Introduction to Dart Language

RDT Training Module

RZ 08-2

Table of Contents

Module #01 : Libraries

Overview: Understanding how to organize Dart code.

1.1 Organizing Dart Code

Dart provides three keywords for code organizing.

1. #library

   This keyword defines a library. The library can be located anywhere accessible by a uri.

   Example: #library("myLibrary");
2. #import

   This keyword will make other libraries available to the current code.

   Example: #import("myLibrary.dart");
   #import("lib/myLibrary.dart");
   #import("../myLibrary.dart");
   #import("http://server.com/myLibrary.dart");

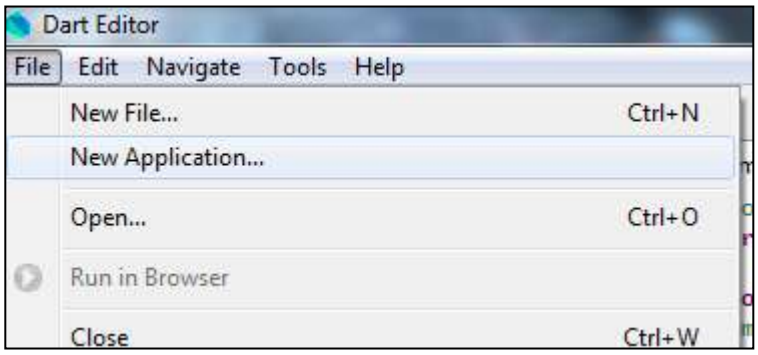
   Note: The http:// format will only work for client side code.
3. #source

   This keyword will include source files in the current library.

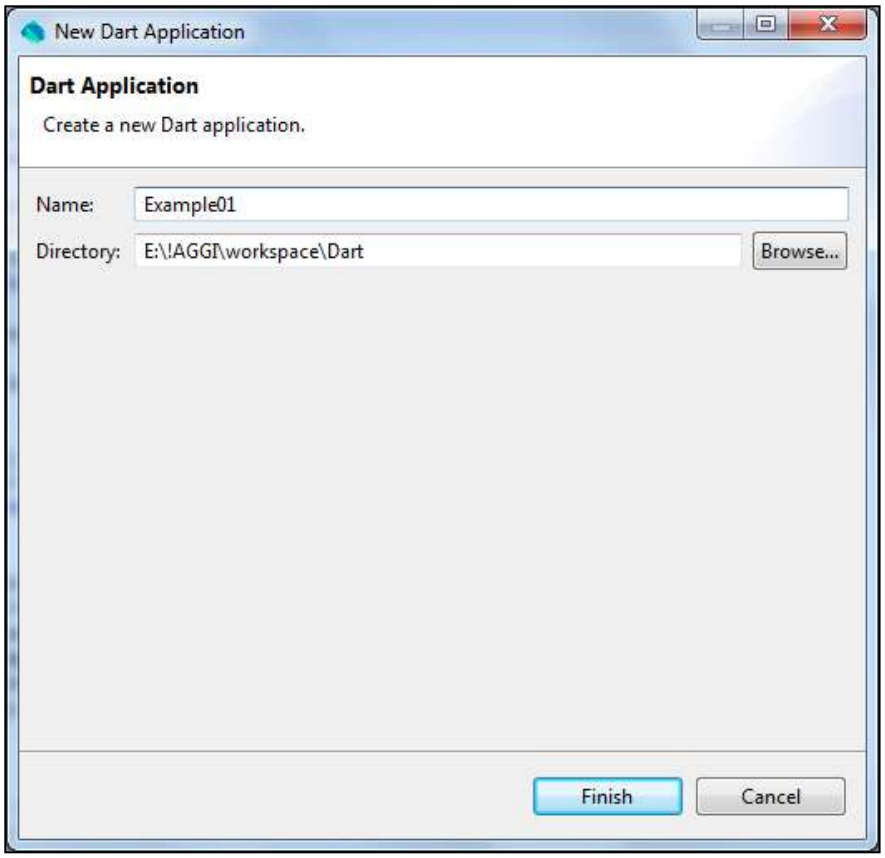
   Example: #source("mySource.dart");

