

## Verifying the Dart SDK installation

After you've installed Dart, run the following command in a terminal to ensure that it's working:

```
dart --version
```

You should see the current Dart version displayed, which at the time of this writing was **2.12.2**.

## Contents of the SDK

Now check out what the Dart SDK offers you by entering the following command in the terminal:

```
dart help
```

You'll see a list of tools that make up the SDK. Although you won't directly interact with most of them in this book, it's helpful to know what they do:

- **analyze:** Your IDE uses this tool to tell you when you've made a mistake in your code. The sooner you know, the sooner you can fix it!
- **compile:** This tool compiles Dart code into an optimized native executable program for Windows, Linux or macOS. This is known as ahead-of-time, or **AOT**, compilation. Alongside native executables, web technologies are another major focus for Dart, so you can also use the **compile** tool to convert Dart code to JavaScript.
- **create:** This is for creating new Dart projects, which you'll do yourself in just a minute.
- **fix:** One of Dart's goals is to continue evolving as a language without becoming bloated by obsolete, or **deprecated**, code. The fix tool is there to help developers update their old projects to use the shiniest new Dart syntax.
- **format:** It's easy for the indentation in your code to get messed up. This nice little tool will automatically fix it for you.
- **migrate:** Version 2.12 was a major update to the Dart language with the addition of sound null safety, which you'll learn about in Chapter 7. This tool helps migrate old projects to use null safety. Since you're starting fresh, though, you won't need to migrate anything. Lucky you!

- **pub:** Pub is the name of the **package manager** for Dart, and pub is the tool that handles the job. A **package** is a collection of third-party code that you can use in your own Dart project. This can save you an incredible amount of time since you don't have to write that code yourself. You can browse the packages available to you on Pub by visiting [pub.dev](https://pub.dev).
- **run:** This runs your Dart program in the Dart **Virtual Machine**, or **VM**. The Dart VM compiles your code right before it's needed. In contrast to AOT, this is known as just-in-time, or **JIT**, compilation, which will let you make small changes to your code and rerun it almost instantly. This is especially useful for applications like Flutter where you'll need to make lots of little changes as you refine the UI.
- **test:** Dart fully supports unit testing and this tool will help you get that done.

## Dart on the command line

Now that you have the Dart SDK installed, you're going to use the Dart VM to run a few lines of code, first in a single file and then as a full project.

### Running a single Dart file

Find or create a convenient folder on your computer where you can save the Dart projects that you create in this book. Create a new file in that folder and name it **hello.dart**

## Writing the code

Next add the following Dart code to that empty file:

```
void main() {  
  print('Hello, Dart!');  
}
```

This creates a Dart function named `main`. Inside that function, you call another function, `print`, which displays the text `Hello, Dart!` on the screen.

## Running the code

Save the file, and then run the following command in the same folder as `hello.dart`

```
dart run hello.dart
```

The `run` keyword is the run tool from the Dart SDK that you learned about earlier. It runs the code in `hello.dart` in the Dart VM.

You should now see the following output in the console:

```
Hello, Dart!
```

Congratulations! You've built and run your first Dart program.

## Setting up a full Dart project

It's nice to be able to run a single file, but as you build bigger projects, you'll want to divide your code into manageable pieces and also include configuration and asset files. To do that you need to create a full Dart project. Remember that create tool? The time has come.

## Creating the project

Go to the location where you want to create your project folder, and then run the following command in the terminal:

```
dart create hello_dart_project
```

This creates a simple Dart project with some default code.

## Running the project

Enter the new folder you just created like so:

```
cd hello_dart_project
```

Now run the project with the following command:

```
dart run bin/hello_dart_project.dart
```

You'll see the text **Hello world!**, which is the output of the code in the default project that the create tool generated.

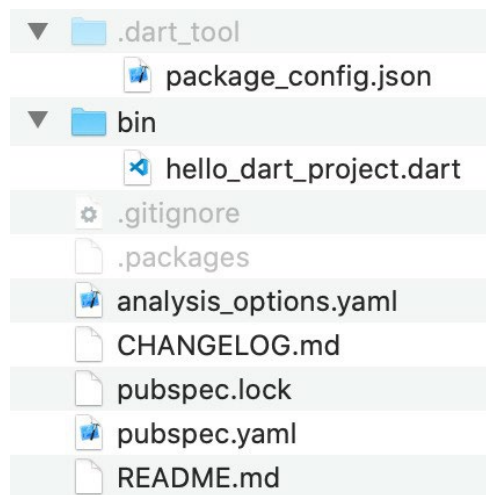
The run keyword is actually optional. Run the project again without it:

```
dart bin/hello_dart_project.dart
```

Again, **Hello world!** is the result.

