



Full-Stack Service Programming

Lecture 5

Dart & Flutter 심화 기술의 이해

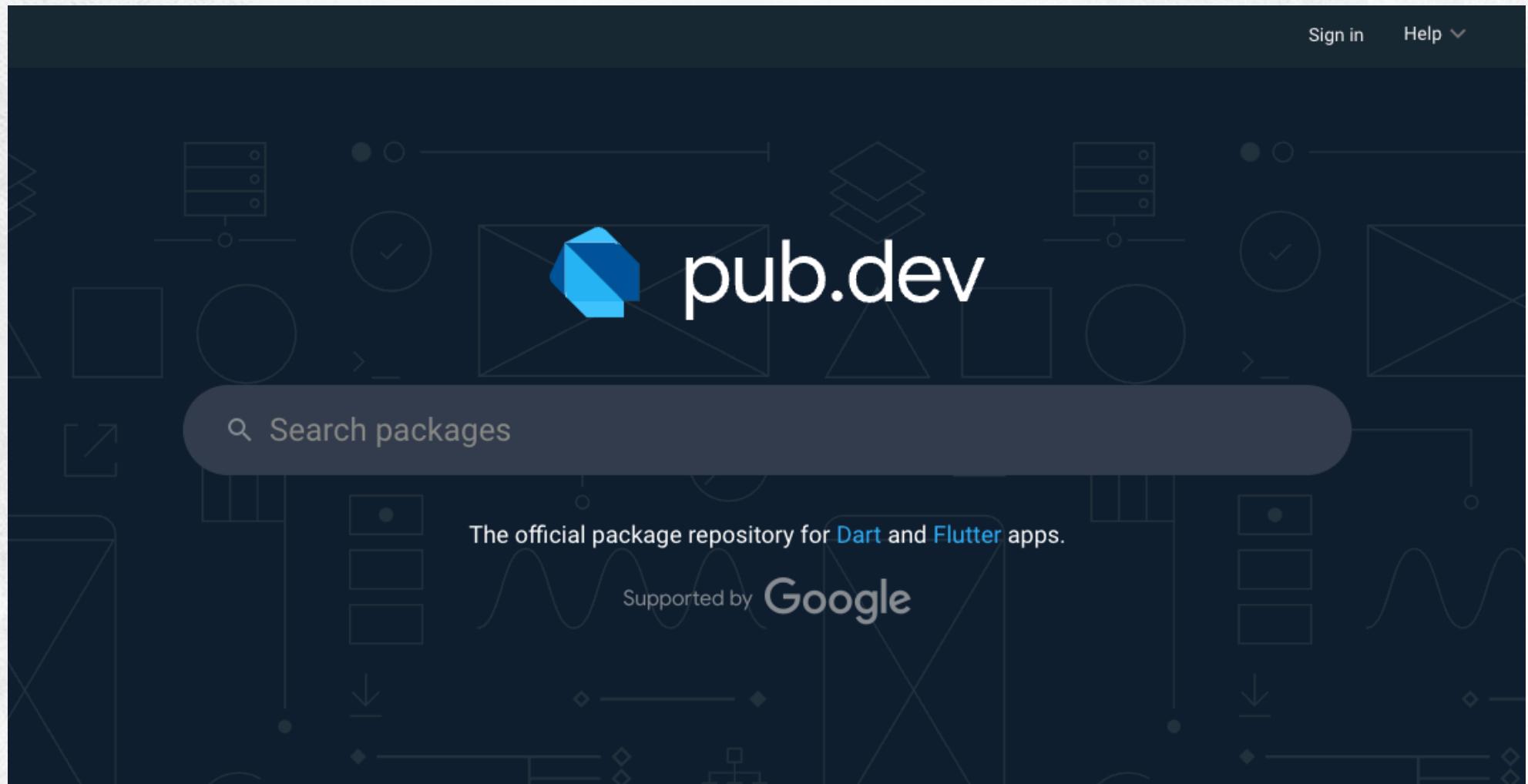
2023. 09. 01

Sungwon Lee

Department of Software Convergence

추가 패키지 활용하기

- Official package repository for Dart and Flutter apps
<https://pub.dev/>



추가 패키지 활용하기

● 예시: Alfred (웹 서버 프레임워크 : 추후 수업 내용)

The screenshot shows the pub.dev search interface with the query 'alfred' entered in the search bar. The results page displays 14 packages. The first result is 'alfred', which is described as a 'performant, expressjs like web server / rest api framework that's easy to use and has all the bits in one place.' It has a version of v1.1.0+2 (56 days ago), BSD-3-Clause, MIT license, and is Dart 3 compatible. It is listed under 'Platforms' (Android, iOS, Linux, macOS, Web, Windows) and 'SDKs'. The package has 309 likes, 140 pub points, and 88% popularity. The second result is 'alfred_workflow', a helper library for Alfred workflows, with 1 like, 140 pub points, and 32% popularity.

pub.dev

Sign in Help ▾

alfred

Platforms RESULTS 14 packages SORT BY SEARCH RELEVANCE

Android

iOS

Linux

macOS

Web

Windows

alfred

A performant, expressjs like web server / rest api framework that's easy to use and has all the bits in one place.

v 1.1.0+2 (56 days ago) BSD-3-Clause, MIT Dart 3 compatible

SDK DART FLUTTER PLATFORM ANDROID IOS LINUX MACOS WINDOWS

SDKs

1 LIKES 140 PUB POINTS 88% POPULARITY

License

Advanced

alfred_workflow

A helper library in Dart for authors of workflows for Alfred.

v 0.3.3 (41 days ago) tusar.dev MIT Dart 3 compatible

SDK DART FLUTTER PLATFORM MACOS

추가 패키지 활용하기

● 예시: Alfred (웹 서버 프레임워크 : 추후 수업 내용)

alfred 1.1.0+2

Published 56 days ago Dart 3 compatible

SDK | DART | FLUTTER | PLATFORM | ANDROID | IOS | LINUX | MACOS | WINDOWS | 309

[Readme](#) [Changelog](#) [Example](#) [Installing](#) [Versions](#) [Scores](#)

Alfred

A performant, expressjs like web server / rest api framework that's easy to use and has all the bits in one place. Dart passing

Quickstart:

```
import 'package:alfred/alfred.dart';

void main() async {
  final app = Alfred();

  app.get('/example', (req, res) => 'Hello world');

  await app.listen();
}
```

There is also a 6 part video series that walks you through creating a web server using Alfred from start to deployment, including databases and authentication. You can find that here:
<https://www.youtube.com/playlist?list=PLkEq83S97rEWsgFEzwBW2pxB7pRYb9wAB>

Index

- [Core principles](#)
- [Usage overview](#)
 - [Quick start guide](#)
- [Routing & incoming requests](#)

309 | 140 | 88%
LIKES PUB POINTS POPULARITY

Publisher
unverified uploader

Metadata
A performant, expressjs like web server / rest api framework that's easy to use and has all the bits in one place.

[Repository \(GitHub\)](#)

[Documentation](#)

[API reference](#)

License
BSD-3-Clause, MIT ([LICENSE](#))

Dependencies
meta, mime, mime_type, path, queue

More
[Packages that depend on alfred](#)

추가 패키지 활용하기

● 예시: Alfred (웹 서버 프레임워크 : 추후 수업 내용)

alfred 1.1.0+2

Published 56 days ago Dart 3 compatible

SDK | DART FLUTTER PLATFORM | ANDROID IOS LINUX MACOS WINDOWS

309

Readme Changelog Example **Installing** Versions Scores

309 | 140 | 88% LIKES PUB POINTS POPULARITY

Use this package as a library

Depend on it

Run this command:

With Dart:

```
$ dart pub add alfred
```

With Flutter:

```
$ flutter pub add alfred
```

This will add a line like this to your package's pubspec.yaml (and run an implicit dart pub get):

```
dependencies:
  alfred: ^1.1.0+2
```

Alternatively, your editor might support dart pub get or flutter pub get. Check the docs for your editor to learn more.

Import it

Now in your Dart code, you can use:

```
import 'package:alfred/alfred.dart';
```

Publisher
unverified uploader

Metadata
A performant, expressive like web server / rest api framework that's easy to use and has all the bits in one place.

Repository (GitHub)

Documentation

API reference

License
BSD-3-Clause, MIT (LICENSE)

Dependencies
meta, mime, mime_type, path, queue

More
Packages that depend on alfred

추가 패키지 활용하기

● 예시: Hive (In-App 데이터베이스 : 추후 수업 내용)

hive

4840 LIKES | 110 PUB POINTS | 100% POPULARITY

Lightweight and blazing fast key-value database written in pure Dart. Strongly encrypted using AES-256.

v 2.2.3 (12 months ago) / 3.0.0-dev (8 months ago) hivedb.dev unknown Dart 3 compatible

SDK DART FLUTTER PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS

hive_generator

279 LIKES | 110 PUB POINTS | 98% POPULARITY

Extension for Hive. Automatically generates TypeAdapters to store any class.

v 2.0.0 (8 months ago) hivedb.dev unknown Dart 3 compatible

SDK DART PLATFORM LINUX MACOS WINDOWS

API result: [hive_generator/hive_generator-library.html](#)

hive_local_storage

5 LIKES | 140 PUB POINTS | 79% POPULARITY

A cache helper which uses hive and flutter_secure_storage to make ease to store session and encrypted data

v 1.0.6 (45 days ago) kishormainali.com MIT Dart 3 compatible

SDK FLUTTER PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS

VOLUME.H CHAPTER.2

추가 패키지 활용하기

● 예시: Hive (In-App 데이터베이스 : 추후 수업 내용)

hive 2.2.3 

Published 12 months ago · [hivedb.dev](#) (Dart 3 compatible) · Latest: 2.2.3 / Prerelease: 3.0.0-dev

SDK DART FLUTTER PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS  4.8K

[Readme](#) [Changelog](#) [Example](#) [Installing](#) [Versions](#) [Scores](#)


Fast, Enjoyable & Secure NoSQL Database

 tests <https://github.com/badges/shields/issues/8671>  coverage 87%  pub.dev v2.2.3  license Apache-2.0

Hive is a lightweight and blazing fast key-value database written in pure Dart. Inspired by Bitcask.

