotEmptyCheck

public **NotEmptyCheck**(EditText *editText*, String *errorMessage*)

NotEmptyCheck

public **NotEmptyCheck**(Spinner *spinner*, String *errorMessage*)

NotEmptyCheck

public **NotEmptyCheck**(CompoundButton *compoundButton*, String *errorMessage*)

Methods

getErrorMsg

public String **getErrorMsg**()

getValue

public String **getValue**()

getView

protected TextView **getView**()
Gets the editText we are working on.

run

public boolean **run**()

RegexCheck

public class **RegexCheck** extends *CheckSingle*
VAlidate a view against the provided rule or pattern

Constructors

RegexCheck

public **RegexCheck**(EditText *view*, String *errorMessage*, String *rule*)

RegexCheck

public **RegexCheck** (EditText *view*, String *errorMessage*, Pattern *pattern*)

RegexCheck

public **RegexCheck** (Spinner *spinner*, String *errorMessage*, String *rule*)

RegexCheck

public **RegexCheck** (Spinner *spinner*, String *errorMessage*, Pattern *pattern*)

Methods

getErrorMsg

public String **getErrorMsg** ()

getValue

public String **getValue** ()

getView

public TextView **getView** ()

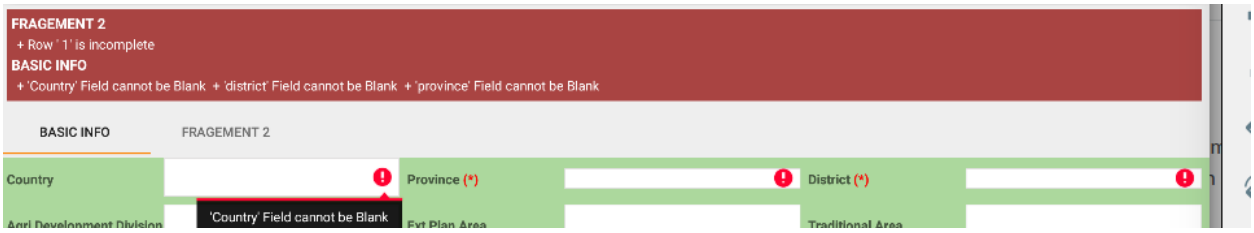
run

public boolean **run** ()

1.4 Renderer

Whilst the error message on individual fields might be enough for most cases. There are time you want to display the errors at convenient location that the user can editText them all at ones in at a centralised location.

A good example is when validating fields that *span multiple fragments*. You may want the use to be able to see all errors from each fragment at the very top.



A renderer is a class that implements `RendererInterface` it defines a method :

- *render(ViewGroup viewGroup)* which takes in a ViewGroup where the errors will be rendered.

A convenience class *ErrorRenderer*. It takes in the context and the validator object on its constructor.

```
// the layout where we display any validation errors
LinearLayout errorSpace = (LinearLayout) findViewById(R.id.error_base);
errorSpace.removeAllViews();// clear space first

ErrorRenderer errorRenderer = new ErrorRenderer(this, validator);
errorRenderer.render(errorSpace);
```

View Example use of ErrorRenderer.

1.4.1 com.eddmash.validation.renderer

ErrorRenderer

public class **ErrorRenderer** implements *RendererInterface*
Renders all errors found in the validator

Constructors

ErrorRenderer

public **ErrorRenderer**(Activity *activity*, *ValidatorInterface validatorInterface*)

Methods

render

public void **render**(ViewGroup *errorSpace*)

RendererInterface

interface **RendererInterface**
Class that implement this interface make it easy to render validation errors *ErrorRenderer*

Methods

render

void **render**(ViewGroup *errorSpace*)

1.5 Validating across fragments Example

Define an interface to be used to communicate between the activity and its fragments

```
public interface FormSaveListener {

    /**
     * Save the data saved by the user. this values are cleared once a save to the␣
↪database is made
     * otherwise this are saved for use on restore.
     *
     */
    void save();

    /**
    * Return a validator object that will be shared to the fragments
    */
    ValidationListener getValidator();
}
```

The activity that contains fragments should implement the *FormSaveListener* interface above.

The activity defines a validator object that this activity will use for validation.

```
public class DetailsActivity extends Activity implements FormSaveListener{
    ValidationListener mainValidator;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        if (getResources().getConfiguration().orientation
                == Configuration.ORIENTATION_LANDSCAPE) {
            // If the screen is now in landscape mode, we can show the
            // dialog in-line with the list so we don't need this activity.
            finish();
            return;
        }

        if (savedInstanceState == null) {
            FragmentA fragmentA = new FragmentA();
            fragmentA.setArguments(getIntent().getExtras());
            getFragmentManager().beginTransaction().add(R.id.fragment_location,␣
↪fragmentA).commit();
        }

        mainValidator = new Validator(this);
    }

    public ValidationListener getValidator(){
        return mainValidator;
    }

    public void save(){
         if (mainValidator.validate()) {
             // .. code to perform if validation passes
         } else {

             // show the errors if validation failed
             // we use the renderer class to handle the display
             ErrorRenderer errorRenderer = new ErrorRenderer(this, validator);
             errorRenderer.render(errorSpace);
         }
```

```
    }
}
```

On each fragement we create a validation object and give it a tag id, so that the errors for each fragment be grouped together.

On each fragement, onAttach ensure to cast the context to form FormSaveListener,

Add the fragments validation object to the activities validation object, This makes it to be validated at the time when the activity validation is done.

We run the activities validation when the submit button is clicked.

```
public class FragmentA extends ListFragment {
    FormSaveListener formSaveListener;
    ValidationListener validator;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        View editText = inflater.inflate(R.layout.fragment_a, container, false);

        validator = new Validator("Fragment_A", getActivity());
        formSaveListener.getValidator().addValidator(validator);

        setUpValidations()
        return editText;
    }

    @Override
    public void onResume() {
        super.onResume();
        Button submitBtn = (Button) editText.findViewById(R.id.form_submit_button);

        submitBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                formSaveListener.save();
            }
        });

    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        try {
            formSaveListener = (FormSaveListener) context;
        } catch (ClassCastException e) {
            throw new ClassCastException(context.toString() + " must implement␣
↪FormSaveListener");
        }
    }

    public void setUpValidations(){
        EditText nameEditText = (EditText) editText.findViewById(R.id.name);
        validator.setValidation(nameEditText, RegexTemplate.NOT_EMPTY, "name cannot␣
↪be empty");
    }
```

```
}
```

CHAPTER 2

Welcome to Android Forms documentation!

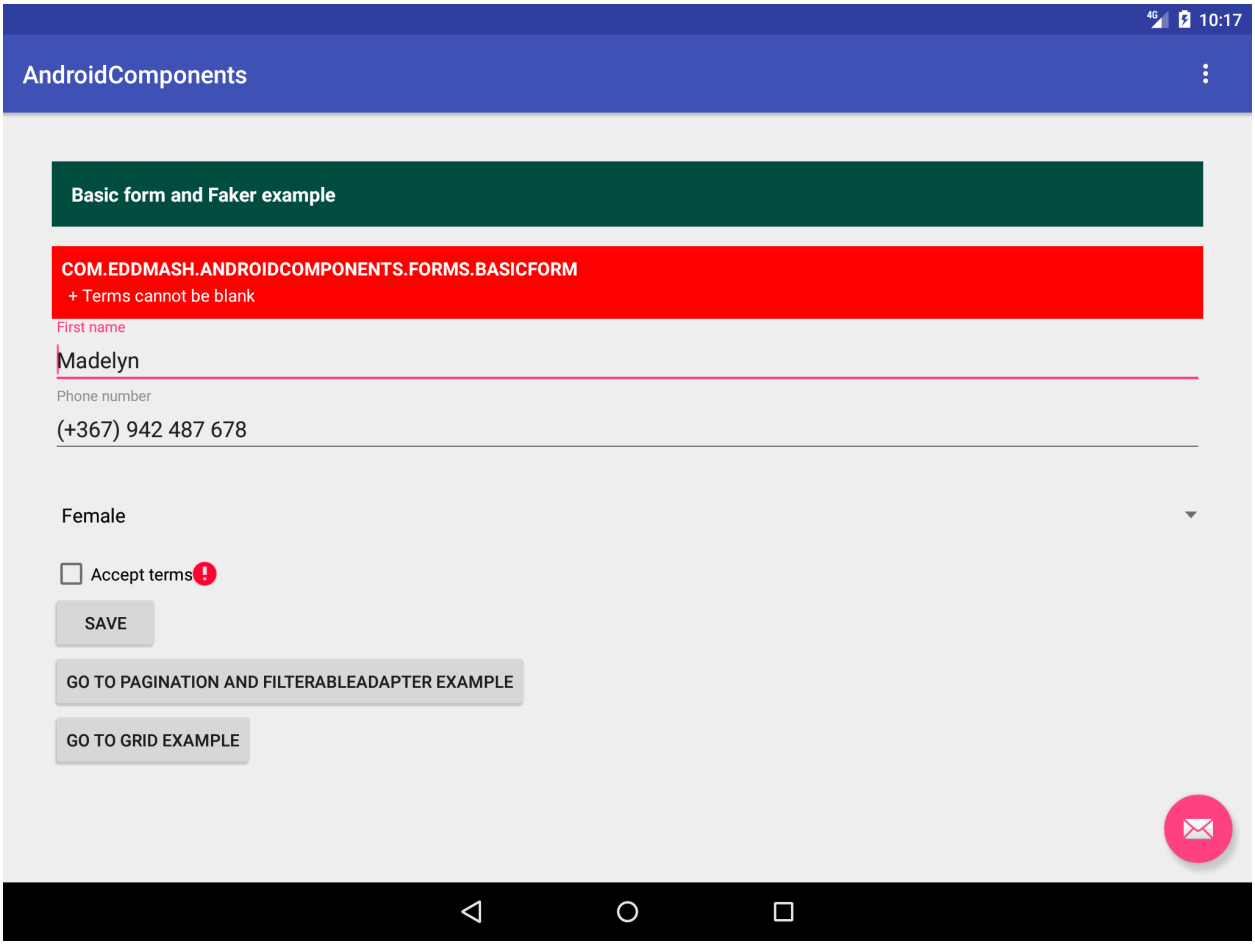
This library makes it easy to deal with views, especially if dealing with a large amount of views.

