pub.dev

flutter_form_builder 6.0.1

Published May 19, 2021 • danvickmiller.dev Null safety • Latest: 6.0.1 / Prerelease: 7.0.0-alpha.2

FLUTTER | ANDROID IOS LINUX MACOS WEB WINDOWS

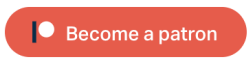
847

Metadata →

This package helps in creation of forms in Flutter by removing the boilerplate code, reusing validation, react to changes, and collect final user input.

More...

Readme Changelog Example Installing Versions Scores



Flutter FormBuilder - flutter_form_builder

This package helps in creation of data collection forms in Flutter by removing the boilerplate needed to build a form, validate fields, react to changes, and collect final user input.

Simple Usage

To use this plugin, add `flutter_form_builder` as a dependency in your pubspec.yaml file.

Flutter Version Guide

- Flutter 1.20 => `v4.*`
- Flutter 2.* with no `null-safety` => `v5.*`

- Flutter 2.* `null-safety => v6.*` - some dependencies (and therefore fields)* were removed to achieve null safety

New Video Tutorial

 Youtube Video Tutorial
 Check out the video tutorial from SyntacOps on Youtube

Example

```dart
final _formKey = GlobalKey<FormBuilderState>();

@override
Widget build(BuildContext context) {
  return Column(
    children: <Widget>[
      FormBuilder(
        key: _formKey,
        autovalidate: true,
        child: Column(
          children: <Widget>[
            FormBuilderFilterChip(
              name: 'filter_chip',
              decoration: InputDecoration(
                labelText: 'Select many options',
              ),
              options: [
                FormBuilderFieldOption(
                    value: 'Test', child: Text('Test')),
                FormBuilderFieldOption(
                    value: 'Test 1', child: Text('Test 1')),
                FormBuilderFieldOption(
                    value: 'Test 2', child: Text('Test 2')),
                FormBuilderFieldOption(
                    value: 'Test 3', child: Text('Test 3')),
                FormBuilderFieldOption(
                    value: 'Test 4', child: Text('Test 4')),
              ],
            ),
            FormBuilderChoiceChip(
              name: 'choice_chip',
              decoration: InputDecoration(
                labelText: 'Select an option',
              ),
              options: [
                FormBuilderFieldOption(
                    value: 'Test', child: Text('Test')),
                FormBuilderFieldOption(
                    value: 'Test 1', child: Text('Test 1')),
                FormBuilderFieldOption(
                    value: 'Test 2', child: Text('Test 2')),
                FormBuilderFieldOption(
                    value: 'Test 3', child: Text('Test 3')),
                FormBuilderFieldOption(
                    value: 'Test 4', child: Text('Test 4')),
              ],
            ),
            FormBuilderDateTimePicker(
              name: 'date',
              // onChanged: _onChanged,
              inputType: InputType.time,
```

```
      inputType: InputType.time,
      decoration: InputDecoration(
        labelText: 'Appointment Time',
      ),
      initialTime: TimeOfDay(hour: 8, minute: 0),
      // initialValue: DateTime.now(),
      // enabled: true,
    ),
    FormBuilderDateRangePicker(
      name: 'date_range',
      firstDate: DateTime(1970),
      lastDate: DateTime(2030),
      format: DateFormat('yyyy-MM-dd'),
      onChanged: _onChanged,
      decoration: InputDecoration(
        labelText: 'Date Range',
        helperText: 'Helper text',
        hintText: 'Hint text',
      ),
    ),
    FormBuilderSlider(
      name: 'slider',
      validator: FormBuilderValidators.compose([
        FormBuilderValidators.min(context, 6),
      ]),
      onChanged: _onChanged,
      min: 0.0,
      max: 10.0,
      initialValue: 7.0,
      divisions: 20,
      activeColor: Colors.red,
      inactiveColor: Colors.pink[100],
      decoration: InputDecoration(
        labelText: 'Number of things',
      ),
    ),
    FormBuilderCheckbox(
      name: 'accept_terms',
      initialValue: false,
      onChanged: _onChanged,
      title: RichText(
        text: TextSpan(
          children: [
            TextSpan(
              text: 'I have read and agree to the ',
              style: TextStyle(color: Colors.black),
            ),
            TextSpan(
              text: 'Terms and Conditions',
              style: TextStyle(color: Colors.blue),
            ),
          ],
        ),
      ),
      validator: FormBuilderValidators.equal(
        context,
        true,
        errorText:
            'You must accept terms and conditions to continue',
      ),
    ),
    FormBuilderTextField(
      name: 'age',
      decoration: InputDecoration(
        labelText:
            'This value is passed along to the [Text.maxLines] attribute
```

```
            ),
            onChanged: _onChanged,
            // valueTransformer: (text) => num.tryParse(text),
            validator: FormBuilderValidators.compose([
              FormBuilderValidators.required(context),
              FormBuilderValidators.numeric(context),
              FormBuilderValidators.max(context, 70),
            ]),
            keyboardType: TextInputType.number,
          ),
          FormBuilderDropdown(
            name: 'gender',
            decoration: InputDecoration(
              labelText: 'Gender',
            ),
            // initialValue: 'Male',
            allowClear: true,
            hint: Text('Select Gender'),
            validator: FormBuilderValidators.compose(
                [FormBuilderValidators.required(context)]),
            items: genderOptions
                .map((gender) => DropdownMenuItem(
                      value: gender,
                      child: Text('$gender'),
                    ))
                .toList(),
          ),
        ],
      ),
    ),
    Row(
      children: <Widget>[
        Expanded(
          child: MaterialButton(
            color: Theme.of(context).accentColor,
            child: Text(
              "Submit",
              style: TextStyle(color: Colors.white),
            ),
            onPressed: () {
              _formKey.currentState.save();
              if (_formKey.currentState.validate()) {
                print(_formKey.currentState.value);
              } else {
                print("validation failed");
              }
            },
          ),
        ),
        SizedBox(width: 20),
        Expanded(
          child: MaterialButton(
            color: Theme.of(context).accentColor,
            child: Text(
              "Reset",
              style: TextStyle(color: Colors.white),
            ),
            onPressed: () {
              _formKey.currentState.reset();
            },
          ),
        ),
      ],
    )
  ],
```

```
    );
  }
```