[Documentation & Samples](#) 

If you need queries, multi-isolate support or links between objects check out [Isar Database](#).

Features

-  Cross platform: mobile, desktop, browser
-  Great performance (see [benchmark](#))
-  Simple, powerful, & intuitive API
-  Strong encryption built in
-  NO native dependencies
-  Batteries included

[Getting Started](#)

4840 | 110 | 100%
LIKES PUB POINTS POPULARITY

Publisher [hivedb.dev](#)

Metadata
Lightweight and blazing fast key-value database written in pure Dart. Strongly encrypted using AES-256.

Repository (GitHub)

Documentation

Documentation

API reference

License
unknown (LICENSE)

Dependencies
[crypto](#), [meta](#)

More
[Packages that depend on hive](#)

추가 패키지 활용하기

● 예시: Hive (In-App 데이터베이스 : 추후 수업 내용)

hive 2.2.3

Published 12 months ago • [hivedb.dev](#) Dart 3 compatible • Latest: [2.2.3](#) / Prerelease: [3.0.0-dev](#)

SDK DART FLUTTER PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS

4.8K

Readme Changelog Example [Installing](#) Versions Scores

4841 LIKES 110 PUB POINTS 100% POPULARITY

Use this package as a library

Depend on it

Run this command:

With Dart:

```
$ dart pub add hive
```

With Flutter:

```
$ flutter pub add hive
```

This will add a line like this to your package's pubspec.yaml (and run an implicit dart pub get):

```
dependencies:
  hive: ^2.2.3
```

Alternatively, your editor might support dart pub get or flutter pub get. Check the docs for your editor to learn more.

Import it

Now in your Dart code, you can use:

```
import 'package:hive/hive.dart';
```

Publisher [hivedb.dev](#)

Metadata

Lightweight and blazing fast key-value database written in pure Dart. Strongly encrypted using AES-256.

Repository ([GitHub](#))

Documentation

[Documentation](#) [API reference](#)

License

[unknown \(LICENSE\)](#)

Dependencies

[crypto, meta](#)

More

[Packages that depend on hive](#)

표준 패키지 확인하기

● 예시: Math (dart:math)

The screenshot shows the pub.dev search interface. The search bar at the top contains the query 'math'. Below the search bar, the results section is titled 'RESULTS 1124 packages'. A 'SORT BY SEARCH RELEVANCE' button is visible. On the left, there is a sidebar with filter options for 'Platforms' (Android, iOS, Linux, macOS, Web, Windows), 'SDKs', 'License', and 'Advanced'. The main results area displays two packages: 'dart:math' and 'math_expressions'. The 'dart:math' package is described as a 'Core library' with 'Null safety' support, version v3.0.5, and a Dart SDK library. It has 253 likes, 140 pub points, and 97% popularity. The 'math_expressions' package is described as a library for parsing and evaluating mathematical expressions, supporting real numbers, vectors, and basic interval arithmetic, version v2.4.0 (3 months ago). It has 253 likes, 140 pub points, and 97% popularity.

pub.dev

Sign in Help ▾

math

Platforms RESULTS 1124 packages SORT BY SEARCH RELEVANCE

Android

iOS

Linux

macOS

Web

Windows

dart:math

Mathematical constants and functions, plus a random number generator.

v 3.0.5 • Dart SDK library Core library Null safety

API results: [dart-math/dart-math-library.html](#)

253 LIKES 140 PUB POINTS 97% POPULARITY

math_expressions

A library for parsing and evaluating mathematical expressions, supporting real numbers, vectors, and basic interval arithmetic.

v 2.4.0 (3 months ago) [leonhardt.co.nz](#) MIT Dart 3 compatible

SDK DART FLUTTER PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS

Most popular packages

Some of the most downloaded packages over the past 60 days



cupertino_icons

Default icons asset for Cupertino widgets based on Apple styled icons

[flutter.dev](#)

get_it

Simple direct Service Locator that allows to decouple the interface from a concrete

[fluttercommunity.dev](#)

badges

A package for creating badges. Badges can be used for an additional marker for any

[yako.io](#)

go_router

A declarative router for Flutter based on Navigation 2 supporting deep linking, data-

[flutter.dev](#)

font_awesome_flutter

The Font Awesome Icon pack available as Flutter Icons. Provides 1600 additional icons

[fluttercommunity.dev](#)

sse

Provides client and server functionality for setting up bi-directional communication

[tools.dart.dev](#)

[VIEW ALL](#)

Top Flutter packages

Some of the top packages that extend Flutter with new features

[font_awesome_flutter](#)

The Font Awesome Icon pack available as Flutter Icons. Provides 1600 additional icons

[fluttercommunity.dev](#)

[flutter_image_comp...](#)

Compress Pictures. Can effectively reduce the size of the transmission.

[image_picker](#)

Flutter plugin for selecting images from the Android and iOS image library, and taking

[flutter.dev](#)

[mask_text_input_for...](#)

The package provides TextInputFormatter for TextField and TextFormField which format the input by a given mask

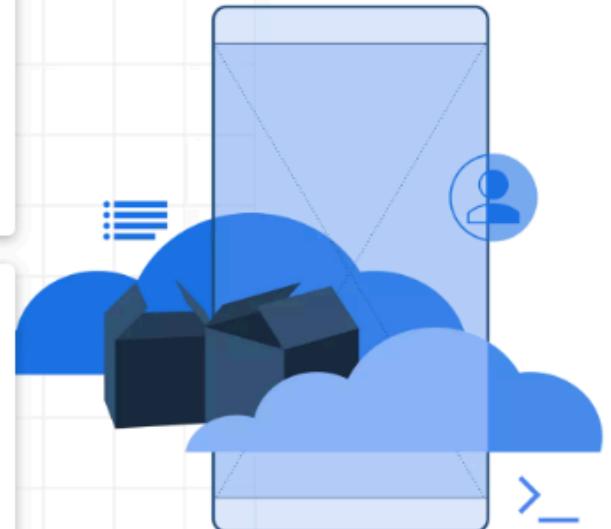
[provider](#)

A wrapper around InheritedWidget to make them easier to use and more

[dash-overflow.net](#)

[sizer](#)

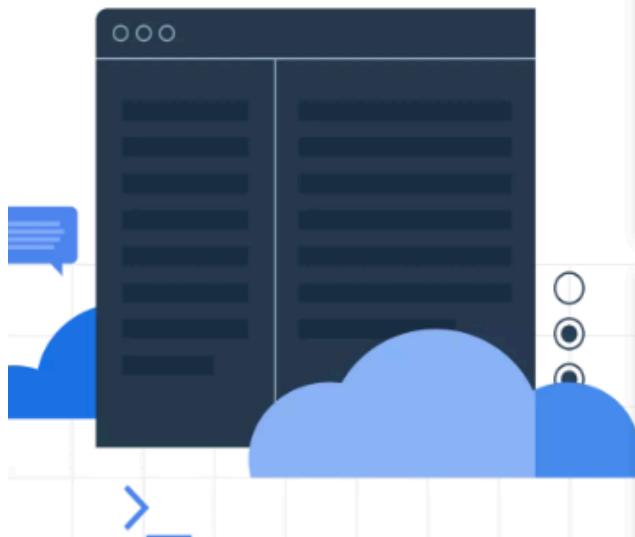
A flutter plugin for Easily make Flutter apps responsive. Automatically adapt UI to different screen sizes. Responsiveness made



[VIEW ALL](#)

Top Dart packages

Some of the top packages for any Dart-based app or program



uuid

RFC4122 (v1, v4, v5) UUID
Generator and Parser for all
Dart platforms (Web, VM,

[yuli.dev](#)

built_collection

Immutable collections based on
the SDK collections. Each SDK
collection class is split into a

[google.dev](#)

email_validator

A simple (but correct) dart class
for validating email addresses

shelf_router

A convenient request router for
the shelf web-framework, with
support for URL-parameters,

[tools.dart.dev](#)

injectable

Injectable is a convenient code
generator for get_it. Inspired by
Angular DI, Guice DI and