It provide a consistent and fluent way of setting, validating, retrieving and saving values from views.

With this library you not have to change or alter you layout in anyway, but you will be able to work with multiple forms on multiple fragments via *FormCollection*.

The library also comes an inbuilt *Data populator* to use when developing. This comes in handy when you want to prepopulate you form with sample data.

For validation the *Validation* is used.



2.1 Usage

To use this library is very easy

- Create *Form* class

```
private class BasicForm extends Form {
        public BasicForm() {
            super();
        }

        public BasicForm(ValidatorInterface validator) {
            super(validator);
        }

        @Override
        public void save() throws FormException {
            // implement the saving logic, you have access to
            // getValues() returns a map of where key is the name of the field and
↪the values
        }
}
```

- Create form object

```
BasicForm form = new BasicForm();
```

- Add *Field* to *FormInterface*

```
// get the views from the layout
Spinner genderSpinner = (Spinner) editText.findViewById(R.id.gender);
EditText fName = (EditText) editText.findViewById(R.id.firstname);
EditText phone_number = (EditText) editText.findViewById(R.id.phone_number);

// add views to the form
form.addField("gender", genderSpinner);
form.addField("firstname", fName);
form.addField("phone_number", fName);

// add validation check
form.addCheck(new NotEmptyCheck(gender,"Gender cannot be blank"));
form.addCheck(new NotEmptyCheck(fName,"Firstname cannot be blank"));
```

- Run *validation* and get the *values*.

```
if(form.isValid()){
    form.getValues()// returns a map of where key is the name of the field and the␣
↪values

}else{

     LinearLayout errorSpace = (LinearLayout) findViewById(R.id.error_base);
     errorSpace.removeAllViews();// clear space first

     ErrorRenderer errorRenderer = new ErrorRenderer(this, form.getValidator());
     errorRenderer.render(errorSpace);
}
```

- *Save* form *values*

```
if(form.isValid()){
    try{
        form.save() // save
    } catch (FormException e) {
        e.printStackTrace();
    }

}else{

    LinearLayout errorSpace = (LinearLayout) findViewById(R.id.error_base);
    errorSpace.removeAllViews();// clear space first

    ErrorRenderer errorRenderer = new ErrorRenderer(this, form.getValidator());
    errorRenderer.render(errorSpace);
}
```

2.2 Form Collection

This library also supports dealing with multiple forms at once.

A *FormCollection* class accepts *InnerForm*

This makes it possible:

- **for forms to depend on each other**
  - that is a form A cannot be validated before form B is validated.
  - that is a form A cannot be saved before form B is saved.
- multiple forms get validated together.
- multiple forms get saved together.

2.2.1 Usage

To use this library is very easy

- Create *InnerForm* class

In this example, the DependOnBasicForm requires the BasicForm to have been validated on the validation stage and be saved on saving stage.

```
 private class BasicForm extends InnerForm {
        public BasicForm() {
            super();
        }

        public BasicForm(ValidatorInterface validator) {
            super(validator);
        }

        @Override
        public String getIdentifier() {
            return "basic_form_id";
        }

        @Override
        public void save() throws FormException {
            // implement the saving logic, you have access to
            // getValues() returns a map of where key is the name of the field and␣
↪the values
        }
}

 private class DependOnBasicForm extends InnerForm {
        public DependOnBasicForm() {
            super();
        }

        public DependOnBasicForm(ValidatorInterface validator) {
            super(validator);
        }

        @Override
        public void save() throws FormException {
            // implement the saving logic, you have access to
            // getValues() returns a map of where key is the name of the field and␣
↪the values
        }
```

```
        @Override
        public String getIdentifier() {
            return "depend_on_basic_form_id";
        }

        @Override
        public String[] requires() {
            return new String[]{"basic_form_id"}; // make this form depend on the␣
↪basicform
        }
}
```

- Create *FormCollectionInterface* object

```
FormCollection formCollection = new FormCollection();
```

- Add *InnerForm* to *FormCollection*.

```
// create instance of inner form
BasicForm basicform = new BasicForm();
DependOnBasicForm dependOnBasicForm = new DependOnBasicForm();

// add inner form to collection
formCollection.addForm(basicform);
formCollection.addForm(dependOnBasicForm);
```

- Run *validation* and get the *values*.

```
if(formCollection.isValid()){
    formCollection.getValues()// returns a map of where key is the name of the field␣
↪and the values

}else{

     LinearLayout errorSpace = (LinearLayout) findViewById(R.id.error_base);
     errorSpace.removeAllViews();// clear space first

     ErrorRenderer errorRenderer = new ErrorRenderer(this, formCollection.
↪getValidator());
     errorRenderer.render(errorSpace);
}
```

- *Save* form *values*

```
if(formCollection.isValid()){
    try{
        formCollection.save() // save
    } catch (FormException e) {
        e.printStackTrace();
    }

}else{

    LinearLayout errorSpace = (LinearLayout) findViewById(R.id.error_base);
    errorSpace.removeAllViews();// clear space first

    ErrorRenderer errorRenderer = new ErrorRenderer(this, formCollection.
↪getValidator());
```

```
    errorRenderer.render(errorSpace);
}
```

2.3 Data Population

When developing, especially when working with form we need to prepopulate the form with sample data to ease up our development.

This is inspired by Php Faker which is also inspired by Ruby Faker . A whole lot of inspiring going on there.

The *faker* is meant to take care of this for us.

Lets populate our *basic form* form from earlier with sample data.

```
form // the basicform from earlier example

// create a populator object
DummyDataPopulator populator = new DummyDataPopulator();

// format how the data should look
// this will generate data with the format a mobile number of we use here in kenya
// +254 722 472 447
populator.setFieldPopulator("phone_number", new Telephone().setFormat("(+###) ### ###
↪###"));


// tell populator to populate the form.
// it will use the provider we set above for the phone number instead of the default␣
↪one which
// would be a random set of numbers
populator.populate(form);

// you could also populate single field on its own
populator.populate(form.getField("gender"));
```

The kind of data that is generated depends on the type of editText and the name of the field. e.g. if a editText is edittext of with inputype of number then the populator will generated numbers for that particular editText. All this is made possible by *Providers*