## The structure of a Dart project

Take a look at the structure and contents of the **hello\_dart\_project** folder:



The purposes of the main items in that folder are as follows:

- **bin**: Contains the executable Dart code.
- **hello\_dart\_project.dart** : Named the same as the project folder, the create tool generated this file for you to put your Dart code in.
- **.gitignore**: Formatted to exclude Dart-related files that you don't need if you're going to host your project on GitHub or another Git repository.

- **analysis\_options.yaml**: Holds special rules that will help you detect issues with your code, a process known as **linting**.
- **CHANGELOG.md**: Holds a manually-curated Markdown-formatted list of the latest updates to your project. Whenever you release a new version of a Dart project, you should let other developers know what you've changed.
- **README.md**: Provides a basic (or not-so-basic) description of what your project does and how to use it. Other developers will appreciate this greatly.
- **pubspec.yaml**: Contains a list of the third-party Pub dependencies you want to use in your project. The name "pubspec" stands for "Pub specifications". You also set the version number of your project in this file.

**Note:** **YAML** stands for "YAML Ain't Markup Language", one of those recursive acronyms that computer programmers like to amuse themselves with. YAML is a clean and readable way to format configuration files, and you'll come across this file type often in your Dart career.

## Simple vs. full console app

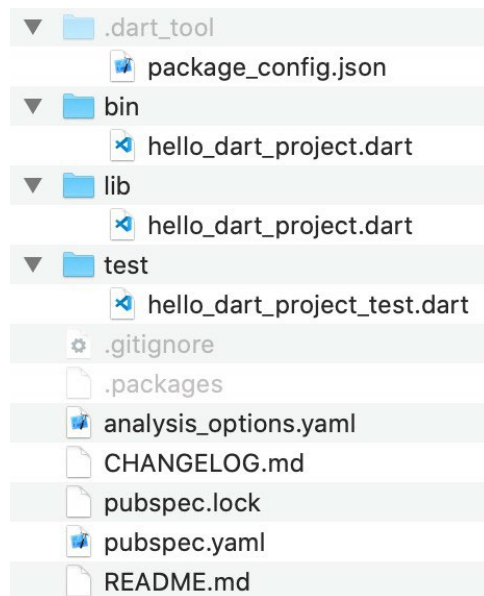
When you created your project using the **create** tool above, it created a simple console app because that's the default.

However, you can create other types of projects using the **--template** option.

For example, if you'd used the following command to create your project:

```
dart create --template console-full hello_dart_project
```

It would have given you the following directories and files:



There are two additional directories: **lib** and **test**. In larger projects, you'll have many **.dart** files that you'll organize under the **lib** folder. You'll also likely want to have tests to run against your Dart projects, and you can place those in the **test** folder.

For your work in this book, creating simple console projects will be enough. If you wish to create full console projects,



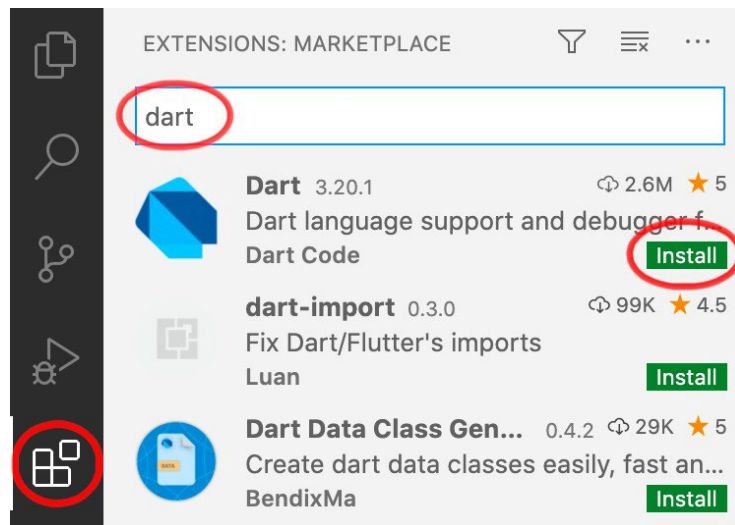
though, it won't make a difference to your progress through this book.

## Using VS Code for Dart development

You've created and run a project from the command line, but it's also possible to do the same thing from within VS Code. This section will walk you through that process.

### Installing the Dart extension

Open Visual Studio Code, and on the left hand side you'll see a vertical toolbar called **Activity Bar**. Click the **Extensions icon**, which looks like four boxes. Then typed **dart** in the search area. When the Dart extension appears, click the **Install button** to install it.



Now your VS Code installation supports Dart. Next you'll learn how to create a Dart project in VS Code.