[codeness.ly](#)

english_words

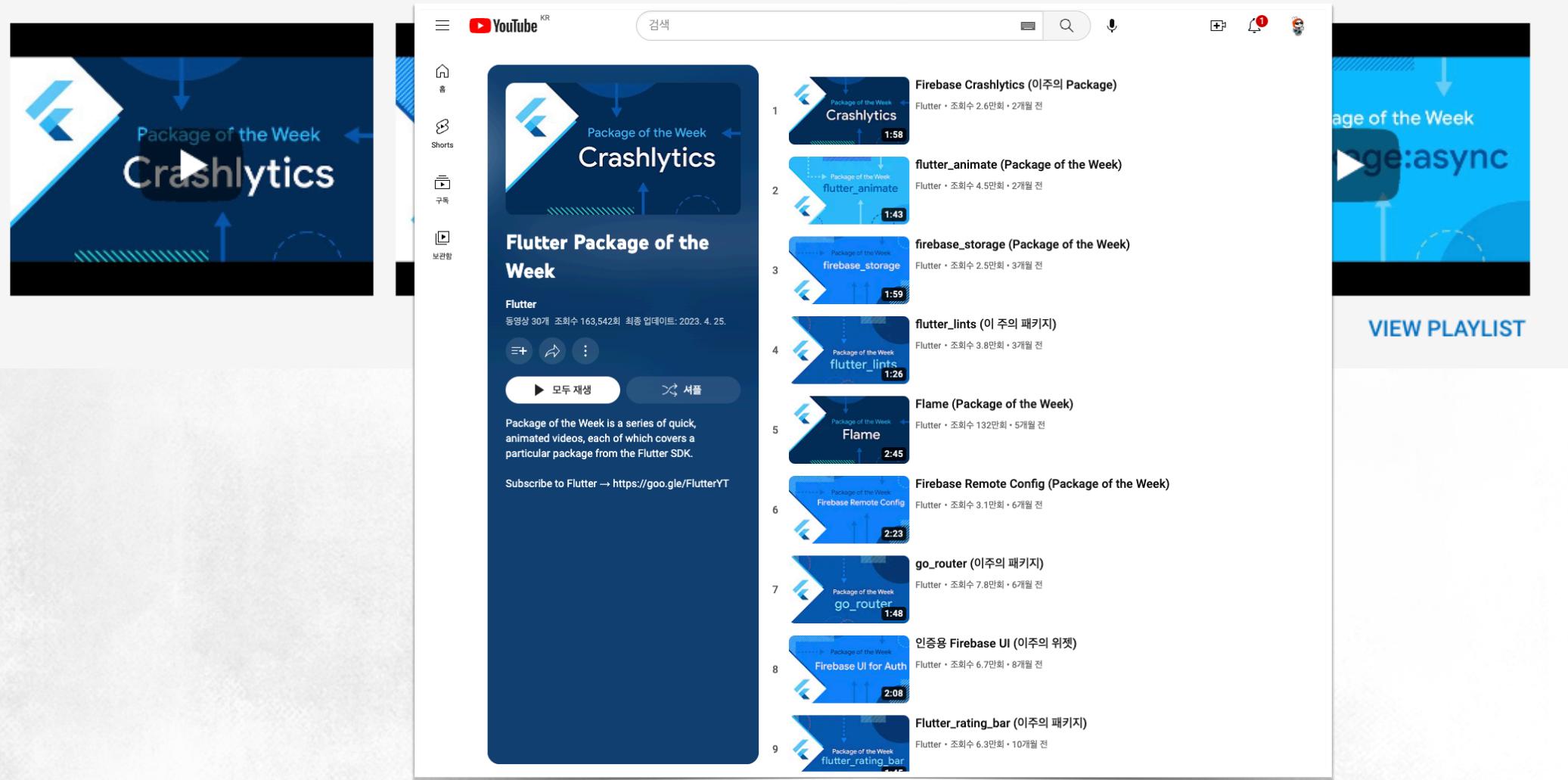
Utilities for working with English
words. Counts syllables,
generates well-sounding word

[filiph.net](#)

[VIEW ALL](#)

Package of the Week

Package of the Week is a series of quick, animated videos, each of which covers a particular package



VOLUME.I CHAPTER.1

Alfred 웹 서버 프레임워크 사용하기

The screenshot shows the pub.dev page for the Alfred package. At the top, there's a navigation bar with the pub.dev logo, a search icon, 'Sign in', and 'Help'. Below the header, the package name 'alfred 1.1.0+2' is displayed with a copy icon. It was published 56 days ago and is Dart 3 compatible. The package has 309 likes, 140 pub points, and 88% popularity. The main content area includes tabs for Readme, Changelog, Example, Installing, Versions, and Scores. The Readme tab is active, showing a brief description: 'A performant, expressjs like web server / rest api framework that's easy to use and has all the bits in one place.' It includes a 'Dart passing' badge. Below the description is a 'Quickstart' section with sample code:

```
import 'package:alfred/alfred.dart';

void main() async {
  final app = Alfred();

  app.get('/example', (req, res) => 'Hello world');

  await app.listen();
}
```

There is also a 6 part video series that walks you through creating a web server using Alfred from start to deployment, including databases and authentication. You can find that here: <https://www.youtube.com/playlist?list=PLkEq83S97rEWsgFEzwBW2pxB7pRYb9wAB>

Index

- Core principles
- Usage overview
 - Quick start guide
- Routing & incoming requests

On the right side, there are sections for Publisher (unverified uploader), Metadata (a performant, expressjs like web server / rest api framework that's easy to use and has all the bits in one place.), Repository (GitHub), Documentation, API reference, License (BSD-3-Clause, MIT (LICENSE)), Dependencies (meta, mime, mime_type, path, queue), and More (Packages that depend on alfred).

Alfred 웹 서버 프레임워크 사용하기

```

1 import 'dart:io';
2 import 'dart:convert';
3
4 Run | Debug
5 Future main() async {
6   var db = <dynamic, dynamic>{};
7
8   var server = await HttpServer.bind(
9     InternetAddress.loopbackIPv4, // ip address
10    4040, // port number
11  );
12
13  printHttpServerActivated(server);
14
15  await for (HttpRequest request in server) {
16    if (request.uri.path.contains('/api/') == true) {
17      printHttpRequestInfo(request);
18      try {
19        switch (request.method) {
20          case 'POST': // Create
21            createDB(db, request);
22            break;
23          case 'GET': // Read
24            readDB(db, request);
25            break;
26          case 'PUT': // Update
27            updateDB(db, request);
28            break;
29          case 'DELETE': // Delete
30            deleteDB(db, request);
31            break;
32          default:
33            print("\$ Unsupported http method");
34        }
35      } catch (err) {
36        print("\$ Exception in http request processing");
37      }
38    } else {
39      printAndSendHttpResponse(
40        db, request, "${request.method} {ERROR: Unsupported API}");
41    }
42  }
}

```

직접 개발

volume-D-chapter-05-server.dart

```

1 import 'dart:io';
2 import 'dart:convert';
3 import 'package:alfred/alfred.dart';
4
5 Run | Debug
6 Future main() async {
7   var db = <dynamic, dynamic>{};
8
9   final app = Alfred();
10
11   app.post('/api/*', (req, res) => createDB(db, req));
12   app.get('/api/*', (req, res) => readDB(db, req));
13   app.put('/api/*', (req, res) => updateDB(db, req));
14   app.delete('/api/*', (req, res) => deleteDB(db, req));
15
16 }

```

Alfred 사용

main.dart

CRUD 서버 개발 예제



VOLUME.I CHAPTER.1

Alfred 웹 서버 프레임워크 사용하기

```
1 import 'dart:io';
2
3 Run | Debug
4 Future main() async {
5   var server = await HttpServer.bind(
6     InternetAddress.loopbackIPv4, // ip address
7     4040, // port number
8   );
9   printHttpServerActivated(server);
10
11   await for (HttpRequest request in server)
12     printHttpRequestInfo(request);
13   try {
14     switch (request.method) {
15       case 'GET':
16         httpGetHandler(request);
17         break;
18       default:
19         print("\$ Unsupported http method");
20     } catch (err) {
21     print("\$ Exception in http request processing");
22   }
23 }
24
25 void printHttpServerActivated(HttpServer server) {
26   var ip = server.address.address;
27   var port = server.port;
28   print('\$ Server activated in \$ip:\$port');
29 }
30
31 void printHttpRequestInfo(HttpRequest request) async {
32   var ip = request.connectionInfo.remoteAddress.address;
33   var port = request.connectionInfo.remotePort;
34   var method = request.method;
35   var path = request.uri.path;
36   print("\$ \$method \$path from \$ip:\$port");
37
38   if (request.headers.contentLength != -1) {
39     print("\$> content-type : \${request.headers.contentType}");
40     print("\$> content-length : \${request.headers.contentLength}");
41   }
42 }
43
44 void httpGetHandler(HttpRequest request) async {
45   var fileName = request.uri.path.substring(1);
46   var fileType = request.uri.path.split('.').last;
47
48   var contentType = {
49     'html': 'text/html',
50     'js': 'application/javascript',
51     'json': 'application/json',
52     'wasm': 'application/wasm',
53     'otf': 'application/x-font-opentype',
54     'ttf': 'application/x-font-ttf',
55     'png': 'image/png',
56     'jpg': 'image/jpeg',
57   };
58
59   try {
60     if (await File(fileName).exists() == true) {
61       request.response.statusCode = HttpStatus.ok;
62       request.response.headers.contentType =
63         ContentType.parse(contentType[fileType]!);
64       var file = new File(fileName);
65       file.readAsBytes().then((List<int> bytes) {
66         bytes.forEach((int b) => request.response.writeCharCode(b));
67         request.response.close();
68       });
69     } else {
70       var content = "File Not found";
71       request.response
72         ..headers.contentType = ContentType('text', 'plain', charset: "utf-8")
73         ..headers.contentLength = content.length
74         ..statusCode = HttpStatus.notFound
75         ..write(content);
76       await request.response.close();
77     }
78   } catch (err) {
79     print("\$ Exception in httpGetHandler()");
80   }
81 }
```

직접 개발
volume-G-chapter-04
simpleHttpFileServer.dart

```
1 import 'dart:io';
2 import 'package:alfred/alfred/dart';
3 import "package:path/path.dart" show dirname;
4
5 void main() async {
6   final app = Alfred();
7   app.get('/*', (req, res) => Directory("build/web"));
8   await app.listen(4040);
9 }
```

Alfred 사용

python -m http.server
동일 기능 프로그램 개발

● main.dart

(volume-D-chapter-05-server.dart의 Alfred 버전)

VOLUME.I CHAPTER.2

Hive 데이터베이스 활용하기

pub.dev

hive 2.2.3

Published 12 months ago • [hivedb.dev](#) (Dart 3 compatible) • Latest: 2.2.3 / Prerelease: 3.0.0-dev

SDK | DART | FLUTTER | PLATFORM | ANDROID | IOS | LINUX | MACOS | WEB | WINDOWS

4.8K

Readme Changelog Example Installing Versions Scores

4841 LIKES | 110 PUB POINTS | 100% POPULARITY

 hive

Fast, Enjoyable & Secure NoSQL Database

[tests](#) <https://github.com/badges/shields/issues/8671> [coverage](#) 87% [pub.dev](#) v2.2.3 [license](#) Apache-2.0

Hive is a lightweight and blazing fast key-value database written in pure Dart. Inspired by [Bitcask](#).

