google web toolkit

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Google Web Toolkit (GWT) is a development toolkit for building and optimizing complex browser-based applications. GWT is used by many products at Google, including Google AdWords and Orkut.

GWT is an open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache License version 2.0.

This tutorial will give you a great understanding of GWT concepts needed to get a web application up and running.

Audience

This tutorial is designed for software professionals who are willing to learn GWT programming in simple and easy steps. This will also give you great understanding on GWT Programming concepts.

After completing this tutorial, you will be at intermediate level of expertise from where you can take yourself at higher level of expertise.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of Java programming language, text editor, and execution of programs, etc. Because we are going to develop web-based applications using GWT, it will be good if you have an understanding of other web technologies such as HTML, CSS, and AJAX.

Copyright and Disclaimer

© Copyright 2015 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

tutorialspoint
SIMPLYEASYLEARNING

Table of Contents

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

1. GWT – OVERVIEW

What is GWT?

Google Web Toolkit (GWT) is a development toolkit to create **RICH Internet Applications (RIA)**. Here are some of its notable features:

- GWT provides developers option to write client-side applications in Java.
- GWT compiles the code written in JAVA to JavaScript code.
- Application written in GWT is cross-browser compliant. GWT automatically generates JavaScript code suitable for each browser.
- GWT is an open source, completely free, and used by thousands of developers around the world. It is licensed under the Apache License version 2.0.

Overall, GWT is a **framework** to build large-scale and high-performance web applications while making them easy-to-maintain.

Why to Use GWT?

Being Java based, you can use JAVA IDEs like Eclipse to develop a GWT application.

Developers can use code auto-complete/refactoring/navigation/project management and all features of IDEs.GWT which provides full debugging capability. Developers can debug the client side application just as a Java Application.

- GWT provides easy integration with Junit and Maven.
- Again being Java based, GWT has a low learning curve for Java Developers.
- GWT generates optimized JavaScript code, produces browser's specific JavaScript code by self.
- GWT provides widgets library which provides most of the tasks required in an application.
- GWT is extensible and custom widget based which can be created to cater to various application needs.

On top of everything, GWT applications can run on all major browsers and smart phones including Android and iOS based phones/tablets.

Disadvantages of GWT

Although GWT offers plenty of advantages, it suffers from the following disadvantages

- **Not Indexable**: Web pages generated by GWT would not be indexed by search engines because these applications are generated dynamically.
- **Not Degradable**: If your application user disables JavaScript then user will just see the basic page and nothing more.
- **Not Designer's Friendly**: GWT is not suitable for web designers who prefer using plain HTML with placeholders for inserting dynamic content at a later point in time.

The GWT Components

The GWT framework can be divided into the following three major parts

- **GWT Java to JavaScript compiler**: This is the most important part of GWT which makes it a powerful tool for building RIAs. The GWT compiler is used to translate all the application code written in Java into JavaScript.
- **JRE Emulation library**: Google Web Toolkit includes a library that emulates a subset of the Java runtime library. The list includes java.lang, java.lang.annotation, java.math, java.io, java.sql, java.util and java.util.logging
- **GWT UI building library**: This part of GWT consists of many subparts which includes the actual UI components, RPC support, History management, and much more.

GWT also provides a GWT Hosted Web Browser which lets you run and execute your GWT applications in hosted mode, where your code runs as Java in the Java Virtual Machine without compiling to JavaScript.

tutorialspoint
SIMPLYEASYLEARNING

2. GWT ENVIRONMENT

This tutorial will guide you on how to prepare a development environment to start your work with GWT Framework. This tutorial will also teach you how to setup JDK, Tomcat and Eclipse on your machine before you setup GWT Framework:

System Requirement

GWT requires JDK 1.6 or higher so the very first requirement is to have JDK installed in your machine.

JDK	1.6 or above.
Memory	No minimum requirement.
Disk Space	No minimum requirement.
Operating System	No minimum requirement.

Follow the given steps to setup your environment to start with GWT application development.

Step 1 – Verify Java Installation on Your Machine

Now open the console and execute the following java command.

OS	Task	Command
Windows	Open Command Console	c:\> java -version
Linux	Open Command Terminal	$ java -version
Mac	Open Terminal	machine:~ joseph$ java -version

tutorialspoint
SIMPLYEASYLEARNING

Let's verify the output for all the operating systems

OS	Generated Output
Windows	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java Hotspot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
Linux	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) ava HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
Mac	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM)64-Bit Server VM (build 17.0-b17, mixed mode, sharing)

Step 2 – Setup Java Development Kit (JDK)

If you do not have Java installed then you can install the Java Software Development Kit (SDK) from Oracle's Java site: **Java SE Downloads**. You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup. Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains java and javac, typically java_install_dir/bin and java_install_dir respectively.

