



Expert  
EXPERT

# Flutter<sup>TM</sup>

Développez vos applications  
mobiles multiplateformes  
avec **Dart**

En téléchargement

 code des exemples et  
démonstrations du livre

 + QUIZ

Version en ligne  
**OFFERTE !**  
pendant 1 an

Julien TRILLARD



Les éléments à télécharger sont disponibles à l'adresse suivante :  
**<http://www.editions-eni.fr>**  
Saisissez la référence ENI de l'ouvrage **EIFLUT** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

## Avant-propos

- 1. Pourquoi cet ouvrage ? ..... 11
  - 1.1 Comment faire son choix ? ..... 12
  - 1.2 Et dans l'univers du développement mobile ? ..... 13
- 2. À qui s'adresse cet ouvrage ? ..... 13
- 3. Structure de l'ouvrage ..... 14

## Chapitre 1 Introduction

- 1. État des lieux du développement mobile ..... 15
  - 1.1 Un marché en progression ..... 15
  - 1.2 Application Android ..... 16
  - 1.3 iOS ..... 16
  - 1.4 Le multiplateforme ..... 17
- 2. Flutter ..... 18
  - 2.1 Présentation ..... 18
  - 2.2 Historique et versions ..... 19
  - 2.3 Dart ..... 20
  - 2.4 Développement cross-platform ..... 21
  - 2.5 Le hot reload ..... 21
  - 2.6 Organisation par widgets ..... 22

**Chapitre 2****Installation de l'environnement**

1. Introduction . . . . .	25
2. Installation de Flutter sur Windows . . . . .	26
2.1 Compatibilité . . . . .	26
2.2 Téléchargement du SDK Flutter . . . . .	26
2.3 Installation du SDK . . . . .	27
3. Installation de Flutter sur macOS . . . . .	31
3.1 Compatibilité . . . . .	31
3.2 Téléchargement du SDK Flutter . . . . .	32
3.3 Installation du SDK . . . . .	32
3.3.1 Mojave (et les versions précédentes) . . . . .	35
3.3.2 Catalina . . . . .	37
3.4 Installation de cocoapods . . . . .	42
3.5 Installation de Xcode . . . . .	42
4. Installation de Flutter sur Linux . . . . .	44
4.1 Compatibilité . . . . .	44
4.2 Téléchargement du SDK Flutter . . . . .	45
4.3 Installation du SDK . . . . .	45
5. Mise à jour du SDK . . . . .	48
5.1 Connaître la version de Flutter installée . . . . .	48
5.2 Être au courant des nouveautés et des mises à jour . . . . .	49
5.3 Faire la mise à jour du SDK . . . . .	50
6. Installation d'Android Studio . . . . .	50
6.1 Introduction . . . . .	50
6.2 Pourquoi choisir Android Studio ? . . . . .	50
6.3 Installation d'Android Studio . . . . .	51
6.4 Installation des plug-ins Flutter et Dart . . . . .	53
7. Flutter Doctor . . . . .	55
7.1 Introduction . . . . .	55
7.2 Analyse de Flutter Doctor . . . . .	55

- 7.3 Corrections et validations ..... 56
  - 7.3.1 Licence Android ..... 56
  - 7.3.2 Plug-ins et autres IDE (Visual Studio Code ou IntelliJ) . . 57
  - 7.3.3 Émulateurs ..... 57

**Chapitre 3**  
**Ma première application**

- 1. Introduction ..... 59
- 2. Création d'un nouveau projet Flutter ..... 60
  - 2.1 Choix du type de projet ..... 61
  - 2.2 Configuration de l'application. .... 62
- 3. Le contenu de l'application ..... 65
  - 3.1 L'architecture de l'application ..... 65
  - 3.2 Le code de l'application ..... 69
- 4. Les émulateurs ..... 74
  - 4.1 Introduction ..... 74
  - 4.2 Paramétrage de l'émulateur Android. .... 75
  - 4.3 Lancement de la première application. .... 83
- 5. Premiers widgets. .... 86

**Chapitre 4**  
**Dart, les fondamentaux**

- 1. Introduction ..... 97
- 2. Syntaxe de base. .... 99
  - 2.1 Variables ..... 99
    - 2.1.1 Les nombres ..... 99
    - 2.1.2 Les chaînes de caractères ..... 100
    - 2.1.3 Les booléens ..... 102
    - 2.1.4 Types dynamiques ..... 102
  - 2.2 Constantes ..... 104