[Documentation & Samples](#)

If you need queries, multi-isolate support or links between objects check out [Isar Database](#).

Features

- 🚀 Cross platform: mobile, desktop, browser
- ⚡ Great performance (see [benchmark](#))
- ❤️ Simple, powerful, & intuitive API
- 🔒 Strong encryption built in
- 📍 NO native dependencies
- 🔋 Batteries included

Getting Started

Check out the [Quick Start](#) documentation to get started.

Publisher: [hivedb.dev](#)

Metadata:
Lightweight and blazing fast key-value database written in pure Dart. Strongly encrypted using AES-256.

Repository ([GitHub](#))

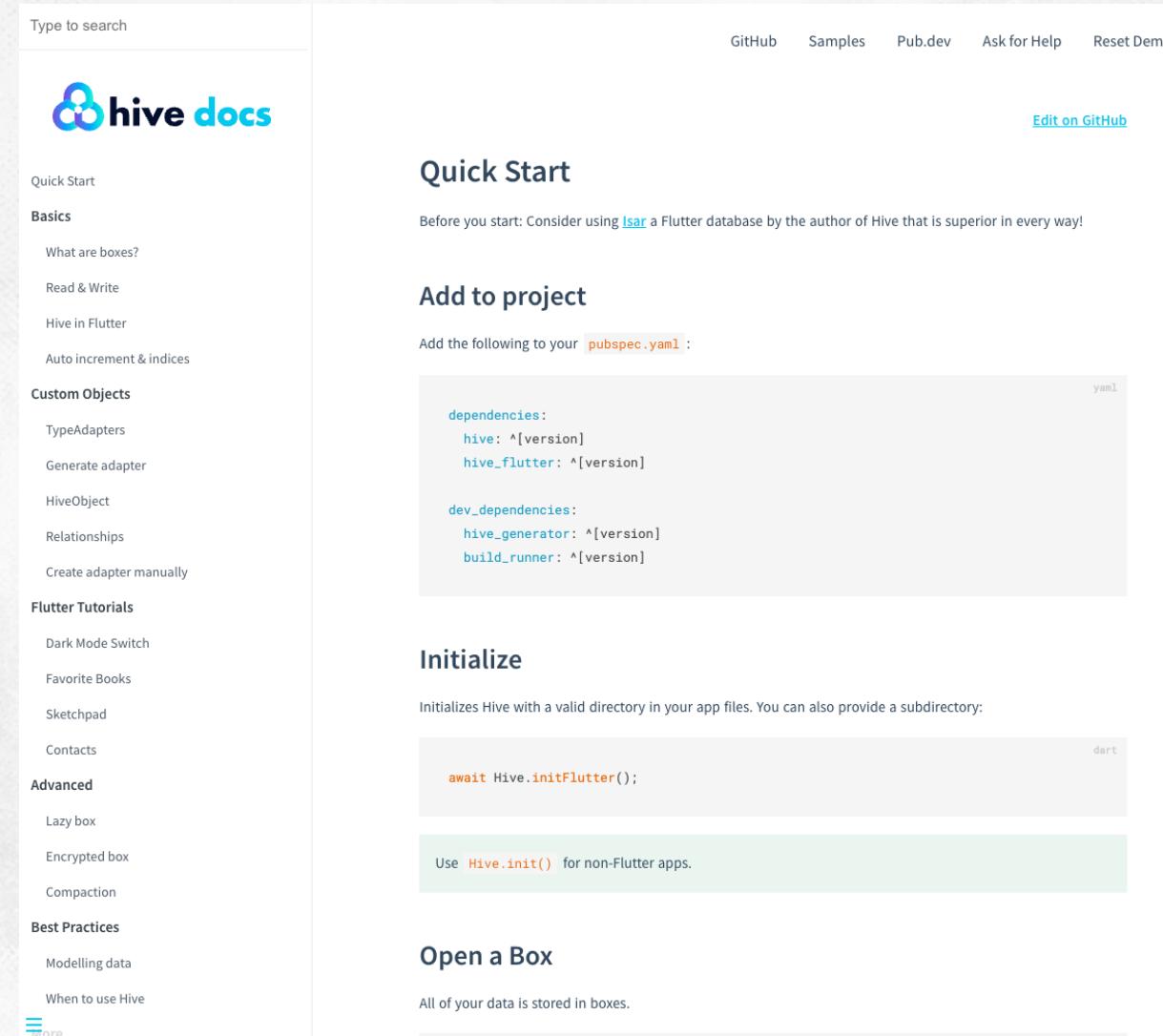
Documentation:
[Documentation](#) [API reference](#)

License: [unknown \(LICENSE\)](#)

Dependencies: [crypto](#), [meta](#)

More: [Packages that depend on hive](#)

Quick Start : <https://docs.hivedb.dev/#/?id=quick-start>



The screenshot shows the 'Quick Start' page of the Hive Docs website. The left sidebar contains a search bar and a navigation menu with sections like 'Quick Start', 'Basics', 'Custom Objects', 'Flutter Tutorials', 'Advanced', 'Best Practices', and a 'more' section. The main content area has a title 'Quick Start' and a note about using lsar instead. It includes sections for 'Add to project' (with code snippets for pubspec.yaml and dev_dependencies), 'Initialize' (with Dart code for await Hive.initFlutter()), and 'Open a Box' (noting that all data is stored in boxes). Top navigation links include GitHub, Samples, Pub.dev, Ask for Help, and Reset Demos.

Type to search

GitHub Samples Pub.dev Ask for Help Reset Demos

hive docs

Quick Start

Before you start: Consider using [lsar](#) a Flutter database by the author of Hive that is superior in every way!

Add to project

Add the following to your `pubspec.yaml`:

```
dependencies:  
  hive: ^[version]  
  hive_flutter: ^[version]
```

```
dev_dependencies:  
  hive_generator: ^[version]  
  build_runner: ^[version]
```

Initialize

Initializes Hive with a valid directory in your app files. You can also provide a subdirectory:

```
await Hive.initFlutter();
```

Use `Hive.init()` for non-Flutter apps.

Open a Box

All of your data is stored in boxes.

Hive 데이터베이스 활용하기 (리뷰 & 실습)

```
drsungwon~$ dart create hellohive
Creating hellohive using template console...
```

```
.gitignore
analysis_options.yaml
CHANGELOG.md
pubspec.yaml
README.md
bin/hellohive.dart
lib/hellohive.dart
test/hellohive_test.dart
```

```
Running pub get...          3.6s
Resolving dependencies...
Changed 47 dependencies!
```

Created project hellohive in hellohive! In order to get started, run the following commands:

```
cd hellohive
dart run
```

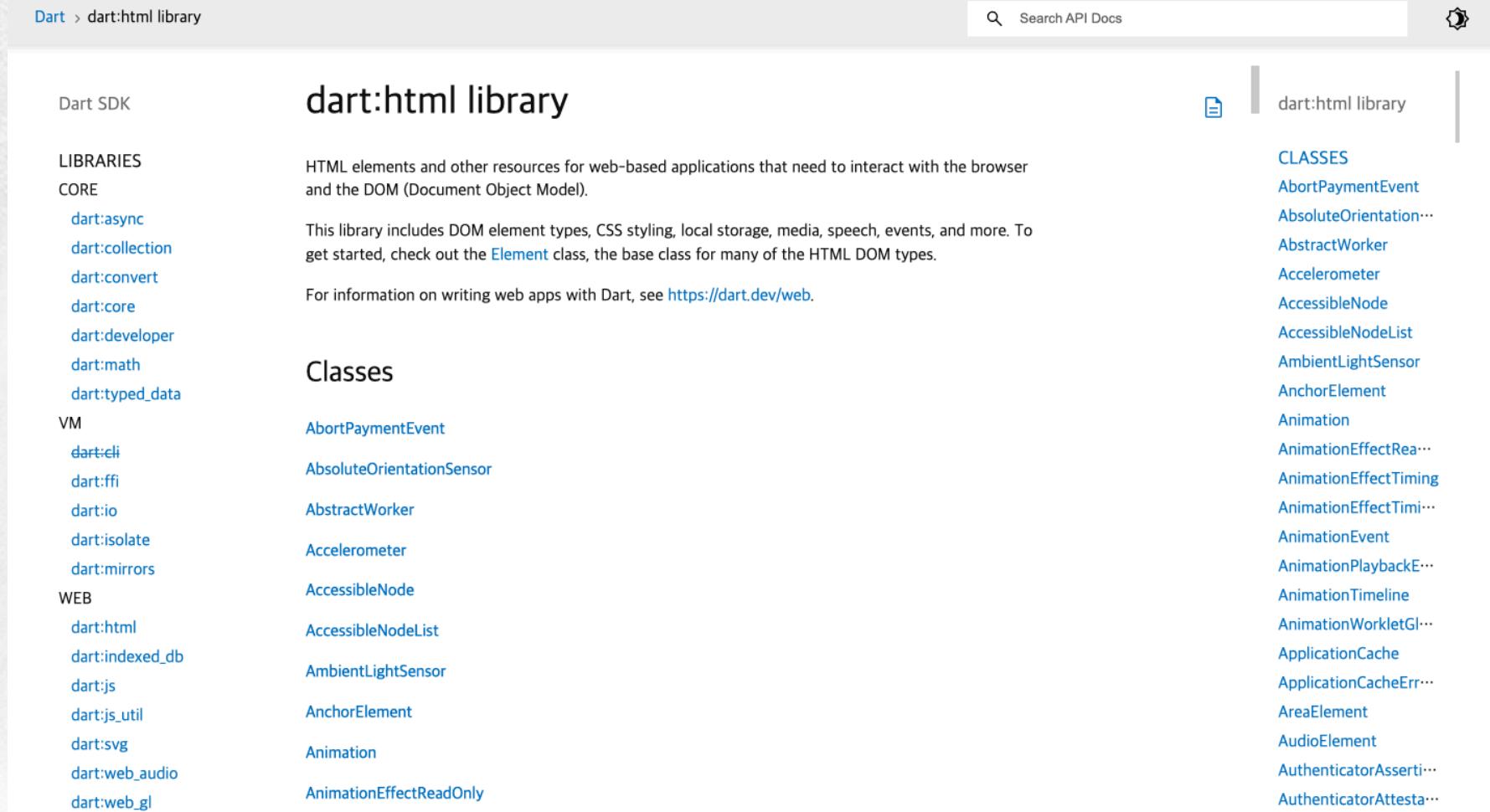
```
drsungwon~$ cd hellohive
drsungwon~$ dart pub add hive
Resolving dependencies...
+ hive 2.2.3
Changed 1 dependency!
```

```
====> volume-I-chapter-02-hellohive_bin.dart 파일을 {프로젝트폴더}/bin/main.dart로 복사
====> volume-I-chapter-02-hellohive_v2_lib.dart 파일을 {프로젝트폴더}/lib/main.dart로 복사
```

```
drsungwon~$ dart run
Building package executable...
Built hellohive:hellohive.
STUD#0001 : STUD#0001: Dart - Seoul
STUD#0002 : STUD#0002: Flutter - Bundang
drsungwon~$
```