Set the **JAVA_HOME** environment variable to point to the base directory location where Java is installed on your machine. For example

OS	Output
Windows	Set the environment variable JAVA_HOME to C:\Program Files\Java\jdk1.6.0_21
Linux	export JAVA_HOME=/usr/local/java-current
Mac	export JAVA_HOME=/Library/Java/Home

Append Java compiler location to System Path.

OS	Output
Windows	Append the string; %JAVA_HOME%\bin to the end of the system variable, Path.
Linux	export PATH=$PATH:$JAVA_HOME/bin/
Mac	not required

Alternatively, if you use an Integrated Development Environment (IDE) like Borland JBuilder, Eclipse, IntelliJ IDEA, or Sun ONE Studio, compile and run a simple program to confirm that the IDE knows where you installed Java, otherwise do proper setup as given document of the IDE.

Step 3 – Setup Eclipse IDE

All the examples in this tutorial have been written using Eclipse IDE. Hence, suggest you to install the latest version of Eclipse on your machine based on your operating system.

To install Eclipse IDE, download the latest Eclipse binaries from **http://www.eclipse.org/downloads/**. Once you downloaded the installation, unpack the binary distribution into a convenient location. For example in C:\eclipse on windows, or /usr/local/eclipse on Linux/Unix and finally set PATH variable appropriately.

Eclipse can be started by executing the following commands on windows machine, or you can simply double click on eclipse.exe