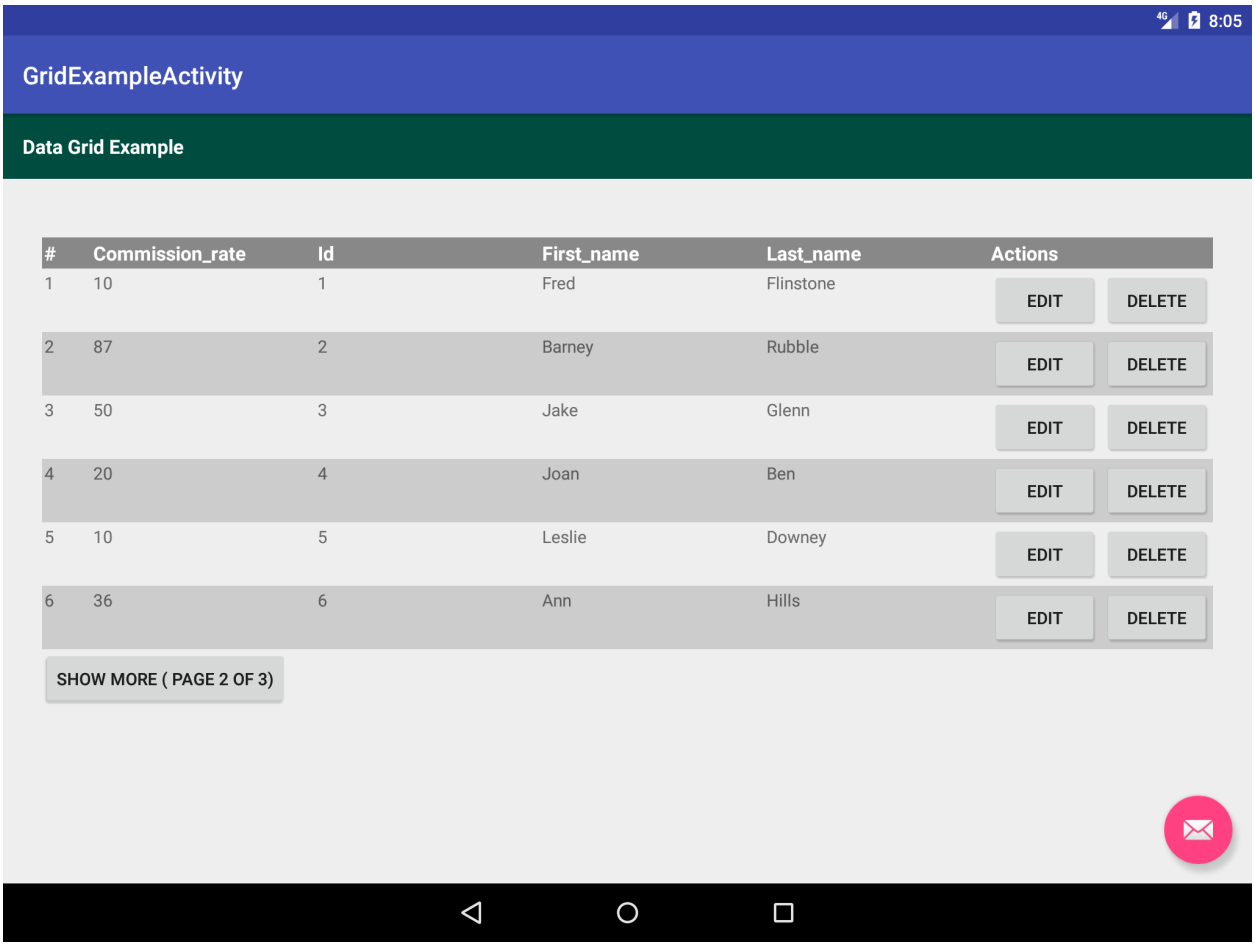
2.3.1 Faker Provider

A Provers is a class the implements the *ProviderInterface*.

CHAPTER 3

Pagination Documentation

This pagination library makes it easy to deal with paginating large set of data in a consistent way.



The paginators use *DataListener* to notify you of any changes that happen e.g

If data for first page is ready for use the *DataListener*

3.1 Paginate data from a List

To paginate a list of records we need to use *ListPaginator*.

```
List data = "..."
MyDataListener datalisterner = new MyDataListener();

ListPaginator pager = new ListPaginator(datalisterner);
pager.setData(data);
// set Page size
pager.setPageSize(20);

// invoke this method to trigger fetching of next page of data
pager.fetchNextPageData();
```

3.2 Paginate data from database

To fetch data from the database we need to use *SqlPaginator*.

```
// get and instance SQLiteDatabase
SQLiteDatabase db = "...";

MyDataListener datalisterner = new MyDataListener();

SqlPaginator pager = new SqlPaginator(datalisterner, db);
// set Page size
pager.setPageSize(20);

// pass in the sql the paginator will use to fetch data
String sql = "select * from employee where paid_date = ?";
String[] = new String[]{"1-12-2018"};
pager.query(sql, params);

// invoke this method to trigger fetching of next page of data
pager.fetchNextPageData();
```

3.3 Create a DataListener

A *Paginator* requires a *DataListener*

Create a DataListener this by implementing the *DataListener* interface.

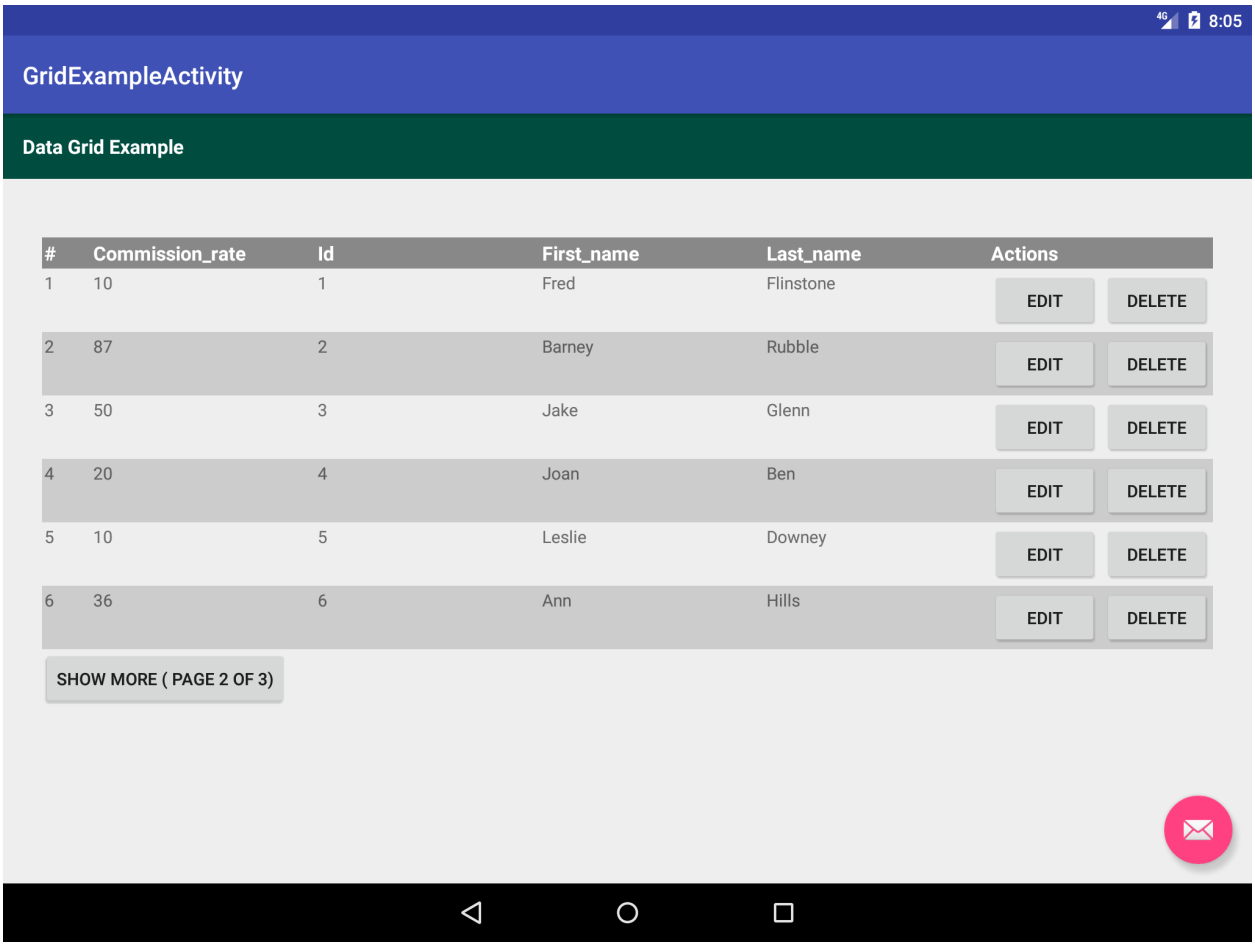
```
public class MyDataListener implements DataListener {

    @Override
    public void onLastPageDataLoaded() {
        // we dont need the load more button if we an on the last page of out data
        loadMoreBtn.setVisibility(View.GONE);
```

```
        adapter.notifyDataSetChanged();
    }

    @Override
    public void onFirstPageDataLoaded(boolean hasMorePages) {
        // update the load more button
        nextBtn(hasMorePages);
        adapter.notifyDataSetChanged();
    }

    @Override
    public void onNextPageDataLoaded() {
        adapter.notifyDataSetChanged();
    }

    void nextBtn(boolean hasMorePages) {
        String msg = getString(R.string.nextPageBtn) + " " + paginator.
↪getCurrentPageString();
        loadMoreBtn.setText(msg);
        if (hasMorePages) {
            loadMoreBtn.setVisibility(View.VISIBLE);
        } else {
            loadMoreBtn.setVisibility(View.GONE);
        }
    }

    @Override
    public void dataUpdate(List<Map> records) {
        // update the adapter with more data as we get them.
        adapter.update(records);
    }

    @Override
    public void preDataLoad(boolean hasMorePages) {
        nextBtn(hasMorePages);
    }
}
```

CHAPTER 4

DataGridView Documentation

DataGridView can be used to display a list or table of data records providing features like pagination.



Its takes a List of Maps that contains data and renders each row using a set of *ColumnInterface* presenting data in the form of a table.

4.1 Usage

The minimal code needed to use DataGridView is as follows

```
dataGridView = (DataGridView) findViewById(R.id.data_view);
dataGridView.setPageSize(3);
List<Map> data = "...";
dataGridView.setData(data);
```

Its also possible to override which columns are used in the grid and customize those columns as one wishes.

Assuming in the data provided to the DataGridView looks like this

```
[{"first_name":"jake", "age":"4"}, {"first_name":"joan", "age":"6"}, ]
```

4.2 Show specific columns

The earlier example will render both the firstname and age column on the grid we can tell the DataGridView to only render the firstname by *setColumns(Map) <DataGridView.setColumns(Map)* as shown below:

```
Map cols = new HashMap();
cols.put("first_name", new Column(this,"first_name","First Name"));
dataGridView.setColumns(cols);
```

4.3 Add columns

On top of columns that relate to the data prodided, with DataGridView you have the ability to *addColumn(BaseColumn, boolean)* e.g. action columns, serial columns or your own custom columns.

```
dataGridView.addColumn(new SerialColumn(this, "#"), DataGridView.LEFT);
dataGridView.addColumn(new EditActionColumn(this, "Edit"), DataGridView.RIGHT);
dataGridView.addColumn(new DeleteActionColumn(this, "Delete"), DataGridView.RIGHT);
```

4.4 Setting Data

The datagrid view supports two forms of setting data.