2.3	Collections	105
2.3.1	Listes	105
2.3.2	Maps	107
2.4	Alternatives	108
2.4.1	If	108
2.4.2	Switch	109
2.4.3	Opérateur ternaire	109
2.4.4	Opérateur de nullité	109
2.5	Boucles	110
2.5.1	For	110
2.5.2	Foreach	111
2.5.3	While	111
2.5.4	Do While	112

## Chapitre 5 Dart, notions avancées

1.	Introduction	113
2.	Fonctions	114
2.1	Introduction	114
2.2	Fonction simple	114
2.3	Fonctions avec retour	115
2.4	Fonctions avec paramètres	116
2.4.1	Forme classique	116
2.4.2	Paramètres optionnels	117
2.5	Fonctions anonymes	118
2.6	Fonctions fléchées ou fonctions lambda	119
3.	Programmation orientée objet (POO)	120
3.1	Introduction	120
3.2	Classe	120
3.3	Attributs	121
3.4	Constructeur	122
3.5	Instanciation	124

- 3.6 Accesseurs et mutateurs (getters et setters) . . . . . 126
- 3.7 Association . . . . . 129
- 3.8 Héritage. . . . . 133
- 3.9 Surchage de méthodes. . . . . 136
- 3.10 Interface . . . . . 138
- 3.11 Mixin. . . . . 140
- 3.12 Extension. . . . . 142
- 3.13 Static . . . . . 143
- 4. Généricité . . . . . 145
- 5. Exception. . . . . 150
  - 5.1 Exceptions systèmes. . . . . 151
  - 5.2 Exceptions personnalisées . . . . . 155
  - 5.3 Exceptions multiples . . . . . 158
- 6. Traitement asynchrone . . . . . 159

**Chapitre 6**  
**Les widgets de base**

- 1. Introduction . . . . . 161
- 2. Material Component . . . . . 162
  - 2.1 Scaffold . . . . . 162
    - 2.1.1 AppBar. . . . . 162
  - 2.2 floatingActionButton. . . . . 166
  - 2.3 FloatingActionButtonLocation . . . . . 168
  - 2.4 bottomNavigationBar . . . . . 170
    - 2.4.1 BottomAppBar . . . . . 170
    - 2.4.2 BottomNavigationBar . . . . . 172
  - 2.5 Drawer. . . . . 175
- 3. Cupertino (Style iOS). . . . . 178
  - 3.1 Scaffold de base. . . . . 179
  - 3.2 Scaffold avec barre de navigation . . . . . 180
  - 3.3 CupertinoNavigationBar . . . . . 183

4.	Détection de la plateforme. . . . .	185
5.	Boutons . . . . .	193
5.1	FloatingActionButton . . . . .	193
5.2	RaisedButton . . . . .	193
5.3	FlatButton. . . . .	197
5.4	IconButton . . . . .	199
5.5	OutlineButton . . . . .	204
5.6	CupertinoButton . . . . .	206
6.	Médias . . . . .	209
6.1	Icônes. . . . .	209
6.1.1	Icons (Material Design) . . . . .	210
6.1.2	CupertinoIcons (Style iOS) . . . . .	212
6.1.3	FontAwesome. . . . .	214
6.2	Images . . . . .	218
6.2.1	Images du projet . . . . .	218
6.2.2	Images du Web . . . . .	221
6.3	Texte . . . . .	223
6.3.1	Text . . . . .	223
6.3.2	Mise en forme du texte . . . . .	223
6.3.3	Choix de la police . . . . .	227
6.3.4	Google Fonts . . . . .	229
6.4	Layouts (mise en page). . . . .	231
6.4.1	Center . . . . .	231
6.4.2	Column . . . . .	231
6.4.3	Row . . . . .	236
6.4.4	Expanded. . . . .	237
6.4.5	Container . . . . .	239
6.4.6	Padding . . . . .	247
6.4.7	Divider. . . . .	249
6.4.8	Stepper . . . . .	250

**Chapitre 7**  
**Les états**

- 1. Introduction ..... 255
- 2. StatelessWidget ..... 256
- 3. StatefulWidget ..... 258
  - 3.1 Cycle de vie ..... 262
- 4. Couplage interface et logique ..... 264
  - 4.1 Provider ..... 265
  - 4.2 BloC ..... 272

**Chapitre 8**  
**Pop-up et navigation**

- 1. Introduction ..... 279
- 2. SnackBar ..... 280
- 3. AlertDialog et CupertinoAlertDialog ..... 285
  - 3.1 AlertDialog ..... 286
  - 3.2 CupertinoAlertDialog ..... 289
- 4. SimpleDialog ..... 290
  - 4.1 SimpleDialog ..... 291
  - 4.2 CupertinoDialog ..... 300
- 5. Feuilles de choix ..... 301
  - 5.1 BottomSheet ..... 301
  - 5.2 CupertinoActionSheet ..... 306
- 6. Routes ..... 310
  - 6.1 Navigator.push ..... 311
  - 6.2 Navigator.pop ..... 313
  - 6.3 Navigator.pushNamed ..... 314

