Dart Programming

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Dart is an open-source general-purpose programming language. It is originally developed by Google and later approved as a standard by ECMA.

Dart is a new programming language meant for the server as well as the browser. Introduced by Google, the **Dart SDK** ships with its compiler – the **Dart VM**. The SDK also includes a utility **-dart2js**, a transpiler that generates JavaScript equivalent of a Dart Script.

This tutorial provides a basic level understanding of the Dart programming language.

Audience

This tutorial will be quite helpful for all those developers who want to develop single-page web applications using Dart. It is meant for programmers with a strong hold on object-oriented concepts.

Prerequisites

The tutorial assumes that the readers have adequate exposure to object-oriented programming concepts. If you have worked on JavaScript, then it will help you further to grasp the concepts of Dart quickly.

Copyright & Disclaimer

Table of Contents

1. Dart – Overview

Dart is an object-oriented language with C-style syntax which can optionally trans compile into JavaScript. It supports a varied range of programming aids like interfaces, classes, collections, generics, and optional typing.

Dart can be extensively used to create single-page applications. Single-page applications apply only to websites and web applications. Single-page applications enable navigation between different screens of the website without loading a different webpage in the browser. A classic example is **GMail** — when you click on a message in your inbox, browser stays on the same webpage, but JavaScript code hides the inbox and brings the message body on screen.

Google has released a special build of **Chromium** – the **Dart VM**. Using Dartium means you don't have to compile your code to JavaScript until you're ready to test on other browsers.

The following table compares the features of Dart and JavaScript.

Feature	Dart	JavaScript
Type system	Optional, dynamic	Weak, dynamic
Classes	Yes, single inheritance	Prototypical
Interfaces	Yes, multiple interfaces	No
Concurrency	Yes, with isolates	Yes, with HTML5 web workers

This tutorial provides a basic level understanding of the Dart programming language.

2. Dart – Environment

This chapter discusses setting up the execution environment for Dart on the Windows platform.

Executing Script Online with DartPad

You may test your scripts online by using the online editor at https://dartpad.dartlang.org/. The Dart Editor executes the script and displays both HTML as well as console output. The online editor is shipped with a set of preset code samples.

A screenshot of the **Dartpad** editor is given below:

Dartpad also enables to code in a more restrictive fashion. This can be achieved by checking the Strong mode option on the bottom right of the editor. Strong mode helps with:

- Stronger static and dynamic checking
- Idiomatic JavaScript code generation for better interoperability.

You may try the following example using Dartpad

```
void main() {
 print('hello world');
}
```

The code will display the following output

```
hello world
```

Setting Up the Local Environment

In this section, let us see how to set up the local environment.

Using the Text Editor

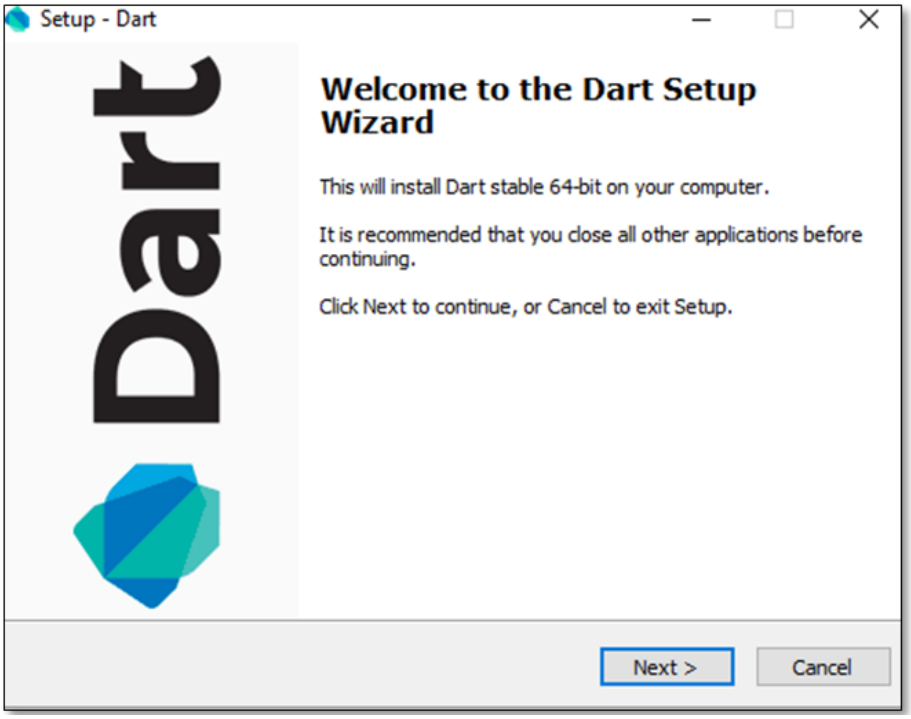
Examples of a few editors include Windows Notepad, Notepad++, Emacs, vim or vi, etc. Editors may vary from one Operating System to another. The source files are typically named with the extension ".dart".