l10n

Just add the `FormBuilderLocalizations.delegate` in the list of your app's `localizationsDelegates`

```
return MaterialApp(
    supportedLocales: [
      Locale('en'),
      Locale('it'),
      Locale('fr'),
      Locale('es'),
    ],
    localizationsDelegates: [
      FormBuilderLocalizations.delegate,
      GlobalMaterialLocalizations.delegate,
      GlobalWidgetsLocalizations.delegate,
    ],
```

Input widgets

The currently supported fields include:

- `FormBuilderCheckbox` - Single Checkbox field
- `FormBuilderCheckboxGroup` - List of Checkboxes for multiple selection
- `FormBuilderChoiceChip` - Creates a chip that acts like a radio button.
- `FormBuilderDateRangePicker` - For selection of a range of dates
- `FormBuilderDateTimePicker` - For `Date`, `Time` and `DateTime` input
- `FormBuilderDropdown` - Used to select one value from a list as a Dropdown
- `FormBuilderFilterChip` - Creates a chip that acts like a checkbox.
- `FormBuilderRadioGroup` - Used to select one value from a list of Radio Widgets
- `FormBuilderRangeSlider` - Used to select a range from a range of values
- `FormBuilderSegmentedControl` - For selection of a value from the `CupertinoSegmentedControl` as an input
- `FormBuilderSlider` - For selection of a numerical value on a slider
- `FormBuilderSwitch` - On/Off switch field
- `FormBuilderTextField` - A Material Design text field input.

In order to create an input field in the form, along with the label, and any applicable validation, there are several attributes that are supported by all types of inputs namely:

Attribute	Type	Default	Required	Description

Attribute	Type	Default	Required	Description
name	String		Yes	This will form in the form val
initialValue	T	null	No	The initial valu the input field
enabled	bool	true	No	Determines wh the field widge accept user in
decoration	InputDecoration	InputDecoration()	No	Defines the bo labels, icons, a styles used to decorate the fi
validator	FormFieldValidator<T>	null	No	A FormFieldVal that will check validity of valu FormField
onChanged	ValueChanged<T>	null	No	This event fun will fire immed the the field va changes
valueTransformer	ValueTransformer<T>	null	No	Function that transforms fiel before saving value. e.g. tran TextField value numeric field f String to nu

Attribute	Type	Default	Required	Description
The rest of the attributes will be determined by the type of Widget being used.				