● Dart:HTML

enables HTML/CSS/JavaScript -> HTML/CSS/Dart



The screenshot shows the Dart API documentation for the `dart:html` library. The left sidebar lists various Dart libraries, and the main content area is dedicated to the `dart:html` library. The right sidebar lists all the classes available in this library.

Dart SDK

- LIBRARIES**
 - CORE
 - `dart:async`
 - `dart:collection`
 - `dart:convert`
 - `dart:core`
 - `dart:developer`
 - `dart:math`
 - `dart:typed_data`
 - VM
 - `dart:cli`
 - `dart:ffi`
 - `dart:io`
 - `dart:isolate`
 - `dart:mirrors`
 - WEB
 - `dart:html`
 - `dart:indexed_db`
 - `dart:js`
 - `dart:js_util`
 - `dart:svg`
 - `dart:web_audio`
 - `dart:web_gl`

dart:html library

CLASSES

- `AbortPaymentEvent`
- `AbsoluteOrientationSensor`
- `AbstractWorker`
- `Accelerometer`
- `AccessibleNode`
- `AccessibleNodeList`
- `AmbientLightSensor`
- `AnchorElement`
- `Animation`
- `AnimationEffectRea...`
- `AnimationEffectTiming`
- `AnimationEffectTimi...`
- `AnimationEvent`
- `AnimationPlaybackE...`
- `AnimationTimeline`
- `AnimationWorkletGl...`
- `ApplicationCache`
- `ApplicationCacheErr...`
- `AreaElement`
- `AudioElement`
- `AuthenticatorAsserti...`
- `AuthenticatorAttesta...`

Dart for Web 활용하기

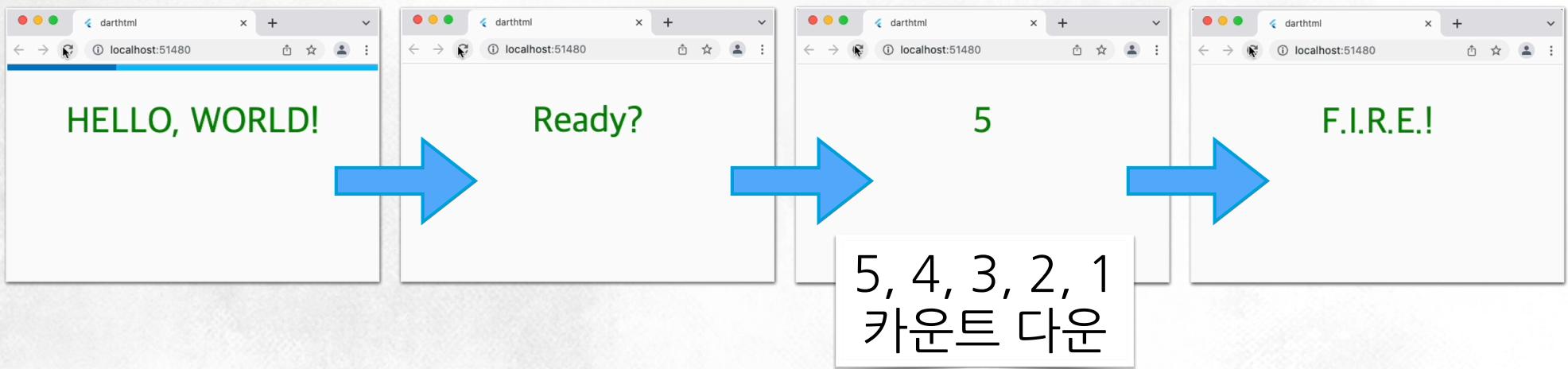
```
1 import 'dart:html';
2 import 'dart:core';
3
4 void main() async {
5   final header = querySelector('#target');
6   header?.text = "Ready?";
7   await Future.delayed(Duration(seconds: 2), () => {});
8
9   for (var count = 5; count > 0; count--) {
10     header?.text = "$count";
11     await Future.delayed(Duration(seconds: 1), () => {});
12   }
13
14   header?.text = "F.I.R.E!";
15 }
```

```
</script>
<center>
<font size ="20em" color="green">
<p id="target">
    HELLO, WORLD!
</p>
</font>
</center>
</body>
</html>
```

index.html

Dart for Web 활용하기 (리뷰 및 실습)

- flutter create hellodarthtml
- {프로젝트폴더}/lib/main.dart 교체
- {프로젝트폴터}/web/index.html 교체
- flutter run -d chrome