Installing the Dart SDK

The current stable version of Dart is **1.21.0**. The **dart sdk** can be downloaded from:

- https://www.dartlang.org/install/archive
- http://www.gekorm.com/dart-windows/

A screenshot of the Dart SDK installation is given below:



On completion of the SDK installation, set the PATH environment variable to:

```
<dart-sdk-path>\bin
```

Verifying the Installation

To verify if Dart has been successfully installed, open the command prompt and enter the following command:

```
Dart
```

If installation is successful, it will show the dart runtime.

IDE Support

A plethora of IDEs support scripting in Dart. Examples include **Eclipse**, **IntelliJ**, and **WebStorm** from Jet brains.

Given below are the steps for configuring the Dart environment using **WebStrom IDE**.

Installing WebStorm

The installation file for **WebStorm** can be downloaded from **http://www.jetbrains.com/webstorm/download/#section=windows-version**.

The WebStorm installation file is available for Mac OS, Windows and Linux.

After downloading the installation files, follow the steps given below:

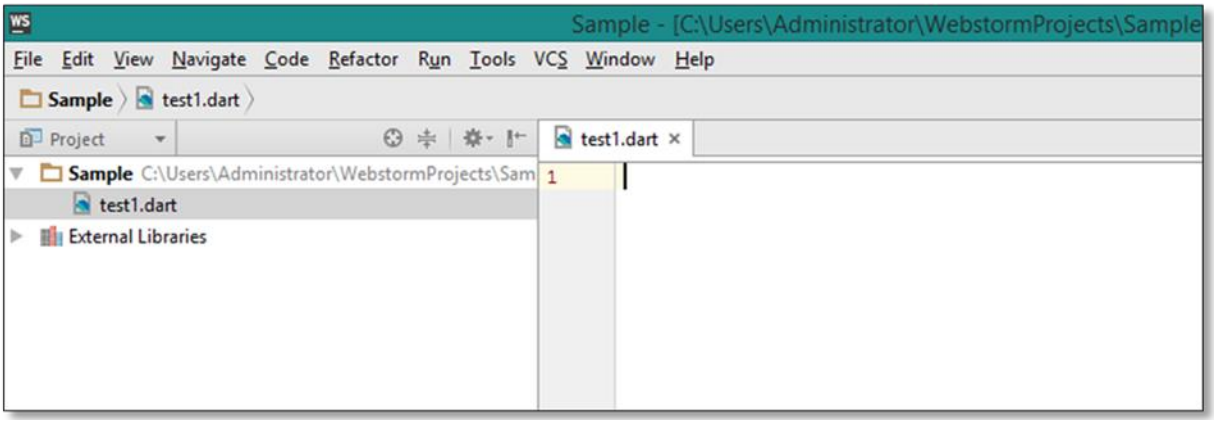
- Install the Dart SDK: Refer to the steps listed above
- Create a new Dart project and configure Dart support
- To create a new Dart project,
    - Click **Create New Project** from the Welcome Screen
    - In the next dialog box, click **Dart**
- If there is no value specified for the **Dart SDK** path, then provide the SDK path. For example, the SDK path may be ***<dart installation directory>*/dart/dart-sdk**.