## Chapitre 9

### Formulaires

1. Introduction . . . . .	321
2. Les champs de saisie de texte . . . . .	321
2.1 TextField . . . . .	322
2.2 CupertinoTextField . . . . .	330
2.3 Types de données . . . . .	333
2.3.1 Mot de passe . . . . .	333
2.3.2 Autres données . . . . .	333
2.4 TextEditingController . . . . .	334
3. Dates et heures . . . . .	337
3.1 Material Design . . . . .	337
3.1.1 Date . . . . .	337
3.1.2 Heure . . . . .	344
3.2 Cupertino . . . . .	347
3.2.1 Date . . . . .	347
3.2.2 Heure . . . . .	351
4. Switch . . . . .	355
4.1 Material Design . . . . .	355
4.2 Cupertino . . . . .	358
5. Radio . . . . .	359
6. Checkbox . . . . .	362
7. Slider . . . . .	366
7.1 Material Design . . . . .	366
7.2 Cupertino . . . . .	368

**Chapitre 10**  
**Listes et grilles**

- 1. Introduction ..... 371
- 2. ListView ..... 372
  - 2.1 ListView.builder ..... 372
  - 2.2 Dismissible ..... 375
  - 2.3 ListTile ..... 377
  - 2.4 ListView.separated ..... 381
  - 2.5 Personnalisation ..... 383
    - 2.5.1 Card ..... 383
    - 2.5.2 Inkwell ..... 385
- 3. GridView ..... 391
  - 3.1 GridView.builder ..... 391
  - 3.2 GridTileBar ..... 392
  - 3.3 Personnalisation ..... 393

**Chapitre 11**  
**API**

- 1. Introduction ..... 397
- 2. Appareil photo ..... 398
  - 2.1 Installation du package ..... 398
  - 2.2 Mise à jour des permissions ..... 398
  - 2.3 Le code du programme ..... 399
  - 2.4 Rendu ..... 404
  - 2.5 Vidéo ..... 404
- 3. GPS ..... 405
  - 3.1 Installation du package ..... 405
  - 3.2 Mises à jour des permissions ..... 406
  - 3.3 Le code du programme ..... 406
  - 3.4 Rendu ..... 409
- 4. Autres API ..... 410

**Chapitre 12****Données**

1. Introduction . . . . .	411
2. Shared Preferences . . . . .	412
2.1 Introduction . . . . .	412
2.2 Récupération des dépendances . . . . .	412
2.3 Création de l'application . . . . .	412
2.3.1 Classe Personne. . . . .	413
2.3.2 main.dart. . . . .	413
2.4 Rendu de l'application . . . . .	422
3. SQLite . . . . .	425
3.1 Introduction . . . . .	425
3.2 Récupération des dépendances . . . . .	426
3.3 Création de l'application . . . . .	427
3.3.1 Classe Personne. . . . .	427
3.3.2 Classe PersonneProvider. . . . .	428
3.3.3 main.dart. . . . .	435
3.4 Rendu de l'application . . . . .	444
4. JSON . . . . .	446
4.1 Définition . . . . .	446
4.2 Récupération des dépendances . . . . .	448
4.3 Création de l'application . . . . .	448
4.3.1 API. . . . .	448
4.3.2 L'application . . . . .	450
4.4 Rendu de l'application . . . . .	456
Index . . . . .	459

## Chapitre 4

# Dart, les fondamentaux

### 1. Introduction

Dart est un langage orienté objet moderne apparu en 2010 à l'initiative de deux développeurs de Google : Lars Bak et Kasper Lund.

À l'origine, ce projet avait comme objectif de remplacer JavaScript à moyen ou long terme, en raison des difficultés à faire évoluer ce dernier. Google a donc conçu Dart à partir de cette idée et la version 1.0 est sortie le 10 octobre 2011.

Le langage a même été adopté par ECMA International en 2014. Pour rappel, il s'agit d'une organisation européenne de standardisation en informatique. Parmi ses responsabilités, elle adopte des standards notamment pour les langages de programmation (y compris de scripts).

Malgré tous les efforts de Google, Dart n'a jamais percé et JavaScript reste bien établi sur son piédestal. Le langage est donc tombé petit à petit dans l'oubli jusqu'à l'arrivée de Flutter courant 2018.