- set a list of map data like we have been doing above using the *setData()* method.
- set a list of map data like we have been doing above using the *setQuery()* method.

4.5 Columns

Learn more about columns *Columns*

4.5.1 Grid Columns

A column is a class that implements the *Columns*

The library comes with *Columns* but you can create custom columns to suit your needs.

In this example we create and EditActionColumn which basically displays a editbutton which if a user clicks the edit activity is launched.

This examples use the the *ActionColumn*.

```
public class EditActionColumn extends ActionColumn {
    public EditActionColumn(Context context, String name) {
        super(context, name);
    }

    @Override
    protected void onItemClick(View v, Map datum) {
        Intent intent = new Intent(getContext(), MainActivity.class);
        intent.putExtra("record_id", String.valueOf(datum.get("id")));
        startActivity(intent);
    }
}
```

com.eddmash.grids.columns

ActionColumn

public abstract class **ActionColumn** extends *Column*
ActionColumn displays action buttons such as update or delete for each row. find more at *ColumnInterface*

Constructors

ActionColumn

public **ActionColumn** (Context *context*, String *name*)

Methods

getDataView

public View **getDataView** (int *position*, Map *datum*)

getSearchView

public View **getSearchView** ()

onItemClick

protected abstract void **onItemClick**(View *v*, Map *datum*)
Action that will be taken when a *ColumnInterface.getDataView(int,Map)* is clicked.

Parameters

- **v** –
- **datum** –

BaseColumn

public abstract class **BaseColumn** implements *ColumnInterface*
The base class for all column defination.

Learn more about *Columns*

Fields

dataGridView

protected *DataGridView* **dataGridView**

Constructors

BaseColumn

public **BaseColumn**(Context *context*)

Methods

getContext

public Context **getContext**()

prepareDataView

protected TextView **prepareDataView**(TextView *view*, float *weight*)

prepareHeaderView

protected TextView **prepareHeaderView**(TextView *view*, float *weight*)

setDisplayView

public void **setDisplayView**(*DataGridView dataGridView*)

CheckColumn

public class **CheckColumn** extends *BaseColumn*

This renders a column of checkable columns which can be used to performa a check all or check some required fields.

learn more about `Columns`

Constructors

CheckColumn

public **CheckColumn**(Context *context*)

CheckColumn

public **CheckColumn**(Context *context*, String *name*)

Methods

getCheckedCheckboxes

public List<CheckBox> **getCheckedCheckboxes**()

getCheckedData

public List<Map> **getCheckedData**()

getDataView

public View **getDataView**(int *index*, Map *datum*)

getLabelView

public View **getLabelView**()

Column

public class **Column** extends *BaseColumn* implements *SearchableColumnInterface*

A column that displays data on the grid. this is an implementation of `ColumnInterface`

Fields

header

protected final Object **header**

name

protected final String **name**

Constructors

Column

public **Column**(Context *context*, String *valueKey*, Object *labelText*)

Methods

getDataView

public View **getDataView**(int *index*, Map *datum*)

getLabelView

public View **getLabelView**()

getSearchView

public View **getSearchView**()

ColumnInterface

public interface **ColumnInterface**

A column provides a view that will be displayed as a *ColumnInterface.getLabelView()* that will be rendered on the header row and *ColumnInterface.getDataView(int,Map)* that will be rendered on each of the data rows.

A simple column definition refers to an key/value in the data map of the *com.eddmash.grids.DataGridView* data list.

Methods

getContext

Context **getContext**()

Context on which *com.eddmash.grids.DataGridView* is being rendered on .

getDataView

View **getDataView**(int *index*, Map *datum*)
This is the view that will be used to create to show the for this column on each of its data rows.

Parameters

- **index** – the id of the row being populated.
- **datum** – data to be render of the row being populated.

Returns view to render for column on the row being populated.

getLabelView

View **getLabelView**()
Creates a view that will be used on the header row.

setDisplayView

void **setDisplayView**(*DataGridView dataGridView*)
The *com.eddmash.grids.DataGridView* that will be used to render this column.

Parameters

- **dataGridView** – *com.eddmash.grids.DataGridView*

DataColumnInterface

public interface **DataColumnInterface** extends *ColumnInterface*
Created by edd on 2/10/18.

SearchableColumnInterface

public interface **SearchableColumnInterface**

Methods

getSearchView

public View **getSearchView**()

SerialColumn

public class **SerialColumn** extends *Column*

Constructors

SerialColumn

public **SerialColumn** (Context *context*, String *name*)

Methods

getDataView

public View **getDataView** (int *index*, Map *datum*)

getLabelView

public View **getLabelView** ()

getSearchView

public View **getSearchView** ()

CHAPTER 5

Javadoc

5.1 com.eddmash.db

5.1.1 ActiveRecord

public class **`ActiveRecord`**

A simple set of helper methods to query for data on android sqlite database.

To get the instance of theis Active record use the getInstance method, this method takes just one parameter. an instance of SQLiteDatabase.

this class is implemented as a singleton meaning only one instance of ActiveRecord ever exists in your application life time.

NB:: the instance of SQLiteDatabase passed in getInstance method is destroyed once the garbage collector destroys the instance of the ActiveRecord.

Methods

all

public List<Map> **`all`** (String *tableName*, String[] *queryCols*)

Returns an list of maps, where the map represents each record in the database.

with keys of the map as the column name and values of the map as the values of the respective columns.

something like this:

[{id:1, username:ken, age:50}, {id:2, username:matt, age:70}]

Parameters

- **`tableName`** –
- **`queryCols`** –

all

public List<Map> **all** (String *tableName*)

Returns an list of maps, where the map represents each record in the database.

with keys of the map as the column name and values of the map as the values of the respective columns.

something like this:

[{id:1, username:ken, age:50}, {id:2, username:matt, age:70}]

Parameters

- **tableName** –

exists

public boolean **exists** (String *tableName*, String *field*, String *value*)

Returns value of the single column requested.

exists

public boolean **exists** (String *sql*, String[] *params*)

Returns value of the single column requested.

finalize

protected void **finalize** ()

find

public List<Map> **find** (String *sql*, String[] *args*)

Returns an list of maps, where the map represents each record in the database.

with keys of the map as the column name and values of the map as the values of the respective columns.

something like this:

[{id:1, username:ken, age:50}, {id:2, username:matt, age:70}]

Parameters

- **sql** –
- **args** –

get

public HashMap **get** (String *sql*, String[] *params*)

Returns a Map representing a single record based on the query.

Parameters

- **sql** –
- **params** – parameters to bind to the query

getDb

public SQLiteDatabase **getDb**()
REturn instance of SQLiteDatabase that the activerecord instance is using.

getInstance

public static *ActiveRecord* **getInstance**(SQLiteDatabase *database*)
Returns an instance of the the activerecord class

Parameters

- **database** –

getScalarInt

public int **getScalarInt**(String *sql*, String[] *params*)

5.2 com.eddmash.dialogs

5.2.1 AlertboxDialog

public class **AlertboxDialog** extends *GenericDialog*

Constructors

AlertboxDialog

public **AlertboxDialog**()

Methods

getContentLayout

protected int **getContentLayout**()

onViewReady

public void **onViewReady**(View *view*, Bundle *savedInstanceState*)

setMessage

public void **setMessage**(String *s*)

5.2.2 GenericDialog

public abstract class **GenericDialog** extends DialogFragment

A generic class that makes it easy to customize dialog boxes. example usage of an implementation *AlertboxDialog*

```
AlertboxDialog alertboxDialog = new AlertboxDialog();
alertboxDialog.disableButtons(true);
alertboxDialog.setTitle("NETWORK ERROR");
alertboxDialog.setIcon(R.drawable.fail);
alertboxDialog.setMessage(activity.getString(R.string.network_error));
alertboxDialog.show(activity.getSupportFragmentManager(), "network_error");
```

Fields

layout

protected View **layout**

leftButton

protected Button **leftButton**

rightButton

protected Button **rightButton**

Constructors

GenericDialog

public **GenericDialog**()

Methods

disableButtons

public void **disableButtons**(boolean *b*)

getContentLayout

protected int **getContentLayout**()

onCreateView

public View **onCreateView**(LayoutInflater *inflater*, ViewGroup *container*, Bundle *savedInstanceState*)

onViewCreated

public void **onViewCreated**(View *view*, Bundle *savedInstanceState*)

onViewReady

protected abstract void **onViewReady**(View *view*, Bundle *savedInstanceState*)

Add your logic to this method since at this point most of the work is done for you and the base layout has been polutated with your content layout

setContentLayout

public void **setContentLayout**(int *layoutID*)

setIcon

public void **setIcon**(int *drawable*)

setLeftButton

public void **setLeftButton**(String *label*, *ButtonClickedListener* *clickListener*)

SEt the listner to be invoked when the left button is clicked

Parameters

- **label** –
- **clickListener** –

setRightButton

public void **setRightButton**(String *label*, *ButtonClickedListener* *clickListener*)

set listener to be invoked when the right button is cliked.