Add a Dart File to the Project

To add a Dart file to the Project:

- Right-click on the Project
- New → Dart File
- Enter the name of the Dart Script

A screenshot of the WebStorm Editor is given below:



The dart2js Tool

The **dart2js** tool compiles Dart code to JavaScript. Compiling Dart code to JS enables running the Dart script on browsers that do not support the Dart VM.

The dart2js tool is shipped as a part of the Dart SDK and can be found in the **/dart-sdk/bin** folder.

To compile Dart to JavaScript, type the following command in the terminal

```
dart2js - - out=<output_file>.js  <dart_script>.dart
```

This command produces a file that contains the JavaScript equivalent of your Dart code. A complete tutorial on using this utility can be found on the official Dart website.

3. Dart – Syntax

Syntax defines a set of rules for writing programs. Every language specification defines its own syntax. A Dart program is composed of:

- Variables and Operators
- Classes
- Functions
- Expressions and Programming Constructs
- Decision Making and Looping Constructs
- Comments
- Libraries and Packages
- Typedefs
- Data structures represented as Collections / Generics

Your First Dart Code

Let us start with the traditional "Hello World" example –

```
main()
{
      print("Hello World!");
}
```

The **main()** function is a predefined method in Dart. This method acts as the entry point to the application. A Dart script needs the **main()** method for execution. **print()** is a predefined function that prints the specified string or value to the standard output i.e. the terminal.

The output of the above code will be:

```
Hello World!
```

Execute a Dart Program

You can execute a Dart program in two ways:

- Via the terminal

- Via the WebStorm IDE

Via the Terminal

To execute a Dart program via the terminal:

- Navigate to the path of the current project
- Type the following command in the Terminal window

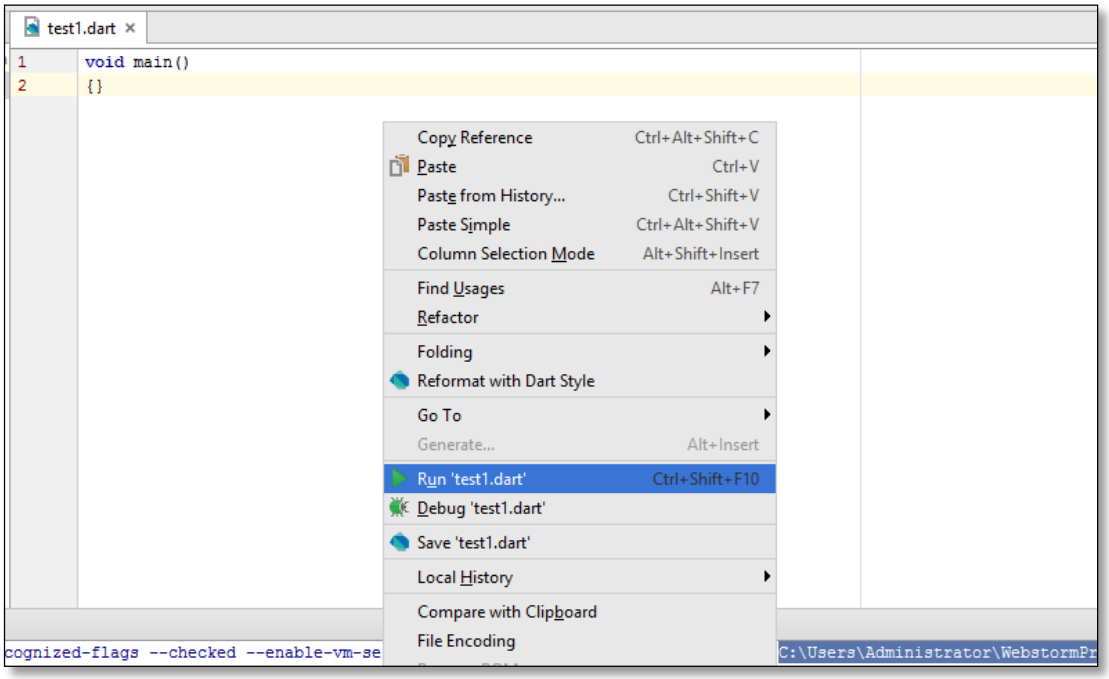
```
dart file_name.dart
```

Via the WebStorm IDE

To execute a Dart program via the WebStorm IDE:

