Flutter – Input Types and Controller

TextField

- A TextField or TextBox is an **input element** which holds the alphanumeric data, such as name, password, address, etc.
- The following code explains a **demo example of TextFiled widget** in Flutter:

```
TextField (
  decoration: InputDecoration(
    border: InputBorder.none,
    labelText: 'Enter Name',
    hintText: 'Enter Your Name'
  ),
);
```

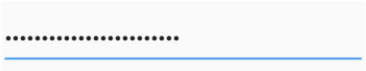
TextField – (contd…)

- Some of the most common attributes used with the TextField widget are as follows:
  - **decoration:** It is used to show the decoration around TextField.
  - **border:** It is used to create a default rounded rectangle border around TextField.
  - **labelText:** It is used to show the label text on the selection of TextField.
  - **hintText:** It is used to show the hint text inside TextField.
  - **icon:** It is used to add icons directly to the TextField.

1. TextField - obscureField

- obscureField : True
  - This property allows a widget to store data and hide it from user like password

```
child: TextField(
    obscureText: true,
    decoration: InputDecoration(
      border: OutlineInputBorder(),
      labelText: 'Password',
      hintText: 'Enter Password',
    ),
  ),
```



2. KeyBoard Types

- A TextField allows you to customise the **type of keyboard** that shows up when the TextField is brought into focus.

```
TextField(
  keyboardType: TextInputType.number,
),
```

- The types are:
  - **TextInputType.text** (Normal complete keyboard)
  - **TextInputType.number** (A numerical keyboard)
  - **TextInputType.emailAddress** (Normal keyboard with an “@”)
  - **TextInputType.datetime** (Numerical keyboard with a “/” and “:”)
  - **TextInputType.numberWithOptions** (Numerical keyboard with options to enabled signed and decimal mode)
  - **TextInputType.multiline** (Optimises for multi-line information)

3. TextInputAction

- Changing textInputAction of the TextField lets you change **the action button of the keyboard** itself.

```
TextField(
  textInputAction: TextInputAction.continueAction,
),
```

OR

```
TextField(
  textInputAction: TextInputAction.send,
),
```

4. Text Capitalization

- The TextField provides a few options on how to capitalise letters in the input from the user

```
TextField(
  textCapitalization: TextCapitalization.sentences,
),
```

- The types are:
  - **TextCapitalization.sentences**

    Hello world. Demo text.

  - **TextCapitalization.characters**

    THE NEW VERSION

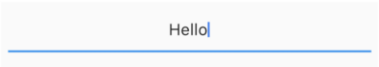
  - **TextCapitalization.words**

    Demo Text

5. Text Style, Alignment and Cursor Options

- Text alignment inside the TextField
  - Use the textAlign property to adjust where cursor is inside the TextField

```
TextField(
  textAlign: TextAlign.center,
),
```

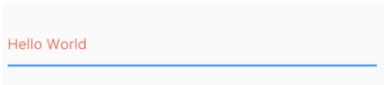


- This has the usual alignment properties: **start, end, left, right, center, justify**.

6. Styling the text inside the TextField

- We use the **style** property to change how the text inside the TextField looks.

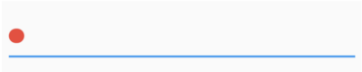
```
TextField(
  style: TextStyle(color: Colors.red, fontWeight: FontWeight.w300),
),
```



7. Changing the cursor in the TextField

- The cursor is customisable directly from the TextField widget
- You are allowed to change the cursor color, width and radius of the corners. For example, here I make a circular red cursor for no apparent reason

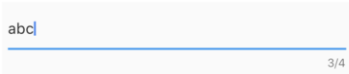
```
TextField(
  cursorColor: Colors.red,
  cursorRadius: Radius.circular(16.0),
  cursorWidth: 16.0,
),
```



8. Controlling the Size and Maximum Length in a TextField

- TextFields can control the maximum number of characters written inside it, the maximum number of lines and expand as text is typed.
- Controlling max characters

```
TextField(
  maxLength: 4,
),
```



9. Making an expandable TextField

- Sometimes, we need a TextField that expands when one line is finished. In Flutter it is slightly odd (yet easy) to do. To do this, **we set maxLines to null**, which is 1 by default.

Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor

```
TextField(
  maxLines: 3,
)
```

Note: Setting the maxLines to a direct value will expand it to that number of lines by default.