Parameters

- **label** –
- **clickListener** –

setTitle

public void **setTitle**(String *title*)

setTitleBackground

public void **setTitleBackground**(int *titleBgColor*)

setTitleTextColor

public void **setTitleTextColor** (int *titleBgColor*)

5.2.3 GenericDialog.ButtonClickedListener

public interface **ButtonClickedListener**

Methods

onClick

abstract void **onClick** (View *view*, DialogFragment *dialog*)

5.2.4 GenericDialog.Dismiss

public static class **Dismiss** implements *ButtonClickedListener*

Methods

onClick

public void **onClick** (View *view*, DialogFragment *dialog*)

5.3 com.eddmash.form

5.3.1 Form

public abstract class **Form** implements *FormInterface*

Constructors

Form

public **Form** ()

Form

public **Form** (*ValidatorInterface validator*)

Methods

addCheck

public void **addCheck** (*CheckInterface check*)

addField

public void **addField**(String *colName*, View *view*)

addField

public void **addField**(*FieldInterface field*)

disableCheck

public void **disableCheck**(*CheckInterface check*)

getErrors

public Map<String, List> **getErrors**()

getField

public *FieldInterface* **getField**(String *fieldName*)

getFields

public Map<String, *FieldInterface*> **getFields**()

getIdentifier

public String **getIdentifier**()

getValidator

public *ValidatorInterface* **getValidator**()

getValue

public Object **getValue**(String *fieldName*)

getValues

public Map<String, Object> **getValues**()

isValid

public boolean **isValid**()

removeField

public void **removeField**(String *colName*)

removeField

public void **removeField**(*FieldInterface field*)

setData

public void **setData**(Map *data*)

setValue

public void **setValue**(String *fieldName*, Object *value*)

validate

public void **validate**()

5.3.2 FormAwareInterface

public interface **FormAwareInterface**

Any class the implements this interface gets the form passed to it via the *setForm(FormInterface)*.

This class come in handy when creating validation checks that need to be away of the form, especially when performing form wide validations like on the *FormInterface.validate()* method.

A good examples is the *Field* which implements this interface.

The following examples how this interface can be used to create form wide check.

```
public class BasicForm extends Form {

     @Override
     public void validate() {
         try {
             addCheck(new PasswordCheck(password1Edittext, password2Edittext,
             "Password should match"));
         } catch (FormException e) {
             e.printStackTrace();
         }
     }

     @Override
     public void save() throws FormException {

     }

}
```

A sample check that uses the FormAwareInterface interface

```
class PasswordCheck extends CheckSingle implements FormAwareInterface {
     private final EditText view;
     private final String errorMsg;
     private FormInterface form;

     public PasswordCheck(EditText view, String errorMsg) {
         this.view = view;
         this.errorMsg = errorMsg;
     }

     @Override
     public boolean run() {
         try {
             Map val = form.getValues();
             Object pass1 = val.get("password_1");
             Object pass2 = val.get("password_2");
             if (pass1.equals(pass2)) {
                 return true;
             }
         } catch (FormException e) {
             e.printStackTrace();
         }
         return false;
     }

     @Override
     public String getErrorMsg() {
         return errorMsg;
     }
     @Override
     protected TextView getView() {
         return view;
     }
     @Override
     public void setForm(FormInterface form) {
         this.form = form;
     }
 }
```

Methods

setForm

void **setForm**(*FormInterface form*)
The form that will hold this class is passed in.

Parameters

- **form** –

5.3.3 FormException

public class **FormException** extends Exception

Constructors

FormException

public **FormException**(String *message*)

5.3.4 FormInterface

public interface **FormInterface**

Methods

addCheck

void **addCheck**(*CheckInterface* *check*)
Add a validation check.

Parameters

- **check** –

addField

void **addField**(*FieldInterface* *field*)

addField

void **addField**(String *colName*, View *view*)

disableCheck

void **disableCheck**(*CheckInterface* *check*)
Disable a validation check

Parameters

- **check** –

getErrors

Map<String, List> **getErrors**()
Returns all the errors on the form after validation.

Returns the key is the form identifier and the values is a list of all the validation errors related with the form.

getField

FieldInterface **getField**(String *fieldName*)

getFields

Map<String, *FieldInterface*> **getFields**()

getIdentifier

String **getIdentifier**()
A unique identifier for this form.

getValidator

ValidatorInterface **getValidator**()
The validator this form will be using to validate the inner forms.

getValue

Object **getValue**(String *fieldName*)
Returns the value a particular field.

The returned values depends on the *field* some fields the values is a string whilst others its a list of string.

Consult the specific *field* to get the returned value.

Parameters

- **fieldName** –

Throws

- ***FormException*** –

getValues

Map<String, Object> **getValues**()
Return the values of each field on the form.

The returned values depends on the *field* some fields the values is a string whilst others its a list of string.

Consult the specific *field* to get the returned value.

Throws

- ***FormException*** –

Returns a map, where keys a field identifier used when adding field to form and values are the fields respective values.

isValid

boolean **isValid**()
This is the entry point for form validations.

It firsts invokes the *validate()* to get the form wide validation .

It the tells the validator to run the validation check.

Returns true only if the validation checks passed.

removeField

void **removeField**(String *replace*)

removeField

void **removeField**(*FieldInterface field*)

save

void **save**()
This is where you should put you saving logic.

throw FormException if validation fails.

Throws

- ***FormException*** –

setData

void **setData**(Map *data*)

setValue

void **setValue**(String *fieldName*, Object *value*)
Set value for a specific.

Parameters

- **fieldName** – the identifier to use to locate the field being set.
- **value** – the value being set, this depends on specific *field* .consult specific *field* to find expected value.

validate

void **validate**()
This is the right place to perform form wide validations. That is validating fields against each other, also validate against parent form fields.

At this point you have access to the getValues() of both parent form and current form you can use this values to compare against.

The recommend approach is to a create check that implements FormAwareInterface and add it to the validator.

This method is invoked before the field specific validations have been run.

5.4 com.eddmash.form.collection

5.4.1 FormCollection

public class **FormCollection** implements *FormCollectionInterface*
Use this class when you want to deal on multiple forms all at once.

Constructors

FormCollection

public **FormCollection**()

FormCollection

public **FormCollection**(*ValidatorInterface validator*)

Methods

addForm

public void **addForm**(*InnerFormInterface form*)

getForm

public *InnerFormInterface* **getForm**(String *identifier*)

getValidator

public *ValidatorInterface* **getValidator**()

isValid

public boolean **isValid**()

removeForm

public void **removeForm**(*InnerFormInterface form*)

save

public boolean **save**()

5.4.2 FormCollectionInterface

public interface **FormCollectionInterface**
Use this class when you want to deal on multiple forms all at once.

Methods

addForm

void **addForm**(*InnerFormInterface* *form*)
Add a form into the collection.

Parameters

- **form** –

Throws

- ***FormException*** –

getForm

InnerFormInterface **getForm**(String *identifier*)
Get a form in the collection by its identifier.

Parameters

- **identifier** –

Throws

- ***FormException*** –

getValidator

ValidatorInterface **getValidator**()
The validator this collection will be using to validate the inner forms.

This is basically a parent validator which calls the individual validators bound to each of the inner forms.

isValid

boolean **isValid**()
Entry point for validation of all the innerforms on this collection.

Runs validation for each of the inner forms.

removeForm

void **removeForm**(*InnerFormInterface* *form*)
REmove a from the collection.

Parameters

- **form** –

save

boolean **save**()

The entry point of form saving.

This method calls the save() of each innerform attached to this collection

Wrap this method in a transaction to ensure if any of the inner form fails to save, All the other innerforms arent saved. for consistency sake.

Throws

- ***FormException*** –

5.4.3 InnerForm

public abstract class **InnerForm** extends *Form* implements *InnerFormInterface*

This is basically *form*that has the capability of being used with a form collection.

Fields

form

protected *FormCollectionInterface* **form**

Constructors

InnerForm

public **InnerForm**()

InnerForm

public **InnerForm**(*ValidatorInterface validator*)

Methods

getParent

public *FormCollectionInterface* **getParent**()

setParent

public void **setParent**(*FormCollectionInterface form*)

5.4.4 InnerFormInterface

public interface **InnerFormInterface** extends *FormInterface*

This is basically *form*that has the capability of being used with a form collection.

This form also has the added capability of depending on another form thats in the same collection as the one it belongs to.

Methods

getParent

FormCollectionInterface **getParent**()

The collection form in which this form belongs to.

requires

String[] **requires**()

An string array of other inner forms that this form should depend on i.e. those forms should be validated before this during the validation stage and should be saved before this is saved.

5.5 com.eddmash.form.faker

5.5.1 Callback

public abstract class **Callback**

Methods

invoke

public abstract String **invoke**()

5.5.2 DummyDataPopulator

public class **DummyDataPopulator** implements *PopulatorInterface*

This is a minimalistic go at data faker.

This intention is to populate the FormInterface and FieldInterfaces.