1.2 Creating a New Application

1. Click File → New Application… on the menubar of Dart Editor.

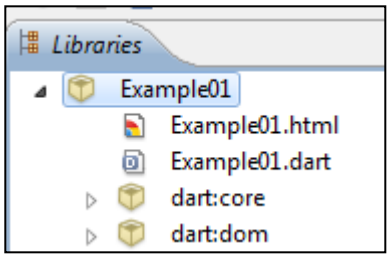


2. Set the name of the application and location of the project.



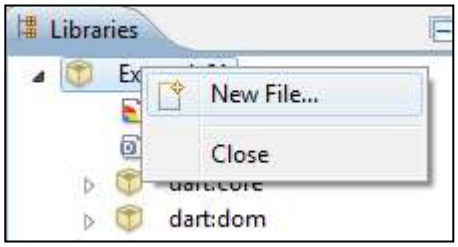
Click the "Finish" button.

3. The new application will be included in the libraries panel.

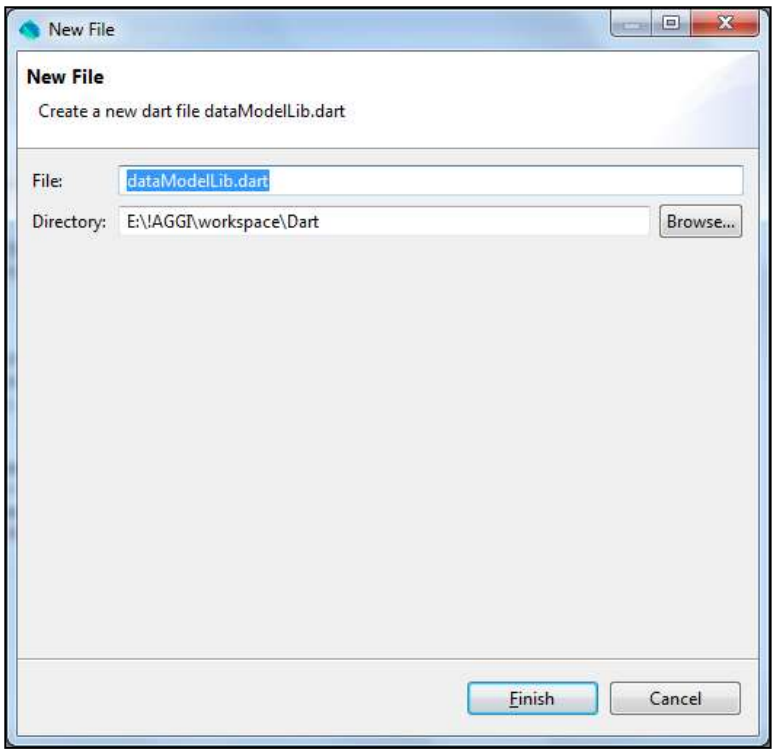


1.3 Creating a Library

1. Right click the Example01 application under libraries panel then click New File… menu.



2. Set the name of the file.

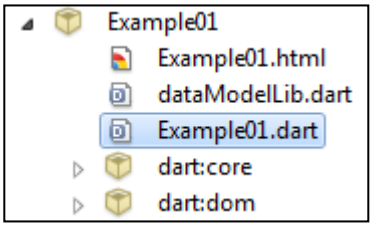


3. Put the following code as an example of using classes and library in Dart.

```
class Order {
  List<OrderItem> orderItems;
}

class OrderItem {
  Product product;
  num qty;
}

class Product {
  String desc;
  num price;
}

class Customer {
  List<Order> orders;
  String name;
  Customer(this.name);
}
```

1.4 Creating the Code

1. Open the Example01.dart file.



2. Put the following code into the file.

```
//using dataModelLib library


#import('dart:dom');
#source('dataModelLib.dart');

class Example01 {

  Example01() {
  }

  void run() {
    Customer customer = new Customer("Reza");
    write("Welcome, " + customer.name + "!");
  }

  void write(String message) {
    // the DOM library defines a global "window" variable
    HTMLDocument doc = window.document;
    HTMLParagraphElement p = doc.createElement('p');
    p.innerText = message;
    doc.body.appendChild(p);
  }
}

void main() {
  new Example01().run();
}
```

Here is the result when the application runs under Google Chrome.