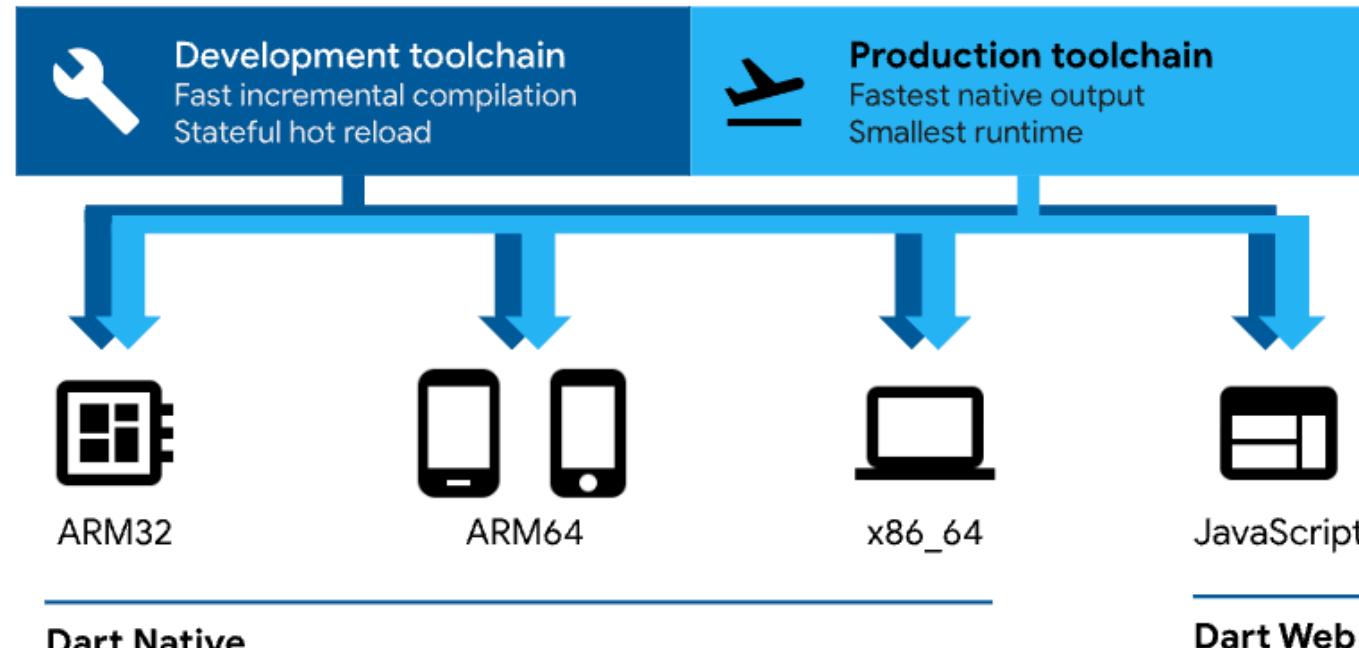


● Dart for Native Platform : <https://dart.dev/overview#platform>

Dart: The platforms

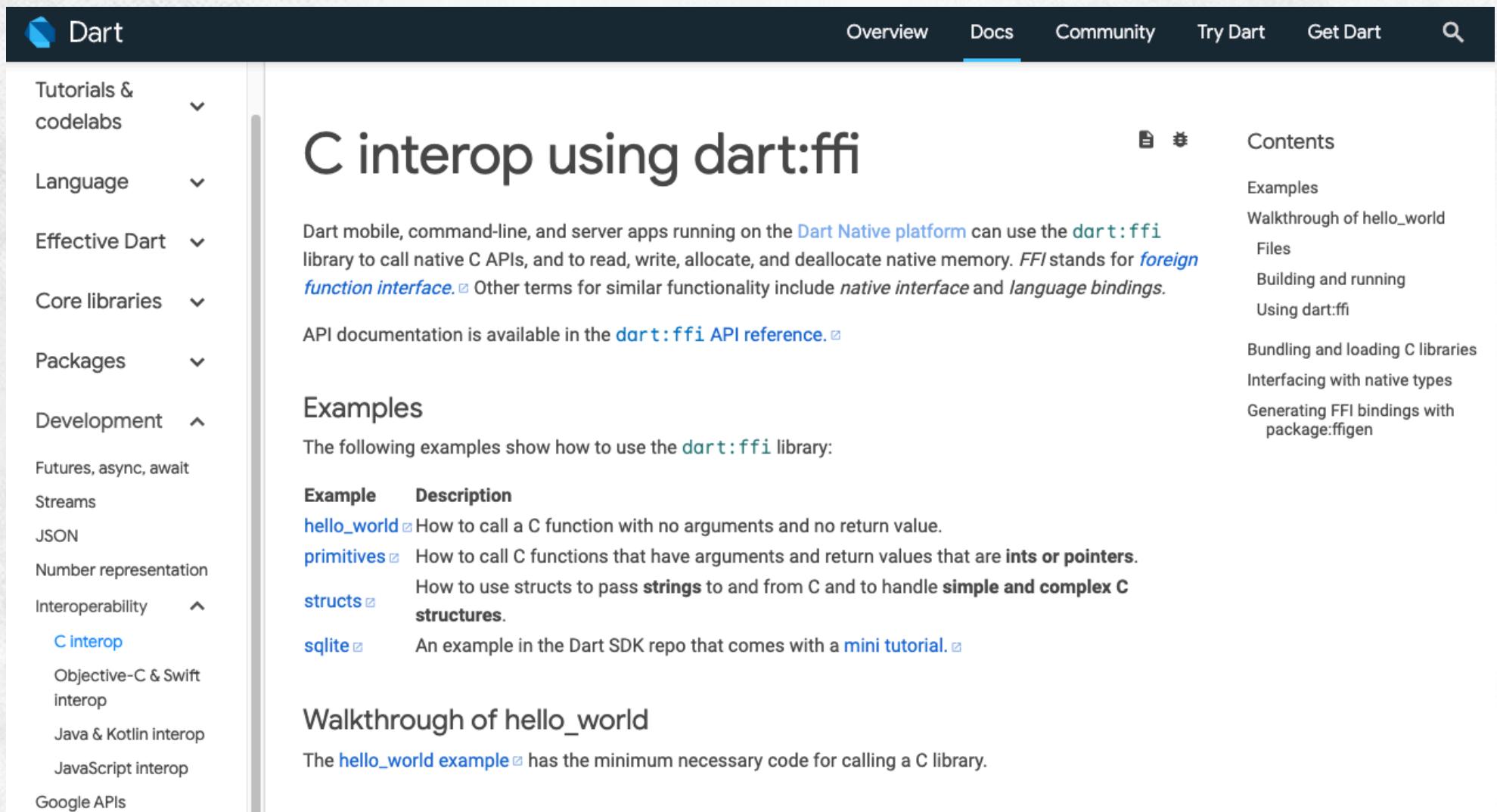
Dart's compiler technology lets you run code in different ways:

- **Native platform:** For apps targeting mobile and desktop devices, Dart includes both a Dart VM with just-in-time (JIT) compilation and an ahead-of-time (AOT) compiler for producing machine code.
- **Web platform:** For apps targeting the web, Dart can compile for development or production purposes. Its web compiler translates Dart into JavaScript.



dart:ffi for C Language Interop

- <https://dart.dev/guides/libraries/c-interop>



The screenshot shows the Dart Dev website with the URL <https://dart.dev/guides/libraries/c-interop> in the address bar. The page title is "C interop using dart:ffi". The left sidebar has a "Development" section expanded, showing "C interop" selected. The main content area contains an introduction to dart:ffi, examples, and a walkthrough of the hello_world example.

C interop using dart:ffi

Dart mobile, command-line, and server apps running on the [Dart Native platform](#) can use the [dart:ffi](#) library to call native C APIs, and to read, write, allocate, and deallocate native memory. FFI stands for [foreign function interface](#). Other terms for similar functionality include *native interface* and *language bindings*.

API documentation is available in the [dart:ffi API reference](#).

Examples

The following examples show how to use the [dart:ffi](#) library:

Example	Description
hello_world	How to call a C function with no arguments and no return value.
primitives	How to call C functions that have arguments and return values that are ints or pointers .
structs	How to use structs to pass strings to and from C and to handle simple and complex C structures .
sqlite	An example in the Dart SDK repo that comes with a mini tutorial .

Walkthrough of hello_world

The [hello_world example](#) has the minimum necessary code for calling a C library.

- <https://docs.flutter.dev/platform-integration/android/c-interop>

The screenshot shows the Flutter documentation website. The top navigation bar includes links for Multi-Platform, Development, Ecosystem, Showcase, Docs, a search icon, social media icons for Twitter and YouTube, and a 'Get started' button. On the left, a sidebar menu is open under the 'Platform integration' section, showing options like 'Supported platforms', 'Build desktop apps with Flutter', 'Write platform-specific code', 'Android' (with sub-options for splash screens, native code binding, host views, state restoration, and ChromeOS targeting), and 'iOS'. The main content area features a large title 'Binding to native Android code using dart:ffi' with a subtitle 'Flutter mobile and desktop apps can use the `dart:ffi` library to call native C APIs. FFI stands for *foreign function interface*. Other terms for similar functionality include *native interface* and *language bindings*'. A note box states: 'Note: This page describes using the `dart:ffi` library in Android apps. For information on iOS, see [Binding to native iOS code using dart:ffi](#). For information in macOS, see [Binding to native macOS code using dart:ffi](#). This feature is not yet supported for web plugins.' Below this, a paragraph explains the prerequisites for using FFI, and another paragraph describes a tutorial for bundling C/C++ sources in a Flutter plugin.

Flutter

Multi-Platform ▾ Development ▾ Ecosystem ▾ Showcase Docs ▾ **Get started**

Navigation & routing

Data & backend

Accessibility & localization

Platform integration

- Supported platforms
- Build desktop apps with Flutter
- Write platform-specific code
- Android
 - Add a splash screen
 - Bind to native code**
 - Host a native Android view
 - Restore state on Android
 - Target ChromeOS with Android
- iOS

Binding to native Android code using dart:ffi

Platform integration > Android > Binding to native Android code using dart:ffi

Flutter mobile and desktop apps can use the `dart:ffi` library to call native C APIs. FFI stands for *foreign function interface*. Other terms for similar functionality include *native interface* and *language bindings*.

Note: This page describes using the `dart:ffi` library in Android apps. For information on iOS, see [Binding to native iOS code using dart:ffi](#). For information in macOS, see [Binding to native macOS code using dart:ffi](#). This feature is not yet supported for web plugins.

Before your library or program can use the FFI library to bind to native code, you must ensure that the native code is loaded and its symbols are visible to Dart. This page focuses on compiling, packaging, and loading Android native code within a Flutter plugin or app.

This tutorial demonstrates how to bundle C/C++ sources in a Flutter plugin and bind to them using the Dart FFI library on both Android and iOS. In this walkthrough, you'll create a C function that implements 32-bit addition and then exposes it through a Dart plugin named "native_add".

- <https://docs.flutter.dev/platform-integration/ios/c-interop>

The screenshot shows a Flutter documentation page. The top navigation bar includes links for Multi-Platform, Development, Ecosystem, Showcase, Docs, a search icon, and social media links for Twitter, YouTube, and Medium. A blue 'Get started' button is also present. On the left, a sidebar menu is open under the 'Platform integration' section, showing options like Accessibility & localization, Supported platforms, Build desktop apps with Flutter, Write platform-specific code, and detailed sections for Android and iOS. The main content area features a large title 'Binding to native iOS code using dart:ffi'. Below it, a breadcrumb trail shows 'Platform integration > iOS > Binding to native iOS code using dart:ffi'. A note states: 'Flutter mobile and desktop apps can use the `dart:ffi` library to call native C APIs. FFI stands for *foreign function interface*. Other terms for similar functionality include *native interface* and *language bindings*'. A callout box contains a note: 'Note: This page describes using the `dart:ffi` library in iOS apps. For information on Android, see [Binding to native Android code using dart:ffi](#). For information in macOS, see [Binding to native macOS code using dart:ffi](#). This feature is not yet supported for web plugins.' The main text explains that before using the FFI library, native code must be loaded and its symbols made visible to Dart. It then provides a tutorial on bundling C/C++ sources in a Flutter plugin and using the Dart FFI library on iOS to implement 32-bit addition.

Flutter

Multi-Platform ▾ Development ▾ Ecosystem ▾ Showcase Docs ▾ [Get started](#)

Accessibility & localization ▾

Platform integration ▾

- Supported platforms
- Build desktop apps with Flutter
- Write platform-specific code
- ▶ Android
- ▼ iOS
- Leverage Apple's system libraries
- Add a launch screen
- Add iOS App Clip support
- Add iOS app extensions
- [Bind to native code](#)
- Host a native iOS view
- Enable debugging on iOS

Binding to native iOS code using `dart:ffi`

Platform integration > iOS > Binding to native iOS code using `dart:ffi`

Flutter mobile and desktop apps can use the `dart:ffi` library to call native C APIs. *FFI* stands for *foreign function interface*. Other terms for similar functionality include *native interface* and *language bindings*.