Constructors

DummyDataPopulator

public **DummyDataPopulator**()

Methods

populate

public void **populate** (*FormInterface* *form*)

populate

public void **populate** (*FieldInterface* *field*)

setFieldProvider

public void **setFieldProvider** (String *name*, *ProviderInterface* *provider*)

5.5.3 FakerException

public class **FakerException** extends Exception

Constructors

FakerException

public **FakerException** ()

FakerException

public **FakerException** (String *message*)

5.5.4 Guess

class **Guess**

Constructors

Guess

Guess (*PopulatorInterface* *populator*)

Methods

guess

public String **guess** (String *name*, View *view*)

5.5.5 PopulatorInterface

public interface **PopulatorInterface**
Its responsible for populating the *FormInterface* or *FieldInterface* provided.

The populator uses *providers* to populate each field presented to the populator.

Methods

populate

void **populate**(*FormInterface form*)
Tell the populator to start the population on the specified form.

Parameters

- **form** –

Throws

- ***FormException*** –

populate

void **populate**(*FieldInterface field*)
Tell the populator to start the population on the specified field.

Parameters

- **field** –

Throws

- ***FormException*** –

setFieldProvider

void **setFieldProvider**(String *name*, *ProviderInterface provider*)
Set the provider to use the populator a specific field.

Parameters

- **name** – the name of the field that will use the provider given.
- **provider** – the provider to use instead of the default onces.

5.6 com.eddmash.form.faker.provider

5.6.1 Company

public class **Company** extends *Provider*
Generate data that relates to companies.

Constructors

Company

public **Company** (*PopulatorInterface* *populator*)

Company

public **Company** (*PopulatorInterface* *populator*, String *format*)

Methods

generate

public String **generate** ()

getCompany

public *Company* **getCompany** ()

getCompanySuffix

public *Company* **getCompanySuffix** ()

getJobTitles

public *Company* **getJobTitles** ()

5.6.2 CoordinatesProvider

public class **CoordinatesProvider** extends *Provider*

Fields

LATITUDE

String **LATITUDE**

LONGITUDE

String **LONGITUDE**

Constructors

CoordinatesProvider

public **CoordinatesProvider**(*PopulatorInterface* *populator*)

CoordinatesProvider

public **CoordinatesProvider**(*PopulatorInterface* *populator*, String *format*)

Methods

generate

public String **generate**()

getLatitude

public *ProviderInterface* **getLatitude**()

getLongitude

public *ProviderInterface* **getLongitude**()

5.6.3 DateProvider

public class **DateProvider** extends *Provider*

Fields

TIME_NOW

public static final String **TIME_NOW**

TODAY

public static final String **TODAY**

dateFormat

protected String **dateFormat**

timeFormat

protected String **timeFormat**

Constructors

DateProvider

public **DateProvider**(*PopulatorInterface* *populator*)

DateProvider

public **DateProvider**(*PopulatorInterface* *populator*, String *format*)

Methods

generate

public String **generate**()

getDate

public *DateProvider* **getDate**()

getDate

public *DateProvider* **getDate**(String *timeFormat*)

getTime

public *DateProvider* **getTime**()

getTime

public *DateProvider* **getTime**(String *timeFormat*)

5.6.4 InternetProvider

public class **InternetProvider** extends *Provider*

Constructors

InternetProvider

public **InternetProvider**(*PopulatorInterface* *populator*)

InternetProvider

public **InternetProvider**(*PopulatorInterface* *populator*, String *format*)

Methods

generate

public String **generate**()

getDomain

public *InternetProvider* **getDomain**()

getEmail

public *InternetProvider* **getEmail**()

getTld

public *InternetProvider* **getTld**()

5.6.5 LocationsProvider

public class **LocationsProvider** extends *Provider*

Fields

ADDRESS

public static final String **ADDRESS**

CITY

public static final String **CITY**

COUNTRY

public static final String **COUNTRY**

Constructors

LocationsProvider

public **LocationsProvider**(*PopulatorInterface populator*)

LocationsProvider

public **LocationsProvider**(*PopulatorInterface populator*, String *format*)

Methods

generate

public String **generate**()

getCity

public *LocationsProvider* **getCity**()

5.6.6 LoremProvider

public class **LoremProvider** extends *Provider*

Constructors

LoremProvider

public **LoremProvider**(*PopulatorInterface populator*)

LoremProvider

public **LoremProvider**(*PopulatorInterface populator*, String *format*)

Methods

generate

public String **generate**()

getWord

public *ProviderInterface* **getWord**()

getWords

public *LoremProvider* **getWords**()

getWords

public *LoremProvider* **getWords**(int *noOfWords*)

5.6.7 PersonProvider

public class **PersonProvider** extends *Provider*

Fields

EMAIL

public static final String **EMAIL**

FEMALE

public static final String **FEMALE**

FIRSTNAME

public static final String **FIRSTNAME**

LASTNAME

public static final String **LASTNAME**

MALE

public static final String **MALE**

NAME

public static final String **NAME**

PROVIDER_NAME

public static final String **PROVIDER_NAME**

Constructors

PersonProvider

public **PersonProvider**(*PopulatorInterface* *populator*)

PersonProvider

public **PersonProvider**(*PopulatorInterface* *populator*, String *format*)

Methods

generate

public String **generate**()

getFirstName

public *ProviderInterface* **getFirstName**()

getFirstName

public *ProviderInterface* **getFirstName**(String *gender*)

getFullName

public *ProviderInterface* **getFullName**()

getFullName

public *ProviderInterface* **getFullName**(String *gender*)

getLastName

public *ProviderInterface* **getLastName**()

getLastName

public *ProviderInterface* **getLastName**(String *gender*)

setGender

public *PersonProvider* **setGender**(String *gender*)

setType

public *PersonProvider* **setType**(String *type*)

5.6.8 Provider

public abstract class **Provider** implements *ProviderInterface*

An implimentation of `ProviderInterface`.

Go to `ProviderInterface` to learn more.

Fields

PATTERN

public static final String **PATTERN**

RANDOM_INT

public static final String **RANDOM_INT**

SEPARATOR

public static final String **SEPARATOR**

format

protected String **format**

populator

protected *PopulatorInterface* **populator**

Constructors

Provider

public **Provider**(*PopulatorInterface* *populator*)

Provider

public **Provider**(*PopulatorInterface* *populator*, String *format*)

Methods

getData

public String **getData**()

getPersonName

public String **getPersonName**()

getPopulator

public *PopulatorInterface* **getPopulator**()

mergeArrays

protected String[] **mergeArrays**(String[] *first*, String[] *second*)

parseFormat

protected String **parseFormat** (String *format*, *Callback* *callback*)

randomDouble

protected Double **randomDouble** ()

randomDouble

protected Double **randomDouble** (int *minNumber*, int *maxNumber*)

randomElement

protected String **randomElement** (String[] *strings*)

randomElement

protected String **randomElement** (String[] *strings*, int *count*)

randomElements

protected String[] **randomElements** (String[] *strings*)

randomElements

protected String[] **randomElements** (String[] *strings*, int *count*)

randomInt

protected Integer **randomInt** ()

randomInt

protected Integer **randomInt** (int *minNumber*, int *maxNumber*)

toString

public String **toString** ()

5.6.9 ProviderInterface

public interface **ProviderInterface**
This class is responsible for generating data.

The default implementation *Provider* uses the *ProviderInterface.generate()* to generate the actual data and then use *ProviderInterface.getData()* to format the data.

That is each method on the default providers is used a setter of the type of data to generate.

Methods

generate

String **generate**()
This should return the default values the populator will use.

getData

String **getData**()
Returns the generated value by the provider.

5.6.10 RandomNumberProvider

public class **RandomNumberProvider** extends *Provider*

Fields

DECIMAL

public static final String **DECIMAL**

INTEGER

public static final String **INTEGER**

Constructors

RandomNumberProvider

public **RandomNumberProvider**(*PopulatorInterface* *populator*)

RandomNumberProvider

public **RandomNumberProvider**(*PopulatorInterface* *populator*, String *format*)

Methods

generate

public String **generate**()

getDecimal

public *ProviderInterface* **getDecimal**()

setMax

public *RandomNumberProvider* **setMax**(int *min*)

setMin

public *RandomNumberProvider* **setMin**(int *max*)

5.6.11 TelephoneProvider

public class **TelephoneProvider** extends *Provider*

Constructors

TelephoneProvider

public **TelephoneProvider**(*PopulatorInterface* *populator*)

TelephoneProvider

public **TelephoneProvider**(*PopulatorInterface* *populator*, String *format*)

Methods

generate

public String **generate**()

5.7 com.eddmash.form.fields

5.7.1 BaseField

public abstract class **BaseField**<V, E> implements *FieldInterface*<V, E>

Fields

form

protected *FormInterface* **form**

isEditable

protected boolean **isEditable**

Constructors

BaseField

BaseField(boolean *isEditable*)

Methods

getForm

public *FormInterface* **getForm**()

getValue

public abstract E **getValue**()

getView

public abstract V **getView**()

isEditable

public boolean **isEditable**()

setForm

public void **setForm**(*FormInterface form*)

setValue

public abstract void **setValue**(E *o*)

5.7.2 CollectionField

public class **CollectionField** extends *BaseField*<List<View>, Object> implements *CollectionFieldInterface*<List<View>, Object>

Field that manipulates multiple individual views together.

Its important to note that the specific fields don't loose there individuality and the values return will be values for each single view.

Setting will be attempted on each single view if its value is found in the map of values passed in.