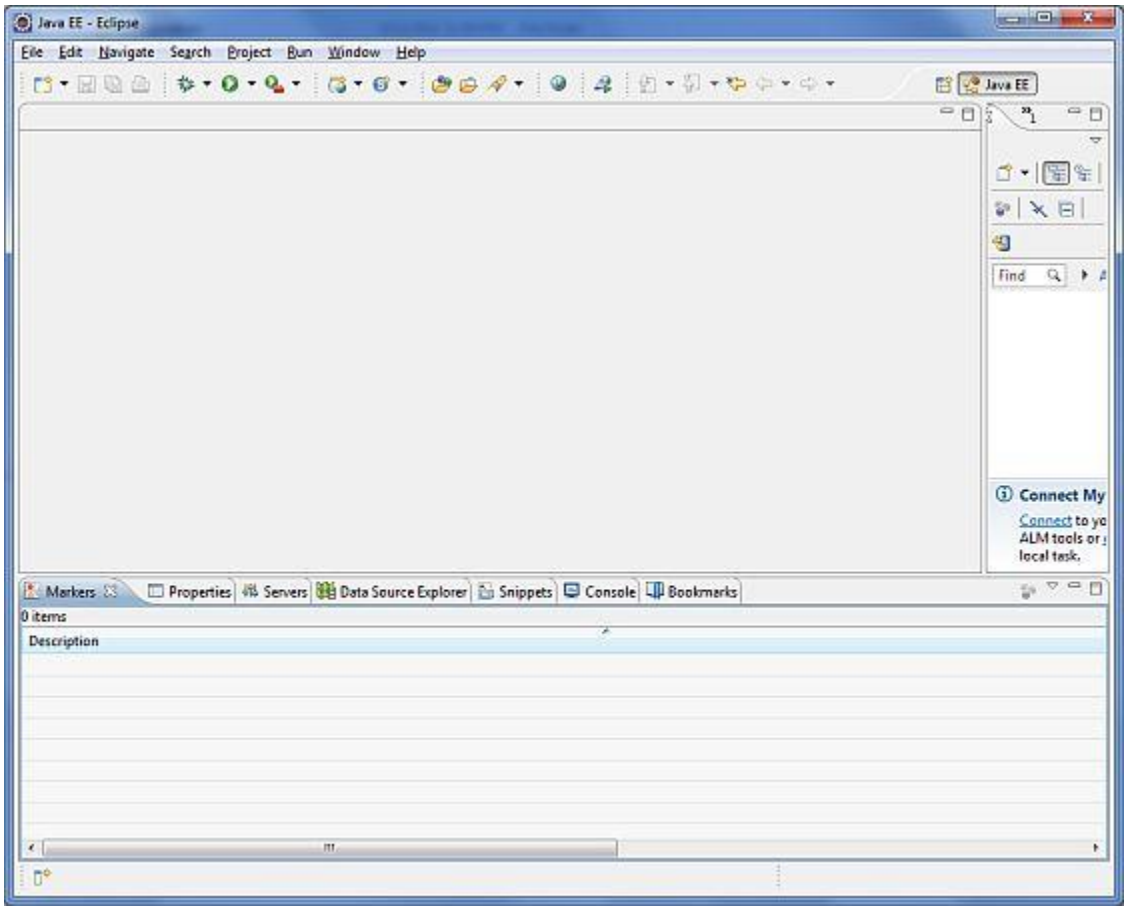
```
%C:\eclipse\eclipse.exe
```

Eclipse can be started by executing the following commands on UNIX (Solaris, Linux, etc.) machine:

```
$/usr/local/eclipse/eclipse
```

After a successful startup, if everything is fine then it should display following result:

tutorialspoint
SIMPLYEASYLEARNING

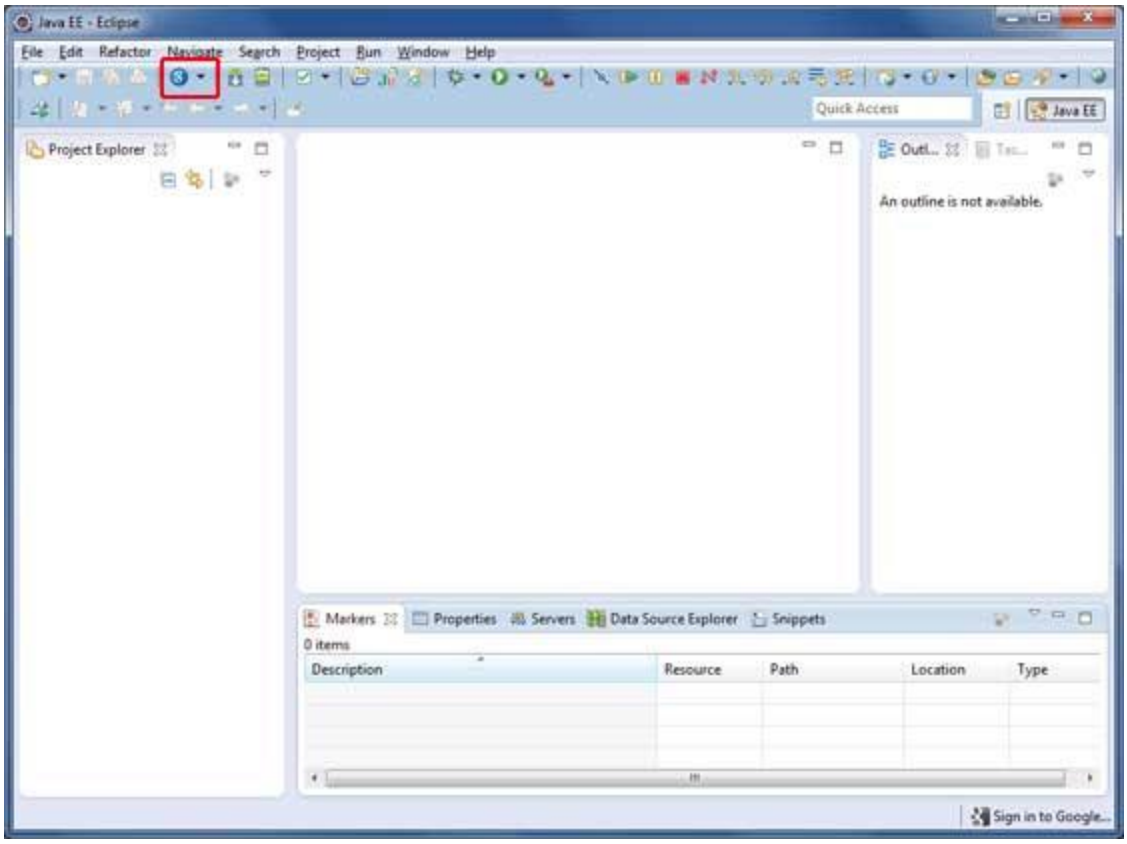


Step 4 – Install GWT SDK & Plugin for Eclipse

Follow the instructions given at the link **Plugin for Eclipse (incl. SDKs)** to install GWT SDK & Plugin for Eclipse version installed on your machine.