Note: This page describes using the `dart:ffi` library in iOS apps. For information on Android, see [Binding to native Android code using `dart:ffi`](#). For information in macOS, see [Binding to native macOS code using `dart:ffi`](#). This feature is not yet supported for web plugins.

Before your library or program can use the FFI library to bind to native code, you must ensure that the native code is loaded and its symbols are visible to Dart. This page focuses on compiling, packaging, and loading iOS native code within a Flutter plugin or app.

This tutorial demonstrates how to bundle C/C++ sources in a Flutter plugin and bind to them using the Dart FFI library on iOS. In this walkthrough, you'll create a C function that implements 32-bit addition and then exposes it through a Dart plugin named "native_add".

Contents

- [Dynamic vs static linking](#)
- [Create an FFI plugin](#)
- [Other use cases](#)
- [iOS and macOS](#)
 - [Platform library](#)
 - [First-party library](#)
 - [Source code](#)
 - [Compiled \(dynamic\) library](#)
 - [Open-source third-party library](#)
 - [Closed-source third-party library](#)
 - [Stripping iOS symbols](#)

- <https://docs.flutter.dev/platform-integration/macos/c-interop>

The screenshot shows the Flutter documentation website. The top navigation bar includes links for Multi-Platform, Development, Ecosystem, Showcase, Docs, a search bar, and social media icons. A blue "Get started" button is on the right. On the left, a sidebar menu is open under "Platform integration", showing options like "Supported platforms", "Build desktop apps with Flutter", "Write platform-specific code", and detailed sections for "Android", "iOS", "Linux", and "macOS". The "macOS" section is expanded, showing "Build a macOS app" and "C interop" (which is highlighted in blue). Other collapsed sections include "Packages & plugins" and "Testing & debugging". The main content area has a large title "Binding to native macOS code using dart:ffi". Below it, a breadcrumb trail shows "Platform integration > macOS > Binding to native macOS code using dart:ffi". A note in a light blue box states: "Note: This page describes using the `dart:ffi` library in macOS desktop apps. For information on Android, see [Binding to native Android code using dart:ffi](#). For information on iOS, see [Binding to native iOS code using dart:ffi](#). This feature is not yet supported for web plugins." The main text explains that before using the FFI library, native code must be loaded and its symbols made visible to Dart. It then describes a tutorial for bundling C/C++ sources in a Flutter plugin and binding them using the Dart FFI library on macOS.

Flutter

Multi-Platform ▾ Development ▾ Ecosystem ▾ Showcase Docs ▾ [Get started](#)

Data & backend ▾

Accessibility & localization ▾

Platform integration ▾

- Supported platforms
- Build desktop apps with Flutter
- Write platform-specific code

▶ Android

▶ iOS

▶ Linux

macOS

- Build a macOS app
- C interop**

▶ Web

▶ Windows

Packages & plugins ▾

Testing & debugging ▾

Binding to native macOS code using dart:ffi

Platform integration > macOS > Binding to native macOS code using dart:ffi

Flutter mobile and desktop apps can use the `dart:ffi` library to call native C APIs. *FFI* stands for *foreign function interface*. Other terms for similar functionality include *native interface* and *language bindings*.

Note: This page describes using the `dart:ffi` library in macOS desktop apps. For information on Android, see [Binding to native Android code using dart:ffi](#). For information on iOS, see [Binding to native iOS code using dart:ffi](#). This feature is not yet supported for web plugins.

Before your library or program can use the FFI library to bind to native code, you must ensure that the native code is loaded and its symbols are visible to Dart. This page focuses on compiling, packaging, and loading macOS native code within a Flutter plugin or app.

This tutorial demonstrates how to bundle C/C++ sources in a Flutter plugin and bind to them using the Dart FFI library on macOS. In this walkthrough, you'll create a C function that implements 32-bit addition and then exposes it through a Dart plugin named "native_add".

Contents

- Dynamic vs static linking
- Step 1: Create a plugin
- Step 2: Add C/C++ sources
- Step 3: Load the code using the FFI library

Other use cases

- iOS and macOS
- Platform library
- First-party library
- Source code
- Compiled (dynamic) library
- Compiled (dynamic) library (macOS)

- <https://docs.flutter.dev/platform-integration/windows/building>

The screenshot shows the Flutter documentation website. The top navigation bar includes links for Multi-Platform, Development, Ecosystem, Showcase, Docs, a search bar, and social media icons. A blue "Get started" button is on the right. On the left, there's a sidebar with a tree view of documentation categories like routing, Data & backend, Accessibility & localization, Platform integration (which is expanded to show Supported platforms, Build desktop apps with Flutter, Write platform-specific code, and sub-sections for Android, iOS, Linux, macOS, Web, and Windows), Packages & plugins, and Testing & debugging. The main content area features a large title "Building Windows apps with Flutter" and a subtitle "Integrating with Windows". Below the titles, there's a paragraph about the Windows programming interface and a note about calling C libraries using Dart's FFI library. To the right of the main content, there's a sidebar titled "Contents" with links to various Windows-related topics.

Flutter

Multi-Platform ▾ Development ▾ Ecosystem ▾ Showcase Docs ▾ [🔍](#) [Twitter icon](#) [YouTube icon](#) [Medium icon](#) [GitHub icon](#) [Get started](#)

routing ▾

Data & backend ▾

Accessibility & localization ▾

Platform integration ▾

- Supported platforms
- Build desktop apps with Flutter
- Write platform-specific code
- ▶ Android
- ▶ iOS
- ▶ Linux
- ▶ macOS
- ▶ Web
- ▼ Windows
- [Build a Windows app](#)

Packages & plugins ▾

Testing & debugging ▾

Building Windows apps with Flutter

Platform integration > Windows > Windows development

This page discusses considerations unique to building Windows apps with Flutter, including shell integration and distribution of Windows apps through the Microsoft Store on Windows.

Integrating with Windows

The Windows programming interface combines traditional Win32 APIs, COM interfaces and more modern Windows Runtime libraries. As all these provide a C-based ABI, you can call into the services provided by the operating system using Dart's Foreign Function Interface library ([dart:ffi](#)). FFI is designed to enable Dart programs to efficiently call into C libraries. It provides Flutter apps with the ability to allocate native memory with `malloc` or `calloc`, support for pointers, structs and callbacks, and ABI types like `long` and `size_t`.

For more information about calling C libraries from Flutter, see [C interop using dart:ffi](#).

Contents

- [Integrating with Windows](#)
- [Supporting Windows UI guidelines](#)
- [Customizing the Windows host application](#)
- [Compiling with Visual Studio](#)
- [Distributing Windows apps](#)
- [MSIX packaging](#)
- [Create a self-signed .pfx certificate for local testing](#)
- [Building your own zip file for Windows](#)

- <https://docs.flutter.dev/platform-integration/linux/building>

The screenshot shows the Flutter documentation website. The top navigation bar includes links for Multi-Platform, Development, Ecosystem, Showcase, Docs, a search icon, social media icons for Twitter, YouTube, and GitHub, and a 'Get started' button. On the left, a sidebar menu is open under the 'Platform integration' section, showing options like Supported platforms, Build desktop apps with Flutter, Write platform-specific code, and detailed sections for Android, iOS, and Linux. The main content area features a large title 'Building Linux apps with Flutter' with a subtitle 'Platform integration > Linux > Linux development'. Below the title, a paragraph explains the page's focus on unique Linux considerations like shell integration and distribution preparation. The main article is titled 'Integrating with Linux', which discusses the Linux programming interface, C interop via dart:ffi, and various Dart packages for Linux integration.

Building Linux apps with Flutter

Platform integration > Linux > Linux development

This page discusses considerations unique to building Linux apps with Flutter, including shell integration and preparation of apps for distribution.

Integrating with Linux

The Linux programming interface, comprising library functions and system calls, is designed around the C language and ABI. Fortunately, Dart provides `dart:ffi`, which is designed to enable Dart programs to efficiently call into C libraries. FFI provides Flutter apps with the ability to allocate native memory with `malloc` or `calloc`, support for pointers, structs and callbacks, and ABI types like `long` and `size_t`.

For more information about calling C libraries from Flutter, see [C interop using dart:ffi](#).

Many apps will benefit from using a package that wraps the underlying library calls in a more convenient, idiomatic Dart API. [Canonical has built a series of packages](#) with a focus on enabling Dart and Flutter on Linux, including support for desktop notifications, dbus, network management, and Bluetooth.