Module #02 : Building a Simple Dart Application UI

Overview: Creating a simple "Hello World" Node.js application

2.1 A Traditional Way to Generate the HTML

Here is an example of one traditional way in generating an HTML page.

```
<html>
  <head>
    <title>Example02</title>
  </head>
  <body>
    <%= someContent %>
    <% someContent2 %>
  </body>
</html>
```

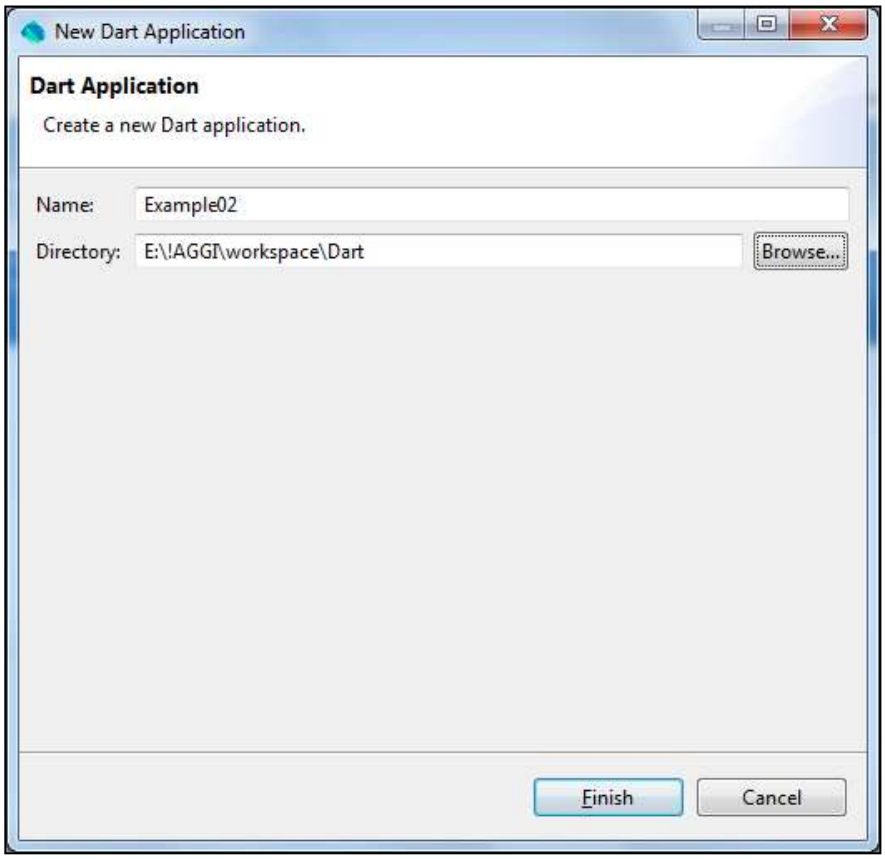
The server will get the request for a page, generate the HTML by replacing the tag libraries or tokens, and return the generated HTML page to client.

2.2 Dart Way to Generate the HTML

Dart is created by Google. If you think about the sort of applications that Google builds, they are most like a standard desktop application that hosted in a browser. Lack of navigation page to page, interact with the data within a single interface.