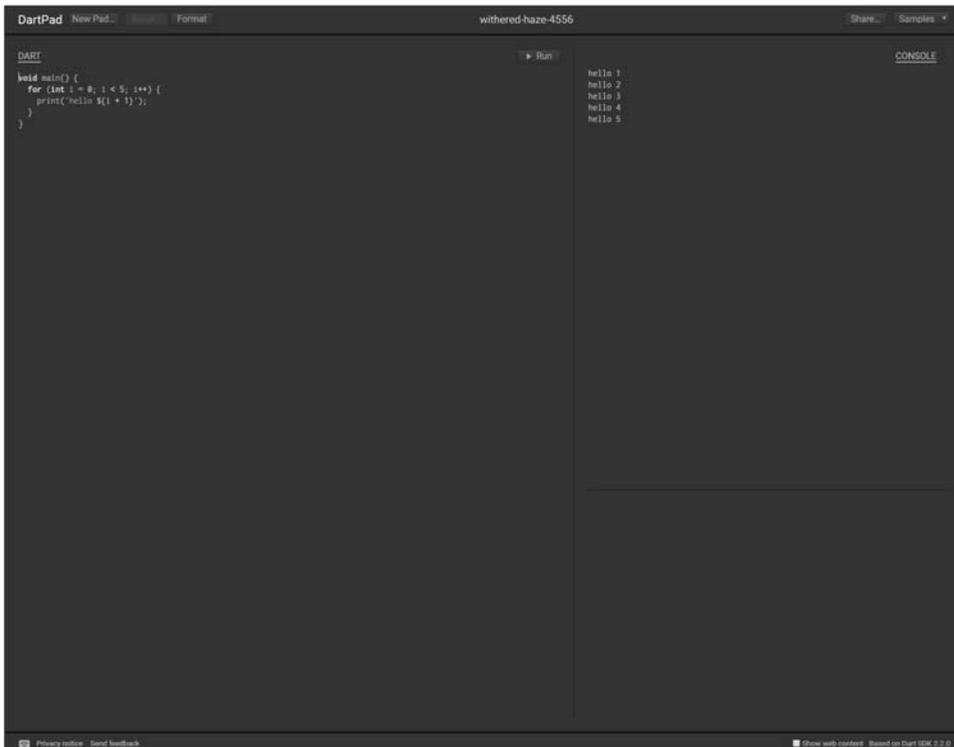
La volonté de pousser le SDK au-devant de la scène, entre autres, a conduit Google à sortir une nouvelle version plus simple, plus rapide et plus intuitive de Dart en début d'année 2018, la 2.0.

Développez vos applications mobiles multiplateformes avec Dart

C'est cette dernière version que nous allons découvrir ici, puisque, comme nous avons déjà pu l'expliquer dans le premier chapitre, Flutter s'appuie sur ce langage. Il est primordial d'en posséder les bases avant de pouvoir pleinement profiter de tous les avantages que nous procure Flutter.

Afin de mettre en œuvre et de tester les exemples qui vont suivre, il est bon de noter l'existence d'un outil extrêmement utile : DartPad.

Vous pourrez le trouver à l'adresse suivante : <https://dartpad.dartlang.org/>



The screenshot shows the DartPad web interface. The top bar includes 'DartPad', 'New Pad...', 'Format', the user ID 'withered-haze-4556', and 'Share...' and 'Samples' options. The main area is split into two panels. The left panel, titled 'DART', contains the following code:

```
void main() {  
  for (int i = 0; i < 5; i++) {  
    print('hello ${i + 1}');  
  }  
}
```

The right panel, titled 'CONSOLE', shows the output of the code:

```
hello 1  
hello 2  
hello 3  
hello 4  
hello 5
```

At the bottom of the interface, there are links for 'Privacy notice' and 'Send feedback', and a note that says 'Based on Dart SDK 2.12.0'.

*DartPad de Google*

## 2. Syntaxe de base

Dans un premier temps, il convient de passer en revue, la syntaxe de base de Dart. Nous allons donc découvrir comment utiliser les variables, les constantes, les collections, les alternatives ainsi que les boucles. Le but du jeu n'est pas d'être exhaustif, mais bien de nous outiller avec les éléments indispensables pour la suite.

### 2.1 Variables

Dart supporte un certain nombre de types comme les nombres, les chaînes de caractères, les booléens, ou encore des collections comme les listes simples ou les listes fonctionnant avec un couple clé/valeur. Nous aborderons ces deux dernières notions un peu plus loin dans ce chapitre.