More generally, many other [packages support Linux](#), including common packages such as `url_launcher`,

FFI로 C 언어 연결하기 (리뷰 & 실습)

- 교재에서 설명한 환경 설정 등을 수행
- volume-I-chapter-07의 sum_ffi.c를 sum_ffi.dart에서 호출

```

1 import 'dart:ffi';
2 import 'dart:io';
3
4 import 'package:path/path.dart' as path;
5
6 typedef SumFunc = Int32 Function(Int32 a, Int32 b);
7 typedef Sum = int Function(int a, int b);
8
9 Run | Debug
10 void main() {
11   // Open the dynamic library
12   var libraryPath =
13     path.join(Directory.current.path, 'sum_ffi_library', 'libsum_ffi.so');
14   if (Platform.isMacOS) {
15     libraryPath = path.join(
16       Directory.current.path, 'sum_ffi_library', 'libsum_ffi.dylib');
17   }
18   if (Platform.isWindows) {
19     libraryPath = path.join(
20       Directory.current.path, 'sum_ffi_library', 'Debug', 'sum_ffi.dll');
21   }
22   final dylib = DynamicLibrary.open(libraryPath);
23
24   // calls int sum(int a, int b);
25   final sumPointer = dylib.lookup<NativeFunction<SumFunc>>('sum');
26   final sum = sumPointer.asFunction<Sum>();
27   print('3 + 5 = ${sum(3, 5)}');
28 }
```

```

1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <stdarg.h>
4 #include "sum_ffi.h"
5
6 int main()
7 {
8   printf("3 + 5 = %d\n", sum(3, 5));
9   return 0;
10 }
11
12 int sum(int a, int b)
13 {
14   return a + b;
15 }
```

참조: Platform Specific API 호출

- FFI와 별개로, Flutter는 플랫폼 API 호출을 지원함
 - Kotlin or Java on Android
 - Swift or Objective-C on iOS
 - C++ on Windows
 - Objective-C on macOS
 - C on Linux

참조: Platform Specific API 호출

- <https://docs.flutter.dev/platform-integration/platform-channels?tab=type-mappings-java-tab>

The screenshot shows the Flutter documentation website. The top navigation bar includes links for Multi-Platform, Development, Ecosystem, Showcase, Docs, and a search bar. Below the navigation is a main content area with a sidebar on the left containing a navigation tree for 'Platform integration'. The main content title is 'Writing custom platform-specific code', with a subtitle 'Platform integration > Platform-specific code'. A note states: 'This guide describes how to write custom platform-specific code. Some platform-specific functionality is available through existing packages; see [using packages](#)'. A callout box contains a note: 'Note: The information in this page is valid for most platforms, but platform-specific code for the web generally uses [JS interoperability](#) or the [dart:html library](#) instead.' The sidebar also lists other sections like 'Contents', 'Architectural overview: platform channels', 'Platform channel data types support and codecs', 'Example: Calling platform-specific code using platform channels', and numbered steps for creating a new app project. At the bottom right, there's a link to 'Typesafe platform channels'.

수업에서 다루지 않은 교재 속 이야기들

- **Volume.H 지속 가능한 개발자로 첫걸음 내딛기**

01. Dart for Embedded 알아 두기
03. Flutter 공식 사이트 레퍼런스와 샘플 활용하기
04. Flutter CLI 명령 이해하기
05. Dart와 Flutter 최신 정보 신청하기

수업에서 다루지 않은 교재 속 이야기들

- **Volume.I 알아 두면 요긴한 분야별 노하우**

03. Docker를 이용한 컨테이너 기반 서비스 개발하기 서버

04. 서로 다른 디바이스 간에 통신하기 네트워크

05. WebSocket 기반 네트워킹 기능 개발하기 네트워크

08. 라즈베리 파이에서 Dart와 Flutter 활용하기

09. XD2Flutter로 디자인 개선하기 앱

10. Flutter 앱 배포하기 앱

11. Concurrency 기반 병렬처리 개발하기 성능

12. 컴퓨터공학 전문 이론 공부하기 성능

13. TensorFlow 활용하기 인공지능

14. Flutter 클라우드 개발 환경 활용하기 도구

수업에서 다루지 않은 교재 속 이야기들

● Docker를 이용한 컨테이너 기반 서비스 개발하기 서버

- ▣ https://hub.docker.com/search?q=Dart&image_filter=official

The screenshot shows the Docker Hub search interface. The search bar at the top contains the query 'Dart'. Below the search bar, there are several filter categories on the left: 'Products' (Images, Extensions, Plugins), 'Trusted Content' (Docker Official Image checked, Verified Publisher, Sponsored OSS), 'Operating Systems' (Linux, Windows), and 'Architectures' (ARM, ARM 64, IBM POWER, IBM Z, PowerPC 64 LE, x86, x86-64). The main results area displays one result: 'Dart - Docker Official Image'. The card for this image includes the Docker logo, the name 'dart', the status 'DOCKER OFFICIAL IMAGE', download statistics ('1M+'), star count ('112'), and a note that it was 'Updated 2 days ago'. A description below the card states 'Dart is a client-optimized language for fast apps on any platform.' At the bottom of the card, there are links for 'Linux', 'ARM 64', 'x86-64', and 'ARM'. To the right of the card, there is a chart showing 'Pulls: 7,918 Last week' with a line graph. A 'Learn more' link is also present.

수업에서 다루지 않은 교재 속 이야기들

● Docker를 이용한 컨테이너 기반 서비스 개발하기 서버

- ▣ https://hub.docker.com/search?q=Dart&image_filter=official

The screenshot shows the Docker Hub search interface. The search bar at the top contains 'Search Docker Hub'. Below it, the search results for 'dart' are displayed. The first result is 'dart' (Docker Official Image), which has over 1M+ pulls and 112 stars. The description states: 'Dart is a client-optimized language for fast apps on any platform.' To the right of the image, there is a button labeled 'docker pull dart' with a copy icon. The page also includes sections for 'Quick reference', 'Supported tags and respective Dockerfile links', and 'About Official Images'.

Docker Hub search results for 'dart':

- dart** (Docker Official Image) - 1M+ pulls, 112 stars. Dart is a client-optimized language for fast apps on any platform.

Quick reference

- Maintained by: The Dart Docker Team
- Where to get help: the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

Supported tags and respective Dockerfile links

- 3.0.5-sdk, 3.0-sdk, 3-sdk, stable-sdk, sdk, 3.0.5, 3.0, 3, stable, latest
- 3.1.0-163.1.beta-sdk, beta-sdk, 3.1.0-163.1.beta, beta

Quick reference (cont.)

Recent Tags

stable-sdk, stable, sdk, latest, beta-sdk, beta
3.1.0-163.1.beta-sdk, 3.1.0-163.1.beta, 3.0.5-sdk, 3.0.5

About Official Images

Docker Official Images are a curated set of Docker open source and drop-in solution repositories.

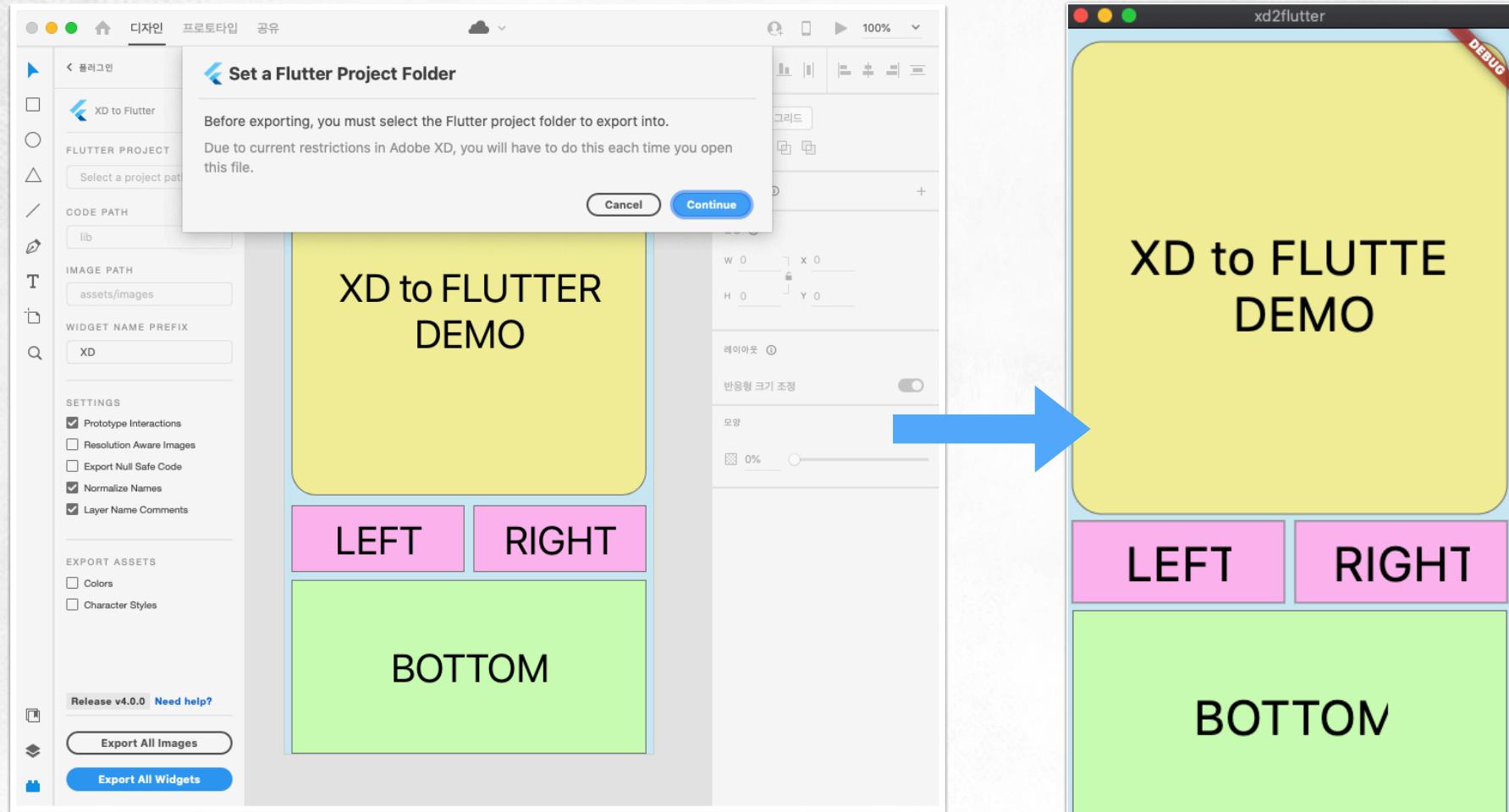
Why Official Images?

These images have clear documentation, promote best practices, and are designed for the most common use cases.

수업에서 다루지 않은 교재 속 이야기들

● **XD2Flutter로 디자인 개선하기 앱**

- ◆ AdobeXD에서 작성한 디자인을 Flutter 코드로 자동 변환



Wrapup

Enjoy Dart & Flutter Life





Thank you