2.3 Simple UI

1. Create a new Dart application, name it with "Example02".



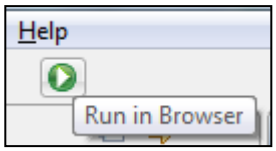
2. A new project will be generated. The Example02.html file will contain this following code.

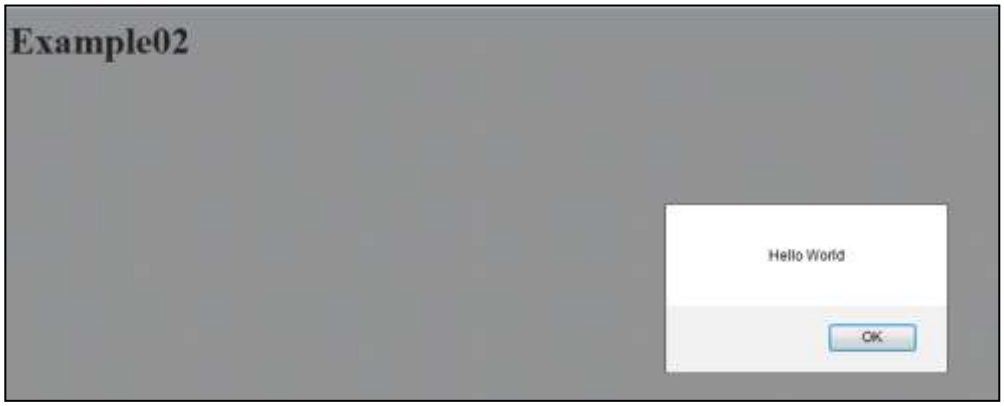
```
<html>
  <head>
    <title>Example02</title>
  </head>
  <body>
    <h1>Example02</h1>
    <script type="text/javascript"
src="Example02.dart.app.js"></script>
  </body>
</html>
```

3. Open the Example02.dart file and put this following code.

```
#library('Example02');

#import('dart:html');

void main() {
  document.window.alert("Hello World");
}
```

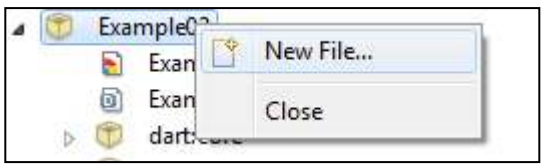
4. Make sure the program runs well by running it in browser.