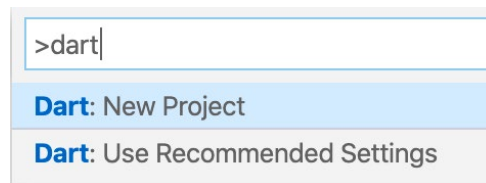
### Creating a new Dart project

The Dart extension in VS Code makes it easy to create a new Dart project. To see how this works, you'll recreate the same simple console project that you previously created from the command line.

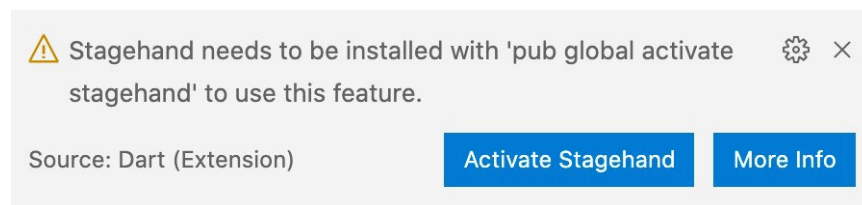
To start, delete the **hello dart\_project** folder and its contents.

You can create a new project from the **Command Palette**. To access the Command Palette, either go to **View Command Palette...** in the menu, or press the shortcut **Command+Shift+P** on a Mac or **Control+Shift+P** on a PC.

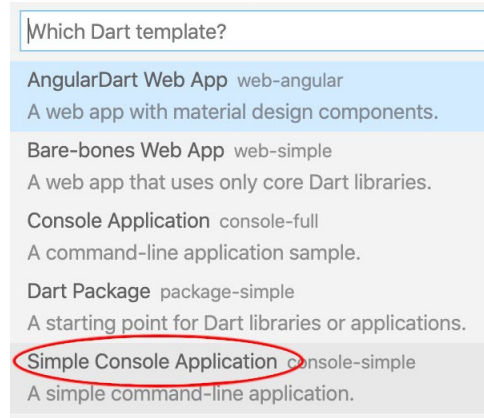
Start typing **dart** to bring up a list of matching commands. Then choose **Dart: New Project**.



If you receive a message about Stagehand, then choose **Activate Stagehand**.



Next, choose **Simple Console Application** from the list.



As before, choose a location to save the project folder that VS Code will create, and name the project **hello dart\_project**

## Browsing the generated code

Open the file **hello dart\_project.dart** in the **bin** directory, and you'll see it contains the following code:

```
void main(List<String> arguments) {  
  print('Hello world!');  
}
```

The `List<String> arguments` portion is only necessary when creating command-line apps that take arguments. For example, take a look at the following imaginary terminal command that prints information about a country:

```
lookup -n spain
```

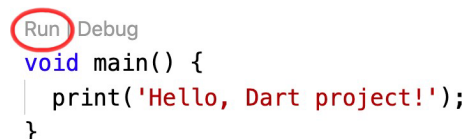
The command-line app name is **lookup** and the arguments are **-n spain**.

Since you won't be creating command-line apps in this book, you can remove the arguments portion to simplify things. Thus, replace the contents of `hello_dart_project.dart` with the following code:

```
void main() {  
  print('Hello, Dart project!');  
}
```

## Running Dart in VS Code

To run your code, click the word **Run** that appears directly over the `main` function.

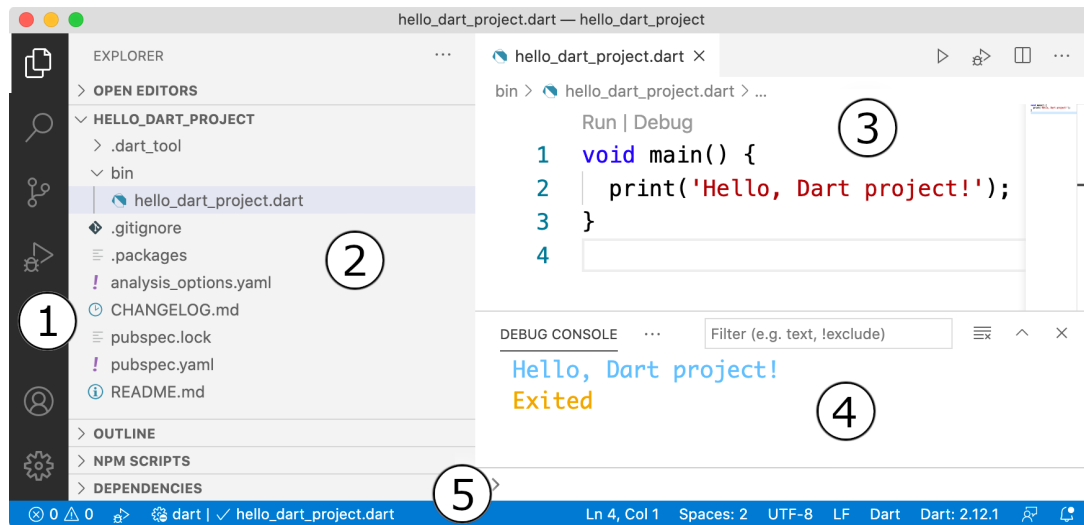


A screenshot of the Visual Studio Code editor showing a Dart file. The code is: `void main() {  
 print('Hello, Dart project!');  
}`. Above the `main` function, there are two buttons: 'Run' and 'Debug'. The 'Run' button is circled in red.