After a successful setup for the GWT plugin, if everything is fine, then it should display the following screen which has **Google icon** marked in red rectangle as shown below:

tutorialspoint
SIMPLYEASYLEARNING



Step 5: Setup Apache Tomcat

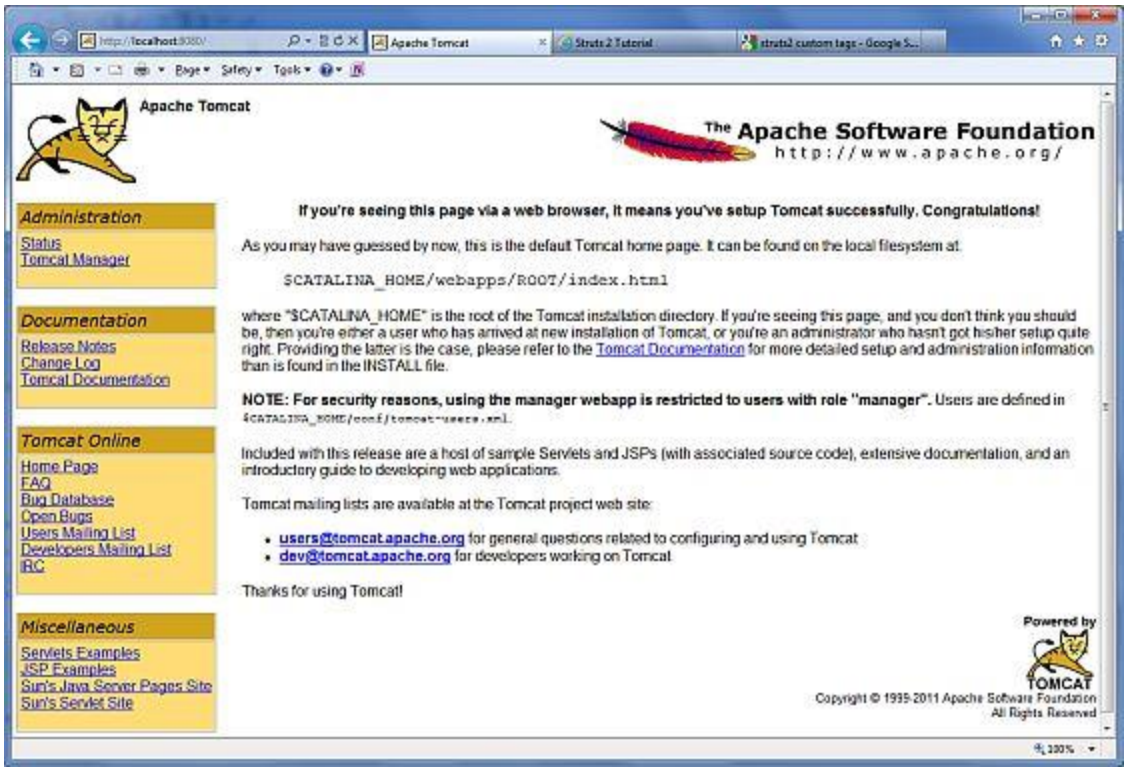
You can download the latest version of Tomcat from **http://tomcat.apache.org/**. Once you downloaded the installation, unpack the binary distribution into a convenient location. For example in C:\apache-tomcat-6.0.33 on windows, or /usr/local/apache-tomcat-6.0.33 on Linux/Unix and set CATALINA_HOME environment variable pointing to the installation locations.

Tomcat can be started by executing the following commands on windows machine, or you can simply double click on startup.bat

```
%CATALINA_HOME%\bin\startup.bat

or /usr/local/apache-tomcat-6.0.33/bin/startup.sh
```

After a successful startup, the default web applications included with Tomcat will be available by visiting **http://localhost:8080/**. If everything is fine then it should display following result:

tutorialspoint
SIMPLYEASYLEARNING



Further information about configuring and running Tomcat can be found in the documentation included here, as well as on the Tomcat web site: http://tomcat.apache.org

Tomcat can be stopped by executing the following commands on windows machine:

```
%CATALINA_HOME%\bin\shutdown

Or C:\apache-tomcat-5.5.29\bin\shutdown
```

Tomcat can be stopped by executing the following commands on UNIX (Solaris, Linux, etc.) machine:

```
$CATALINA_HOME/bin/shutdown.sh

Or

/usr/local/apache-tomcat-5.5.29/bin/shutdown.sh
```