- Right-click the Dart script file on the IDE. (The file should contain the **main()** function to enable execution)

- Click on the '**Run <file_name>**' option. A screenshot of the same is given below:



One can alternatively click the  button or use the shortcut **Ctrl+Shift+F10** to execute the Dart Script.

Dart Command-Line Options

Dart command-line options are used to modify Dart Script execution. Common command-line options for Dart include the following:

Command-Line Option	Description
-c or --c	Enables both assertions and type checks (checked mode).
--version	Displays VM version information.
--packages <path>	Specifies the path to the package resolution configuration file.
-p <path>	Specifies where to find imported libraries. This option cannot be used with --packages.
-h or --help	Displays help

Enabling Checked Mode

Dart programs run in two modes namely-

- Checked Mode
- Production Mode (Default)

It is recommended to run the Dart VM in **checked mode** during development and testing, since it adds warnings and errors to aid development and debugging process. The checked mode enforces various checks like type-checking etc. To turn on the checked mode, add the -c or –-checked option before the script-file name while running the script.

However, to ensure performance benefit while running the script, it is recommended to run the script in the **production mode**.

Consider the following **Test.dart** script file:

```
void main()
{
  int n="hello";
  print(n);
}
```

Run the script by entering:

```
dart Test.dart
```

Though there is a type-mismatch the script executes successfully as the checked mode is turned off. The script will result in the following output

```
hello
```

Now try executing the script with the "- - checked" or the "-c" option:

```
dart -c Test.dart
```

Or,

```
dart - - checked Test.dart
```

The Dart VM will throw an error stating that there is a type mismatch.

```
Unhandled exception:
type 'String' is not a subtype of type 'int' of 'n' where
  String is from dart:core
  int is from dart:core
#0      main (file:///C:/Users/Administrator/Desktop/test.dart:3:9)
#1      _startIsolate.<anonymous closure> (dart:isolate-patch/isolate_patch.dart :261)
#2      _RawReceivePortImpl._handleMessage (dart:isolate-patch/isolate_patch.dart:148)
```

Identifiers in Dart

Identifiers are names given to elements in a program like variables, functions etc. The rules for identifiers are –

Identifiers can include both, characters and digits. However, the identifier cannot begin with a digit.

- Identifiers cannot include special symbols except for underscore (_) or a dollar sign ($).
- Identifiers cannot be keywords.
- They must be unique.
- Identifiers are case-sensitive.
- Identifiers cannot contain spaces.

The following tables lists a few examples of valid and invalid identifiers –

Valid identifiers	Invalid identifiers
firstName	Var
first_name	first name
num1	first-name
$result	1number

Keywords in Dart

Keywords have a special meaning in the context of a language. The following table lists some keywords in Dart.

abstract 1	continue	false	new	this
as 1	default	final	null	throw
assert	deferred 1	finally	operator 1	true
async 2	do	for	part 1	try
async* 2	dynamic 1	get 1	rethrow	typedef 1
await 2	else	if	return	var
break	enum	implements 1	set 1	void
case	export 1	import 1	static 1	while
catch	external 1	in	super	with
class	extends	is	switch	yield 2
const	factory 1	library 1	sync* 2	yield* 2

Whitespace and Line Breaks

Dart ignores spaces, tabs, and newlines that appear in programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Dart is Case-sensitive

Dart is case-sensitive. This means that Dart differentiates between uppercase and lowercase characters.

Statements end with a Semicolon

Each line of instruction is called a statement. Each dart statement must end with a semicolon (;). A single line can contain multiple statements. However, these statements must be separated by a semicolon.

End of ebook preview
If you liked what you saw…
Buy it from our store @ **https://store.tutorialspoint.com**