Constructors

CollectionField

public **CollectionField**(String *tag*)

CollectionField

public **CollectionField**(String *name*, boolean *isEditable*)

Methods

addField

public void **addField**(String *name*, View *view*)

addField

public void **addField**(*FieldInterface field*)

addField

public void **addField**(String *name*, View *view*, boolean *editable*)

getFields

public Map<String, *FieldInterface*> **getFields**()

getName

public String **getName**()

getValue

public Map<String, Object> **getValue**()

getView

public List<View> **getView**()

setValue

public void **setValue**(Object *o*)

5.7.3 CollectionFieldInterface

public interface **CollectionFieldInterface**<T, E> extends *FieldInterface*<T, E>
Field that manipulates multiple views together.

Its important to note that the specific fields don't loose there individuality and the values return will be values for each single view.

Setting will be attempted on each single view if its value is found in the map of values passed in.

Methods

addField

void **addField**(String *name*, View *view*)
Add view to the collection.

Parameters

- **name** – identify the view uniquely
- **view** – the view instance

addField

void **addField**(String *name*, View *view*, boolean *editable*)
Add view to the collection.

Parameters

- **name** – identify the view uniquely
- **view** – the view instance
- **editable** – indicate if view allows having its values being set, true if view is editable, else false.

addField

void **addField**(*FieldInterface field*)
Add field to the collection.

Parameters

- **field** – field to be added to the collection

getFields

Map<String, *FieldInterface*> **getFields**()
The fields that make up the collection

Returns map of fields. that are the in the collection field.

getValue

E **getValue**()

getView

T **getView**()

setValue

void **setValue**(E *o*)

5.7.4 FieldInterface

public interface **FieldInterface**<T, E> extends *FormAwareInterface*
This provides a consitent way of dealing with the different views provided by android.

Methods

getForm

FormInterface **getForm**()
The form instance this field is attached to.

Returns form

getName

String **getName**()
A name that uniquely identify the view. this is use when you need to pull a specific field from the form instance.

Returns name

getValue

E **getValue**()
Returns the value of the view

Throws

- ***FormException*** –

Returns Object

getView

T **getView**()
The actual view object(s) we are operating on.

Note this may return a list of view objects in case of CollectionField

Throws

- ***FormException*** – in case it not possible to retrieve the view object

Returns a view instance

isEditable

boolean **isEditable**()
Is the view editable, this tells the form not to set values for the view and also tells the populator not to populate it.

Returns true if editable, false otherwise

setValue

void **setValue**(E *o*)
Set view value.

Parameters

- **o** –

Throws

- ***FormException*** –

5.7.5 MultiField

public class **MultiField** extends *BaseField*<List<View>, Map> implements *MultiFieldInterface*
This fields deals with multi fields but they are each treated as one.

This mean the values return are for those fields that have values, the setting also happens to those fields who data is provided.

Constructors

MultiField

public **MultiField**(String *name*, boolean *isEditable*)

MultiField

public **MultiField**(String *name*)

Methods

addView

public void **addView**(String *id*, View *view*)

getChildCount

public int **getChildCount**()

getField

public *FieldInterface* **getField**(String *id*)

getFields

public List<*FieldInterface*> **getFields**()

getName

public String **getName**()

getValue

public Map **getValue**()

getView

public List<View> **getView**()

removeView

public void **removeView**(String *id*)

setValue

public void **setValue**(Map *o*)

5.7.6 MultiFieldInterface

public interface **MultiFieldInterface**

Methods

addView

void **addView**(String *id*, View *view*)

getChildCount

int **getChildCount**()
Get the number of views the multifield contains.

getField

FieldInterface **getField**(String *id*)

getFields

List<*FieldInterface*> **getFields**()

removeView

void **removeView**(String *id*)

5.7.7 SimpleField

public class **SimpleField** extends *BaseField*<Object, Object>

Constructors

SimpleField

public **SimpleField**(String *name*, Object *value*)

SimpleField

public **SimpleField**(String *name*, Object *value*, boolean *isEditable*)

Methods

getName

public String **getName**()

getValue

public Object **getValue**()

getView

public Object **getView**()

setValue

public void **setValue**(Object *o*)

5.7.8 ViewField

public class **ViewField** extends *BaseField*<View, String>

Constructors

ViewField

public **ViewField**(String *name*, View *view*)

ViewField

public **ViewField**(String *name*, View *view*, boolean *isEditable*)

Methods

getName

public String **getName**()

getSpinnerValuePosition

public int **getSpinnerValuePosition**(Spinner *spinner*, Object *val*)

getValue

public String **getValue**()

getView

public View **getView**()

setIsEditable

public void **setIsEditable**(boolean *isEditable*)

setValue

public void **setValue**(String *val*)

5.8 com.eddmash.form.values

5.8.1 MapValue

public class **MapValue** implements *ValueInterface*<Map>
ValueInterface

Use this if you need to be able to access the map that the value and label where pulled from latter.

Constructors

MapValue

public **MapValue**(Map *item*, String *labelCol*, String *valueCol*)

Methods

fromCollection

public static List<*ValueInterface*> **fromCollection**(List<Map> *data*, String *colKey*, String *valueKey*)
Take list of maps and prepares them for use as values on a spinner.

Parameters

- **data** –

getItem

public Map **getItem**()

getLabel

public String **getLabel**()

getValue

public String **getValue**()

toString

public String **toString**()

5.8.2 SimpleValue

public class **SimpleValue** implements *ValueInterface*<String>

ValueInterface

This basically deals with simples string values.

Constructors

SimpleValue

public **SimpleValue**(String *value*, String *label*)

Methods

getItem

public String **getItem**()

getLabel

public String **getLabel**()

getValue

public String **getValue**()

toString

public String **toString**()

5.8.3 ValueInterface

public interface **ValueInterface**<T>

This class make it easy to deal with data passed into views like spinner.

e.g. if you have a user records [{“gender”:”male”, “id”:”1”},{“gender”:”Female”, “id”:”2”}] and you would like to display the ‘gender’ on the spinner and whilst making it easy to get the id assigned to the gender when its selected.

```
List genders = new ArrayList<>();
genders.add(new SimpleValue(" ", " "));
genders.add(new SimpleValue("1", "Male"));
genders.add(new SimpleValue("2", "Female"));

Spinner genderSpinner = findViewById(R.id.gender);

ArrayAdapter adapter = new ArrayAdapter<>(this,
                                          android.R.layout.simple_list_item_1,␣
↪genders);
genderSpinner.setAdapter(adapter);
```

With this setup you can retrieve the id of the selected gender as follows

```
 genderSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()
↪{
      @Override
      public void onItemClick(AdapterView> adapterView, View view, int i,  long l)
      {
              if (adapterView.getSelectedItem() instanceof ValueInterface) {
              ValueInterface val = (ValueInterface) adapterView.getSelectedItem();
              val.getItem(); // get the data structure backing this value item
              val.getLabel(); // the the label
              val.getValue(); // get the actual value
          }
      }
});
```

Parameters

- **<T>** –

Methods

getItem

T **getItem**()

This should return the actual datastructure backing the ValueInterface instance. *MapValue.getItem()* } which returns a map from which the value and label were retrieved.

getLabel

String **getLabel**()

This is displayed to the use, e.g. in the examples above Male and Female will be displated.

getValue

String **getValue**()

The actual value that we need regardless of what is displayed to the user.

5.9 com.eddmash.grids

5.9.1 DataGridView

public class **DataGridView** extends LinearLayout

DataGridView can be used to display a list or table of data records providing features like pagination.

Its takes a List of Maps that contains data and renders each row using a set of *ColumnInterface* presenting data in the form of a table.

The minimal code needed to use DataGridView is as follows:

```
dataGridView = (DataGridView) findViewById(R.id.data_view);
dataGridView.setPageSize(3);
List data = "...;
dataGridView.setData(data);
```

Its also possible to override which columns are used in the grid and customize those columns as one wishes.

Assuming in the data provided to the gridview looks like this

[{“first_name”:”jake”, “age”:”4”}, {“first_name”:”joan”, “age”:”6”},]

The ealier example will render both the firstname and age column on the grid we can tell the gridview to only render the firstname by *DataGridView.setColumns(Map)* as shown below:

```
Map cols = new HashMap();
cols.put("first_name", new Column(this,"first_name","First Name"));
dataGridView.setColumns(cols);
```

Fields

LEFT

public static final boolean **LEFT**

RIGHT

public static final boolean **RIGHT**

data

protected List<Map> **data**

Constructors

DataGridView

public **DataGridView**(Context *context*)

DataGridView

public **DataGridView**(Context *context*, AttributeSet *attrs*)

DataGridView

public **DataGridView**(Context *context*, AttributeSet *attrs*, int *defStyleAttr*)

Methods

addColumn

public void **addColumn**(*BaseColumn* *col*, boolean *position*)
Add extra columns to the dataview

Parameters

- **col** –
- **position** – true to add column at the beginning, false to add to the right

addToolbarView

public void **addToolbarView**(View *view*)

getColumns

public Map<String, *ColumnInterface*> **getColumns**()
Returns of columns to use on this grid.