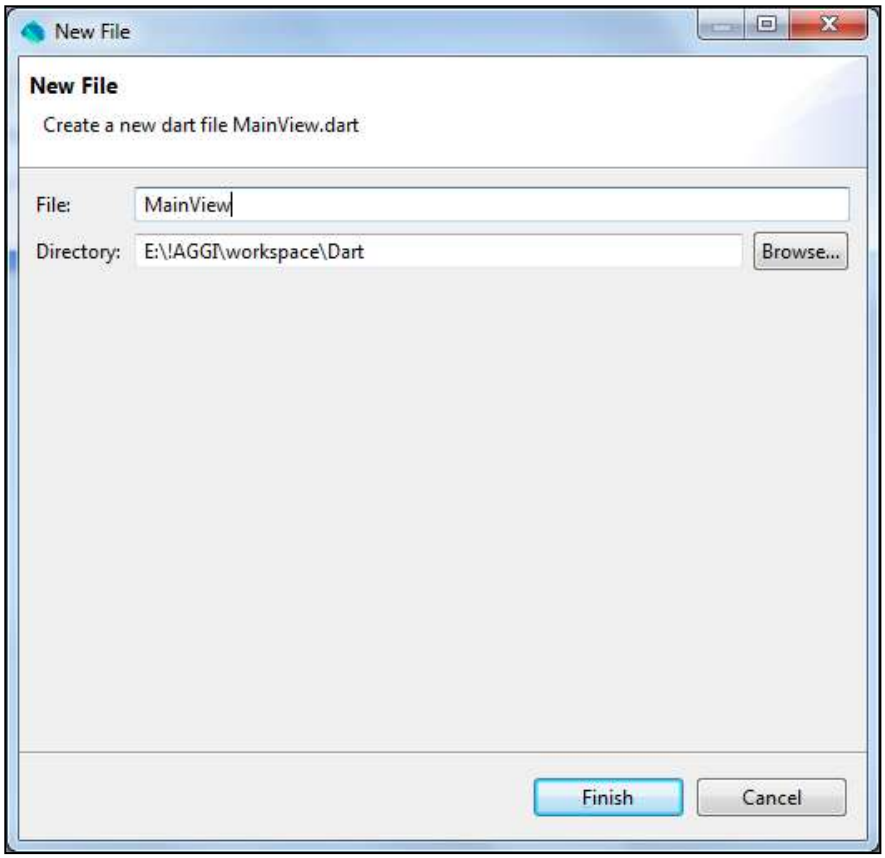


The `#import('dart:html');` loads the dart html library. Once the application runs, it will execute the `main()` function that will show a dialog box using this code `document.window.alert("Hello World");`.

5. Create the class of the view. Right click on the project name under libraries panel, then click New File… .



Set the filename to "MainView".



Click "Finish" button.

6. Put this following code to MainView.dart file.

```
class MainView extends CompositeView {
  MainView() {
    View titleView = new View.html('<h1>Post It</h1>');
    this.addChild(titleView);
  }
}
```

The code above will be detected with errors because the CompositeView class can't be found. The following line need to be added to Example02.dart file.

```
#import('libraries/view/view.dart');
```

Remember that `'libraries/view/view.dart'` is a relative path to your Dart directory.

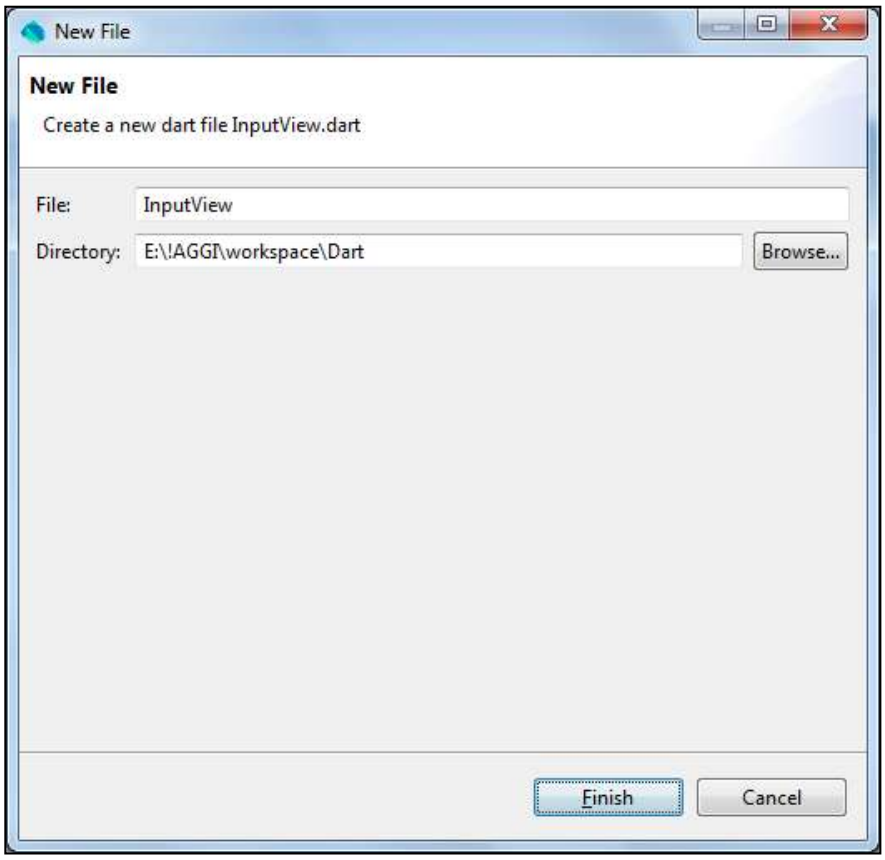
The CompositeView class is a view that is made up from other views.