10. Decorating the TextField

- Till now we focused on the features Flutter offers for input.
- For decorating the TextField, we use the **decoration** property which takes an InputDecoration
- Use the **hint** and label properties to give information to the user
- Hint : Demo Text
- Label : Demo Text
- You can add icons using “icon”, “prefixIcon” and “suffixIcon”

```
TextField(
  decoration: InputDecoration(
    icon: Icon(Icons.print)
  ),
),
```

11. Retrieving information from a TextField

1. The easiest way to do this is to use the **onChanged** method and store the current value in a simple variable. Here is the sample code for it:

```
String value = "";

TextField(
  onChanged: (text) {
    value = text;
  },
)
```

11. Retrieving information from a TextField (contd…)

2. The second way to do this is to use a **TextEditingController**. The controller is attached to the TextField and lets us listen and control the text of the TextField as well

```
TextEditingController controller = TextEditingController();

TextField(
  controller: controller,
)
```

- And we can listen to changes using

```
controller.addListener(() {
  // Do something here
});
```

Other callbacks from the TextField

- The TextField widget also provides other callbacks such as
  - onEditingCompleted
  - onSubmitted

```
onEditingComplete: () {},

onSubmitted: (value) {},
```

Flutter : State Management

Flutter – State Management

- Managing state in an application is one of the most important and necessary process in the
- Let us consider a simple shopping cart application.
  - User will login using their credentials into the application.
  - Once user is logged in, the application should persist the logged in user detail in all the screen.
  - Again, when the user selects a product and saved into a cart, the cart information should persist between the pages until the user checked out the cart.
  - User and their cart information at any instance is called the state of the application at that instance. life cycle of an application.

Flutter – State Management

- A state management can be divided into two categories based on the duration the particular state lasts in an application.
  - Ephemeral – Last for a few seconds like the current state of an animation or a single page like current rating of a product. *Flutter* supports its through StatefulWidget.
  - app state – Last for entire application like logged in user details, cart information, etc., *Flutter* supports its through scoped_mode

Ephemeral State Management

- Since Flutter application is composed of widgets, the state management is also done by widgets
- The entry point of the state management is Statefulwidget.

```
class RatingBox extends StatefulWidget {
}
```

- Create a state for RatingBox, _RatingBoxState by inheriting State

```
class _RatingBoxState extends State<RatingBox> {
 }
```

- Override the createState of StatefulWidget method to create the state, _RatingBoxState

```
class RatingBox extends StatefulWidget {
  @override
  _RatingBoxState createState() => _RatingBoxState();
}
```

Ephemeral State Management : Example

- This state is also known as UI State or local state. It is a type of state which is related to the **specific widget,** or you can say that it is a state that contains in a single widget. In this kind of state, you do not need to use state management techniques. The common example of this state is **Text Field**.

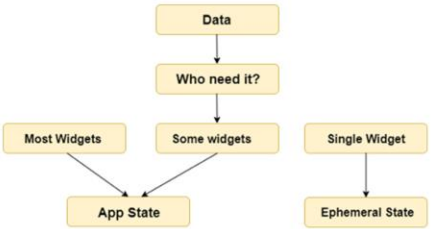
```
class MyHomepage extends StatefulWidget {
 @override
 MyHomepageState createState() => MyHomepageState();
}

class MyHomepageState extends State<MyHomepage> {
 String _name = "Peter";

 @override
 Widget build(BuildContext context) {
  return RaisedButton(
    child: Text(_name),
    onPressed: () {
      setState(() {
        _name = _name == "Peter" ? "John" : "Peter";
      });
    },
  );
 }
}
```

App State – State Management

- It is a type of state that we want to **share** across various parts of our app and want to keep between user sessions.
- Some of the examples of this state are User preferences, Login info, notifications in a social networking app, the shopping cart in an e-commerce app, read/unread state of articles in a news app, etc



App State – State Management (contd…)

- The simplest example of app state management can be learned by using the **provider package.**
- The state management with the provider is easy to understand and requires less coding.
- A provider is a **third-party** library. Here, we need to understand three main concepts to use this library.
  - ChangeNotifier
  - ChangeNotifierProvider
  - Consumer

ChangeNotifier

- **ChangeNotifier** is a simple class, which provides change notification to its listeners.
- It is easy to understand, implement, and optimized for a small number of listeners.
- It is used for the listener to observe a model for changes. In this, we only use the **notifyListener()** method to inform the listeners.

```
import 'package:flutter/material.dart';

class Counter with ChangeNotifier {
 int _counter;

 Counter(this._counter);

 getCounter() => _counter;
 setCounter(int counter) => _counter = counter;

 void increment() {
  _counter++;
  notifyListeners();
 }

 void decrement() {
  _counter--;
  notifyListeners();
 }
}
```