Returns grid columns.

getContentLayout

public LinearLayout **getContentLayout**()

getCurrentPageString

protected String **getCurrentPageString**()

getDataListener

protected *DataListener* **getDataListener**()

getFooterLayout

protected LinearLayout **getFooterLayout**()

getNextPageBtn

protected Button **getNextPageBtn**()

isStripped

public void **isStripped**(boolean *isStripped*)

makeDataRows

protected void **makeDataRows**()

makeFooterRow

protected void **makeFooterRow**()

makeHeaderRow

protected void **makeHeaderRow**()

makeToolbarRow

protected void **makeToolbarRow**()

setColumns

public void **setColumns**(Map *attributesLabel*)

Determine how and which columns of your data will be displayed by passing them here.

This overrides the default implementation of using all the columns in you data.

this method accepts a map in the following form {"gender}

Parameters

- **attributesLabel** – a map

setData

public void **setData**(List<Map> *data*)

You can populate the datagrid using list of map data. This data will be paginated based on the page size set

Parameters

- **data** – data to display.

setHeaderColor

public void **setHeaderColor**(int *headerColor*)

setPageSize

public void **setPageSize** (int *pageSize*)

setPaginator

public void **setPaginator** (*PaginatorInterface* *paginator*, LazyResolver *lazyResolver*)

setQuery

public void **setQuery** (SQLiteDatabase *database*, String *sql*, String[] *params*)

You can populate the datagrid using a query. THe data grid will fetch data in a paginated manner base on the page size set.

```
dataGridView = (DataGridView) findViewById(R.id.content_dsp);
dataGridView.setPageSize(3);
dataGridView.setQuery(SqlHelper.getInstance(this).getReadableDatabase(),
                                "select * from coffees", new String[]{});
```

Parameters

- **database** – a SQLiteDatabase to use
- **sql** – querys to fetch data
- **params** – binding params to the query

setStripColor

public void **setStripColor** (int *stripColor*)

5.9.2 DataGridView.DataViewListener

public class **DataViewListener** implements *DataListener*

Methods

dataUpdate

public void **dataUpdate** (List<Map> *records*)

onFirstPageDataLoaded

public void **onFirstPageDataLoaded** (boolean *hasMorePages*)

onLastPageDataLoaded

public void **onLastPageDataLoaded** ()

onNextPageDataLoaded

public void **onNextPageDataLoaded**()

preDataLoad

public void **preDataLoad**(boolean *hasMorePages*)

5.10 com.eddmash.pagination

5.10.1 DataListener

public interface **DataListener**
Implementation of this class will be notified each time there is a change in the *SqlPaginator*.

Methods

dataUpdate

void **dataUpdate**(List<Map> *records*)
Use this method to update whichever data structure you using to hold the data.

This is invoked when new data is received. it asynchronously on the doBackground method of an AsyncTask.

Please note this method is never run on the main UI thread.

Parameters

- **records** – list of data for the current page.

onFirstPageDataLoaded

void **onFirstPageDataLoaded**(boolean *hasMorePages*)
Triggered when data for the first page has been loaded and is ready for use.

The data can be accessed on *DataListener.dataUpdate(List)* method.

Parameters

- **hasMorePages** – true if there are more pages to load.

onLastPageDataLoaded

void **onLastPageDataLoaded**()
Triggered when data for the last page has been loaded and is ready for use.

The data can be accessed on *DataListener.dataUpdate(List)* method.

onNextPageDataLoaded

void **onNextPageDataLoaded**()
Invoked when a new list of records has been added to the current records. this is called after fetchNextPageData();

This method should be run on the main ui thread. on AsyncTask this should be invoked on onPostExecute()

preDataLoad

void **preDataLoad**(boolean *hasMorePages*)
Invoked before the next page is loaded.

Parameters

- **hasMorePages** – true if there are more pages to load.

5.10.2 ListPaginator

public class **ListPaginator** extends *Paginator*
An implimentation of `PaginatorInterface`

Constructors

ListPaginator

public **ListPaginator**(*DataListener dataListener*)

Methods

getNextPageRecords

protected List<Map> **getNextPageRecords**(int *startPoint*, int *endPoint*)

setData

public void **setData**(List<Map> *results*)

5.10.3 Paginator

public abstract class **Paginator** implements *PaginatorInterface*

Fields

_currentRecordsCounter

protected int **_currentRecordsCounter**

_pageCount

protected int **_pageCount**

_totalRecords

protected int **_totalRecords**

currentPage

protected int **currentPage**

dataListener

protected *DataListener* **dataListener**

isLastPage

protected boolean **isLastPage**

logTag

protected String **logTag**

newPageStartPoint

public int **newPageStartPoint**

pageSize

protected int **pageSize**

populating

protected boolean **populating**
Monitor if we are currently populating.

Constructors

Paginator

public **Paginator** (*DataListener dataListener*)

Methods

fetchNextPageData

public void **fetchNextPageData**()

getCurrentPage

protected int **getCurrentPage**()

getCurrentPageString

public String **getCurrentPageString**()

getNextPageRecords

protected abstract List<Map> **getNextPageRecords**(int *startPoint*, int *endPoint*)

getPageCount

public int **getPageCount**()

getTotalRecords

public int **getTotalRecords**()

setPageSize

public void **setPageSize**(int *pageSize*)

5.10.4 Paginator.LoadDataTask

protected static class **LoadDataTask** extends AsyncTask<Integer, Void, Void>

Constructors

LoadDataTask

LoadDataTask(*Paginator paginator*)

Methods

doInBackground

protected Void **doInBackground**(Integer... *params*)

onPostExecute

protected void **onPostExecute** (Void *done*)

5.10.5 PaginatorInterface

public interface **PaginatorInterface**

Class that implement this interface help you manage paginated data can data that's split across several pages.

ListPaginator this is used to resolve data that is not alot, i.e. data that can be held inthe memory with little to no cost on the app, this is usually data held in a List.

With large data that is usally stored in the database the *SqlPaginator* comes in handy, this paginator does not hold any data within it.

It only fetches the data and provides this data to you, its upto you to know how to store or display this data in an efficient manner.

Methods

fetchNextPageData

void **fetchNextPageData** ()

This does the actual loading of data. The loading should be done asynchronously.

getPageCount

int **getPageCount** ()

The total number of pages.

getTotalRecords

int **getTotalRecords** ()

The total number of records.

setPageSize

void **setPageSize** (int *pageSize*)

The number of records to display per page.

Parameters

- **pageSize** –

5.10.6 SqlPaginator

public class **SqlPaginator** extends *Paginator*

An implimentation of *PaginatorInterface*

Constructors

SqlPaginator

public **SqlPaginator** (*DataListener dataListener*, SQLiteDatabase *database*)

Methods

getNextPageRecords

protected List<Map> **getNextPageRecords** (int *startPoint*, int *endPoint*)

query

public void **query** (String *sql*, String[] *params*)

5.11 com.eddmash.validation

5.11.1 ValidationListener

public interface **ValidationListener**
Interface definition of callbacks to be invoked when the validation state has changed.

Methods

onValidationFailed

void **onValidationFailed** (*ValidatorInterface validatorInterface*)
Invoked when validation failed

onValidationSuccess

void **onValidationSuccess** (*ValidatorInterface validatorInterface*)
Invoked when the validation passed successfully.

5.11.2 Validator

public class **Validator** implements *ValidatorInterface*

Constructors

Validator

public **Validator** ()

Validator

public **Validator** (String *tag*)

Methods

addCheck

public void **addCheck** (*CheckInterface checkInterface*)

addValidator

public void **addValidator** (*ValidatorInterface validator*)

clearErrors

public void **clearErrors** ()

disableCheck

public void **disableCheck** (*CheckInterface checkInterface*)
see

disableValidator

public void **disableValidator** (*ValidatorInterface validatorInterface*)

getErrors

public Map<String, List> **getErrors** ()

getErrorsByTag

public List **getErrorsByTag** (String *tag*)

toString

public String **toString** ()

validate

public boolean **validate** ()

validate

public void **validate**(*ValidationListener validationListener*)

5.11.3 ValidatorInterface

public interface **ValidatorInterface**
Validators should implement this interface

Validator

Methods

addCheck

void **addCheck**(*CheckInterface checkInterface*)
Adds validation checks to be enforced by a validator

Parameters

- **checkInterface** –

addValidator

void **addValidator**(*ValidatorInterface validatorInterface*)
Add another validator to validated at the time this one is being validated.

Parameters

- **validatorInterface** – the validatorInterface object

clearErrors

void **clearErrors**()
Clear all the errors from the validator.

maybe use when you have already run the validation onces and want to run the validation again using the same ValidatorInterface instance

disableCheck

void **disableCheck**(*CheckInterface checkInterface*)
disable validation check

Parameters

- **checkInterface** – the validation check to disable.

disableValidator

void **disableValidator**(*ValidatorInterface validatorInterface*)
Disable the validator from being validated any more.

Parameters

- **validatorInterface** – validatorInterface object

getErrors

Map<String, List> **getErrors**()
Returns all error that the validator found as a HashMap. with the key being tags if your passed in any when creating the validator otherwise all errors afre returned under the tag NON_SPECIFIC

the value of the HashMap consists an ArrayList of errors that relate to each tag

Returns Map

getErrorsByTag

List **getErrorsByTag**(String *tag*)
Gets a list of errors for a specific tag.

Parameters

- **tag** –

validate

boolean **validate**()
Does the actual validation.

Returns boolean true of valid

validate

void **validate**(*ValidationListener validationListener*)
Does the actual validation.

Parameters

- **validationListener** – listener that is

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

B

C

H

I

L

M

N

O

P

Q

R

S

T

V