7. Open the Example02.dart file and update the code.

```
#library('Example02');

#import('dart:html');
#import('libraries/view/view.dart');
#source('MainView.dart');

void main() {
  try {
    MainView mainView = new MainView();
    mainView.addToDocument(document.body);
  }
  catch(Exception ex) {
    document.window.alert("Exception: " + ex.toString());
  }
}
```

The `addToDocument()` function is used to add a view as a child of the node provided.

8. Create a class that renders textbox for inputting data.



```
class InputView extends View {
  InputElement textBox;

  Element render() {
    textBox = document.createElement("input");
    return textBox;
  }
}
```

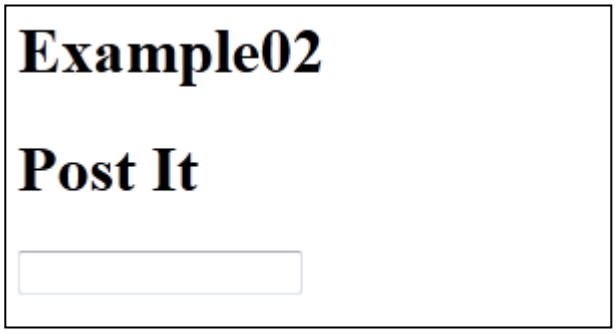
9. Open the MainView.dart file and update the code.

```
class MainView extends CompositeView {
  MainView() {
    View titleView = new View.html('<h1>Post It</h1>');
    this.addChild(titleView);
    InputView inputView = new InputView();
    this.addChild(inputView);
  }
}
```

10. Add this following line in Example02.dart file.

```
#source('InputView.dart');
```

11. Run the application and review the result.



There will be a first heading and a textbox.

Module #03 : Dart Application with Script

Overview: Creating a simple Dart application that can add some text to a Div container

3.1 View with Event

1. We still need the previous example, update some code into it so that the application can do some script.

2. Create a class that render a Div element contains posted text from user.

```
class PostView extends View {
  DivElement posted;

  Element render() {
    posted = document.createElement('div');
    posted.innerHTML = '';
    return posted;
  }

  void addPost(String post) {
    posted.innerHTML = post + '<br />' + posted.innerHTML;
  }
}
```

3. Create a class that render a button element with some an event handler.

```
class PostView extends View {
  DivElement posted;

  Element render() {
    posted = document.createElement('div');
    posted.innerHTML = '';
    return posted;
  }

  void addPost(String post) {
    posted.innerHTML = post + '<br />' + posted.innerHTML;
  }
}
```

4. Open the MainView.dart file and update the code.

```
class MainView extends CompositeView {
  MainView() {
    View titleView = new View.html('<h1>Post It</h1>');
    this.addChild(titleView);
    InputView inputView = new InputView();
    this.addChild(inputView);

    PostView postView = new PostView();
    ButtonView buttonView = new ButtonView(postView, inputView);
    this.addChild(buttonView);
    this.addChild(postView);
  }
}
```

5. Run the application and you will get something like this.

The button created has an event of clicking that will append a div container with text written in the textbox input. The events provided are the same with JavaScript events. See the documentation for further information.