#### 2.1.1 Les nombres

##### Syntaxe de base

Concernant les nombres, on peut noter la possibilité d'utiliser les entiers (Integer) ou les nombres réels (Double).

L'écriture va se faire de manière assez classique :

```
int monEntier;  
double monReel;
```

##### Remarque

*À ce moment-là, si la variable n'est pas initialisée avec une valeur choisie, elle prendra null comme valeur par défaut. Ce traitement est également valable pour les nombres (entier ou réel) puisqu'en Dart, tout est considéré comme objet. C'est un point notable puisque cela diverge de certains langages.*

L'initialisation est réalisée comme suit :

```
int monEntier = 2;  
double monReel = 1.856;
```

### Héritage de la classe num

Les `int` et les `double` héritent de la classe `num` (les notions d'héritage seront développées plus amplement dans le prochain chapitre). Il est donc possible d'utiliser ce dernier pour garantir que la variable est bien un nombre de type `int` ou `double`. Si ce n'est pas le cas, une erreur de compilation se produira.

```
num unNombre = 318;  
num unNombreDifferent = 3.265464;
```

### La manipulation des nombres

Il est possible de mener différentes opérations sur les nombres telles que des conversions depuis une chaîne de caractères. Pour cela, il suffit d'utiliser la syntaxe suivante :

```
int.tryParse(valeurAConvertir) ;
```

Par exemple, pour convertir la chaîne de caractère « 32 » en entier et l'affecter immédiatement dans une variable il suffit d'écrire ceci :

```
int nombreDepuisChaineDeCaracteres = int.tryParse("32");
```

## 2.1.2 Les chaînes de caractères

### Syntaxe de base

La déclaration et l'initialisation des chaînes de caractères est semblable à celle des nombres :

```
String maChaine = 'Hello World ! ';
```

La valeur de la chaîne est entourée de simple ou double *quote*.

### La manipulation des chaînes de caractères

#### La conversion

Comme dans tant d'autres langages, il existe beaucoup de possibilités qui sont offertes concernant la manipulation de ces chaînes.

On peut noter d'abord la conversion depuis un nombre grâce à la fonction `toString()`. Sa syntaxe est des plus simple :

```
maChaine.toString() ;
```

Il ne faut pas hésiter à utiliser cette fonction très puissante. D'autant qu'il est possible, comme décrit dans le chapitre suivant, de venir modifier le résultat de cette dernière. Voici un exemple :

```
int conv = 12;
String numConv = conv.toString();
```

### La mise en majuscule ou minuscule

Des opérations classiques de mise en minuscules ou majuscules existent également, elles utilisent respectivement les syntaxes suivantes :

```
maChaine.toLowerCase()
```

ou

```
maChaine.toUpperCase()
```

En prenant l'exemple d'une chaîne de caractères nommée `minMaj` qui aurait pour valeur « Bonjour », il serait possible d'utiliser les deux fonctions comme ceci :

```
String minMaj = "Bonjour";
print(minMaj.toLowerCase());
print(minMaj.toUpperCase());
```

### Remarque

*La fonction `print()` sert à afficher la valeur d'un objet dans la console.*

Le résultat de ces opérations sera le suivant :



*Résultats des fonctions `toLowerCase()` et `toUpperCase()`*

### Le découpage de la chaîne de caractères

Enfin, il est également possible de découper une chaîne grâce à la fonction `substring()`. La syntaxe à utiliser est la suivante :

```
maChaîne.substring(valeurDeDepartIncluse, ValeurDeFinExcluse) ;
```

Concrètement, il est possible d'avoir l'exemple suivant :

```
print(minMaj.substring(0,3)); //Bon
```

Le premier paramètre est le point de départ de la scission. Dans l'exemple, le 'B' correspond à la position 0 dans la chaîne de caractères. Ce paramètre est donc inclus dans le résultat.

Le second paramètre, 3, est quant à lui le terminus et sera exclu. Dans l'exemple, il s'agit du 'j'.

La console affichera donc 'Bon'.

### 2.1.3 Les booléens

Un autre type très courant et qui sera extrêmement utile est le type booléen. Prenant les valeurs vrai ou faux, il servira notamment dans l'établissement d'alternatives.

L'écriture se fait comme suit :

```
bool choix1 = true;  
bool choix2 = false;
```

### 2.1.4 Types dynamiques

#### Le type var

Enfin, il peut être intéressant d'évoquer les types `var` et `dynamic`. Ils sont utilisés quand on ne souhaite pas typer une variable au moment de sa déclaration. Au moment de l'initialisation, le système, en fonction de la valeur passée, déterminera lui-même le type approprié.