You'll see `Hello, Dart project!` appear in the debug console.

## Exploring the VS Code UI

This is a good opportunity to explore the various parts of the Visual Studio Code user interface.



The numbers below correspond to the various areas of the user interface:

1. **Activity Bar:** Choose which content to show in the side bar.
2. **Side Bar:** The Explorer is displaying the current project and file.
3. **Editor:** Write your Dart code here.
4. **Panels:** Show program output, run terminal commands, and more.
5. **Status Bar:** Display information about the current project.

## More ways to run your project

You ran your project earlier by pressing the **Run** label over the `main` function. Here are three more ways that you can run your project:

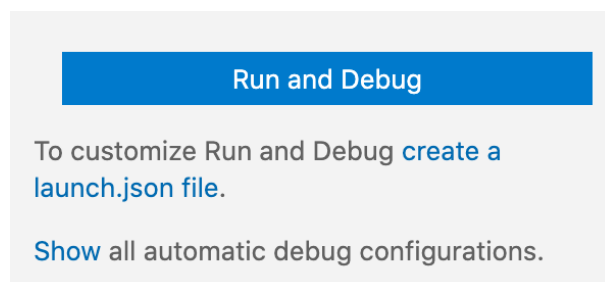
1. Choose **Run ▶ Start Debugging** from the menu.
2. Press **F5**.
3. Click the **triangular run button** in the top right corner.



All of these do the same thing. This time use **F5** to run the program, and you'll see **Hello, Dart project!** appear in the debug console again.

## Project configuration file

Sometimes when pressing F5 to run your project, VS Code doesn't know where to look for the `main` function, and you'll see a message like this:

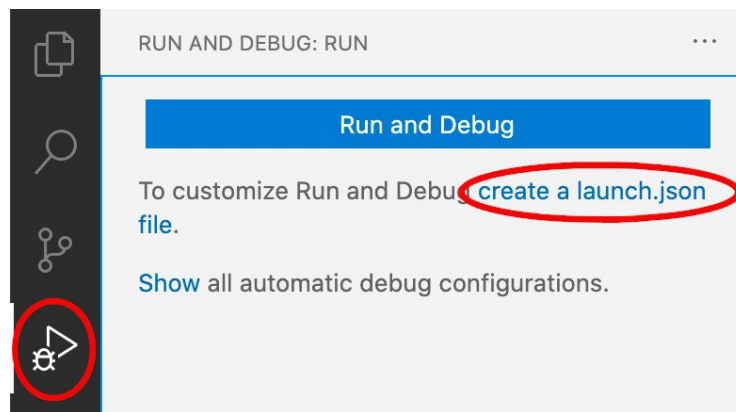


This is recommending that you create a `launch.json`, which will help VS Code know which file to use to launch your app.

Since you originally ran your program by pressing the **Run** label, you may not have gotten this recommendation from VS Code. However, it's still a good idea to create a `launch.json` file, so you'll do that next.

## Creating launch.json

Create the `launch.json` file by clicking the **Debug** icon in the **Activity Bar** and then clicking the link to **create a launch.json file** as shown in the image below.



This will create a new `launch.json` file in the `.vscode` folder.

## Updating the contents

Now replace the contents of `launch.json` with the following:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Dart",
      "type": "dart",
      "request": "launch",
      "program": "bin/hello_dart_project.dart"
    }
  ]
}
```

These are what the configuration elements mean:

- **name:** This is your project's configuration name. You can call it whatever you like.
- **type:** The type lets VS Code know that this is a Dart project.
- **request:** A request of `launch` tells VS Code that you want to run the project.
- **program:** This is the location where your program will start its execution. VS Code will look for the `main` function here so that it can launch your app.

## Running your project again

Save your changes and run the file again by pressing **F5**. VS Code should be satisfied now that it knows where to look for the `main` function.

Excellent! You're all set to explore Dart further in the rest of this book.

## Key points

- **Visual Studio Code** is an **Integrated Development Environment** that you can use to write Dart code when you have the Dart extension installed.
- The **Dart SDK** provides the underlying tools needed to compile and run Dart apps.



- Dart code run from the command line or in VS Code uses the **Dart Virtual Machine**
- The VS Code window is divided into the **Activity Bar**, **Side Bar** **Editor Panel** and **Status Bar**.
- **Pub** is the package manager that Dart uses to add third- party source code to your projects.