Module #04 : Reading and Writing File System

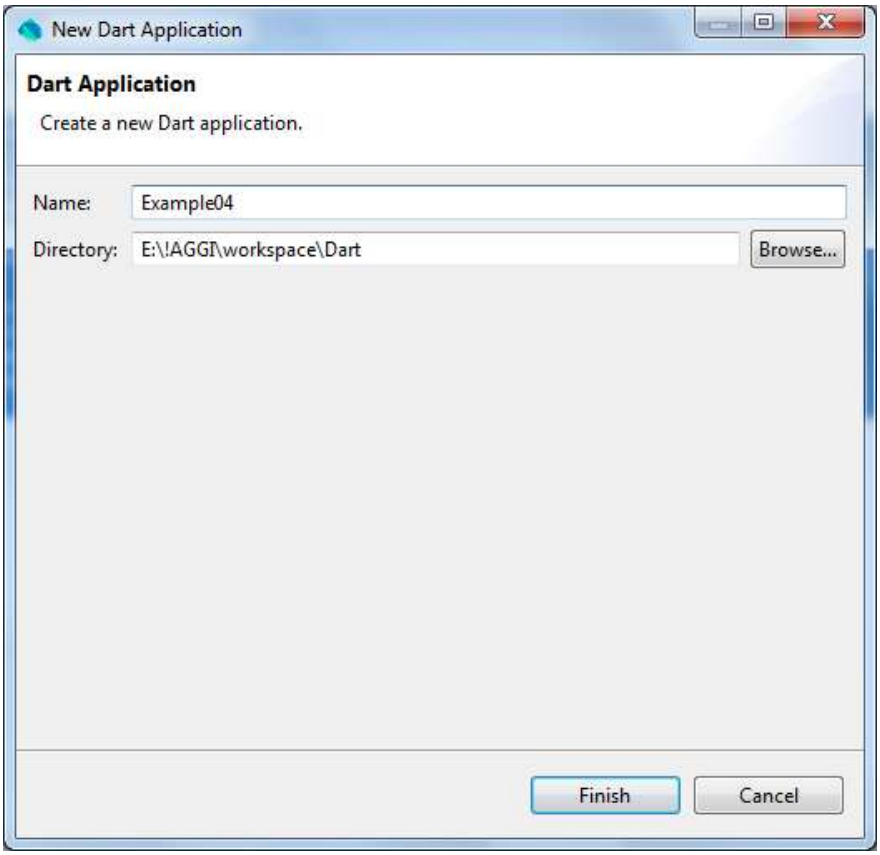
Overview: Creating an application that can read and write a file

4.1 Read and Write in Server

One of the tasks of server side code is often to be able to read and write to the file system. We will do that things using Dart language.

4.2 Read and Write File

1. Create a new Dart application, name it with "Example04"..



2. Open the Example04.dart file and put this following code.

```
#library('fileSystem');

void main() {
  final String filename = 'myFile.txt';

  /* Write */
  File writableFile = new File(filename);

  if(writableFile.existsSync()) {
    print("File ${filename} already exists, the old file will be
overwritten!");
  }
  else {
    print("File ${filename} doesn't exist, a new file will be
created!");
    writableFile.createSync();
  }

  writableFile.openSync(true);
  writableFile.writeStringSync("Test write line 1\nThis is line
2");

  print("File ${filename} has been written.");
  writableFile.closeSync();
  print("File ${filename} has been closed.");

  /* Read */
  File readableFile = new File(filename);
  readableFile.openSync(false);
  int fileLength = readableFile.lengthSync();

  // Create buffer that big enough for the file
  List<int> buffer = new List<int>(fileLength);
  print("File ${filename} has been opened. Length =
${fileLength}.");

  // Read the contents
  readableFile.readListSync(buffer, 0, fileLength);
  readableFile.closeSync();

  // Convert buffer to string
  String text = new String.fromCharCodes(buffer);

  print("${filename}: \n" + text);
}
```

The File interface is hosted at http://code.google.com/p/dart/source/browse/trunk/dart/runtime/bin/file.dart that provides us with a few methods for managing file system. The

constructor of File class contains a boolean variable. Passing **true** into the **new File()** constructor means writeable. The code above uses **file.createSync()** and **file.openSync()** to check whether the file exists or not, and create and open the file. The **file.writeStringSync()** method is used to write lines of text into the file. The **file.closeSync()** is used to close the file.

After that, we open the file with the same filename and read the contents into a buffer which is a list of int. Then the list of int is converted into string using **String.fromCharCodes()** function.