Additional input fields

To make this package compartible with as many platforms as Flutter supports, we separated some input fields into their own packages because they depend on platform-specific plugins. Here's are the links to some of the packages that could be used with `FormBuilder`

- FormBuilderFilePicker - Picks image(s) from user device storage.
- FormBuilderImagePicker - Picks image(s) from Gallery or Camera.
- FormBuilderLocationField - Geographic location input.
- FormBuilderPhoneField - International phone number input.

Building your own custom field

To build your own field within a `FormBuilder`, we use `FormBuilderField` which will require that you define your own field.

```
var options = ["Option 1", "Option 2", "Option 3"];
```

```
FormBuilderField(
  name: "name",
  validator: FormBuilderValidators.compose([
    FormBuilderValidators.required(context),
  ]),
  builder: (FormFieldState<dynamic> field) {
    return InputDecorator(
      decoration: InputDecoration(
        labelText: "Select option",
        contentPadding:
            EdgeInsets.only(top: 10.0, bottom: 0.0),
        border: InputBorder.none,
        errorText: field.errorText,
      ),
      child: Container(
        height: 200,
        child: CupertinoPicker(
          itemExtent: 30,
          children: options.map((c) => Text(c)).toList(),
          onSelectedItemChanged: (index) {
            field.didChange(options[index]);
          },
        ),
      ),
    );
  },
),
```

Programmatically changing field value

You can either change the value of one field at a time like so:

```
_formKey.currentState.fields['color_picker'].didChange(Colors.black);
```

Or multiple fields value like so:

```
_formKey.currentState.patchValue({
  'age': '50',
  'slider': 6.7,
  'filter_chip': ['Test 1'],
  'choice_chip': 'Test 2',
  'rate': 4,
  'chips_test': [
    Contact('Andrew', 'stock@man.com', 'https://d2gg9evh47fn9z.cloudfront.net/8(
  ],
});
```

Validation

The `validator` attribute in fields take in a `FormFieldValidator` which checks the validity of the field. A `FormFieldValidator` returns `null` if validation is successful and a `String` for the `errorText` if validation fails.

Built-in Validators

This package comes with several most common `FormFieldValidator` s such as required, numeric, mail, URL, min, max, minLength, maxLength, IP, credit card etc. with default `errorText` .

Available built-in validators include:

- `FormBuilderValidators.creditCard()` - requires the field's value to be a valid credit card number.
- `FormBuilderValidators.date()` - requires the field's value to be a valid date string.
- `FormBuilderValidators.email()` - requires the field's value to be a valid email address.
- `FormBuilderValidators.equal()` - requires the field's value be equal to provided object.
- `FormBuilderValidators.integer()` - requires the field's value to be an integer.
- `FormBuilderValidators.ip()` - requires the field's value to be a valid IP address.
- `FormBuilderValidators.match()` - requires the field's value to match the provided regex pattern.
- `FormBuilderValidators.max()` - requires the field's value to be less than or equal to the provided number.
- `FormBuilderValidators.maxLength()` - requires the length of the field's value to be less than or equal to the provided maximum length.
- `FormBuilderValidators.min()` - requires the field's value to be greater than or equal to the provided number.
- `FormBuilderValidators.minLength()` - requires the length of the field's value to be greater than or equal to the provided minimum length.
- `FormBuilderValidators.numeric()` - requires the field's value to be a valid number.
- `FormBuilderValidators.required()` - requires the field have a non-empty value.
- `FormBuilderValidators.url()` - requires the field's value to be a valid url.

Using multiple validators

`FormBuilderValidators` class comes with a very useful static function named `compose()` which takes any number of `FormFieldValidator` functions. On validation each validator is run and if any returns a non-null value (i.e. a String), validation fails and the `errorText` for the field is set as the returned string.

Validation example:

```
FormBuilderTextField(
  name: 'age',
  decoration: InputDecoration(labelText: 'Age'),
  validator: FormBuilderValidators.compose([
    FormBuilderValidators.numeric(context, errorText: 'La edad debe ser numérica
    FormBuilderValidators.max(context, 70),
    (val){
      if(val < 0)
        return 'We cannot have a negative age';
      return null;
    }
  ]),
),
```

Programmatically inducing an error

Declare a variable to hold your error:

```
String _emailError;
```

Use the variable as the `errorText` within `InputDecoration`

```
FormBuilderTextField(
  name: 'email',
  decoration: InputDecoration(
    labelText: 'Email',
    errorText: _emailError,
  ),
  validator: FormBuilderValidators.compose([
      FormBuilderValidators.required(context),
      FormBuilderValidators.email(context),
  ]),
),
```

Set the error text

```
RaisedButton(
  child: Text('Submit'),
  onPressed: () async {
    setState(() => _emailError = null);
    if(checkIfEmailExists()){
      setState(() => _emailError = 'Email already taken.');
    }
  },
),
```

Conditional validation

You can also validate a field based on the value of another field

```
FormBuilderRadioGroup(
  decoration: InputDecoration(labelText: 'My best language'),
  name: 'my_language',
  validator: FormBuilderValidators.required(context),
  options: [
    'Dart',
    'Kotlin',
    'Java',
    'Swift',
    'Objective-C',
    'Other'
  ]
    .map((lang) => FormBuilderFieldOption(value: lang))
    .toList(growable: false),
  ),
  FormBuilderTextField(
    name: 'specify',
    decoration:
        InputDecoration(labelText: 'If Other, please specify'),
    validator: (val) {
      if (_formKey.currentState.fields['my_language']?.value ==
              'Other' &&
          (val == null || val.isEmpty)) {
        return 'Kindly specify your language';
      }
      return null;
    },
  ),
```

SUPPORT

Issues and PRs

Any kind of support in the form of reporting bugs, answering questions or PRs is always appreciated.

We especially welcome efforts to internationalize/localize the package by translating the default validation `errorText` strings.

Localizing messages

1. With the app's root directory as the current directory, generate `l10n/intl_messages.arb` from `lib/localization/form_builder_localizations.dart`:

   ```
   flutter pub pub run intl_translation:extract_to_arb --output-dir=lib/l10n
   lib/localization/form_builder_localizations.dart
   ```

2. The `intl_messages.arb` file is a JSON format map with one entry for each `Intl.message()` function defined in `lib/localization/form_builder_localizations.dart`. This file serves as a template for

the different translations (for example `intl_en.arb` and `intl_es.arb` are English and Spanish translations respectively). You are therefore you are required to copy the `intl_messages.arb` and put the content in a new file with the name of your locale with a name with format `intl_<locale>.arb` (e.g. `intl_fr.arb` for French Translations).

3. Translate the messages in the new file to the required language.

4. With the app's root directory as the current directory, generate `intl_messages_<locale>.dart` for your `intl_<locale>.arb` file and update `intl_messages_all.dart`, which imports all of the messages files:

```
flutter pub run intl_translation:generate_from_arb --output-dir=lib/l10n --no-use-deferred-loading lib/localization/form_builder_localizations.dart lib/l10n/intl_<en>.arb lib/l10n/intl_messages.arb
```

e.g. To generate for French run: `flutter pub run intl_translation:generate_from_arb --output-dir=lib/l10n --no-use-deferred-loading lib/localization/form_builder_localizations.dart lib/l10n/intl_fr.arb lib/l10n/intl_messages.arb`

- Alternatively you could run the following command to generate Dart translation files for all the `intl_<locale>.arb` files in the `l10n/` directory:

```
flutter pub pub run intl_translation:generate_from_arb --output-dir=lib/l10n --no-use-deferred-loading lib/localization/form_builder_localizations.dart lib/l10n/intl_*.arb
```

5. Include your new language to `FormBuilderLocalization`'s supported languages. Go to `lib/localization/form_builder_localizations.dart` and include the language like so:

```
@override
  bool isSupported(Locale locale) {
    return ['en', 'es', 'fr'].contains(locale.languageCode);
  }
```

6. Submit your PR and be of help to millions of people all over the world!

Coffee :-)

If this package was helpful to you in delivering your project or you just wanna to support this package, a cup of coffee would be highly appreciated ;-)

CREDITS

Contributors

Made with contributors-img.