3. GWT APPLICATIONS

Before we start with creating actual "HelloWorld" application using GWT, let us see what the actual parts of a GWT application are:

A GWT application consists of following four important parts out of which last part is optional but first three parts are mandatory

- Module descriptors
- Public resources
- Client-side code
- Server-side code

Sample locations of different parts of a typical GWT **Hello World** application looks like the below:

Name	Location
Project root	HelloWorld/
Module descriptor	src/com/tutorialspoint/HelloWorld.gwt.xml
Public resources	src/com/tutorialspoint/war/
Client-side code	src/com/tutorialspoint/client/
Server-side code	src/com/tutorialspoint/server/

Module Descriptors

A module descriptor is the configuration file in the form of XML which is used to configure a GWT application.

A module descriptor file extension is *.gwt.xml, where * is the name of the application and this file should reside in the project's root.

Following will be the default module descriptor HelloWorld.gwt.xml for a HelloWorld application:

```
<?xml version="1.0" encoding="utf-8"?>
<module rename-to='helloworld'>
   <!-- inherit the core web toolkit stuff.                          -->
   <inherits name='com.google.gwt.user.user'/>
```


tutorialspoint
SIMPLYEASYLEARNING

```
   <!-- inherit the default gwt style sheet.                        -->
   <inherits name='com.google.gwt.user.theme.clean.Clean'/>

   <!-- specify the app entry point class.                          -->
   <entry-point class='com.tutorialspoint.client.HelloWorld'/>

   <!-- specify the paths for translatable code                     -->
   <source path='...'/>
   <source path='...'/>

   <!-- specify the paths for static files like html, css etc.      -->
   <public path='...'/>
   <public path='...'/>

   <!-- specify the paths for external javascript files             -->
   <script src="js-url" />
   <script src="js-url" />

   <!-- specify the paths for external style sheet files             -->
   <stylesheet  src="css-url" />
   <stylesheet  src="css-url" />
</module>
```

Following is a brief detail of different parts used in module descriptor

S.No.	Nodes & Description
1	<module rename-to="helloworld"> This provides name of the application.

tutorialspoint
SIMPLYEASYLEARNING

2	<inherits name="logical-module-name" />This adds other gwt module in application just like import does in java applications. Any number of modules can be inherited in this manner.
3	<entry-point class="classname" />This specifies the name of class which will start loading the GWT Application. Any number of entry-point classes can be added and they are called sequentially in the order in which they appear in the module file. So when the onModuleLoad() of your first entry point finishes, the next entry point is called immediately.
4	<source path="path" />This specifies the names of source folders which GWT compiler will search for source compilation.
5	<public path="path" />The public path is the place in your project where static resources referenced by your GWT module, such as CSS or images, are stored. The default public path is the public subdirectory underneath where the Module XML File is stored.
6	<script src="js-url" />Automatically injects the external JavaScript file located at the location specified by src.
7	<stylesheet src="css-url" />Automatically injects the external CSS file located at the location specified by src.

Public Resources

These are the various files referenced by your GWT module, such as Host HTML page, CSS or images.

The location of these resources can be configured using <public path="path" /> element in module configuration file. By default, it is the public subdirectory underneath where the Module XML File is stored.

When you compile your application into JavaScript, all the files that can be found on your public path are copied to the module's output directory.

The most important public resource is host page which is used to invoke actual GWT application. A typical HTML host page for an application might not include any visible HTML body content at all but it is always expected to include GWT application via a <script.../> tag as follows

```
<html>
<head>
<title>Hello World</title>
   <link rel="stylesheet" href="HelloWorld.css"/>
```


tutorialspoint
SIMPLYEASYLEARNING

```
   <script language="javascript" src="helloworld/helloworld.nocache.js">
   </script>
</head>
<body>

<h1>Hello World</h1>
<p>Welcome to first GWT application</p>

</body>
</html>
```

Following is the sample style sheet which we have included in our host page:

```
body {
   text-align: center;
   font-family: verdana, sans-serif;
}
h1 {
  font-size: 2em;
  font-weight: bold;
  color: #777777;
  margin: 40px 0px 70px;
  text-align: center;
}
```

Client-side Code

This is the actual Java code written for implementing the business logic of the application and with this, GWT compiler translates into JavaScript, which will eventually run inside the browser. The location of these resources can be configured using <source path="path" /> element in module configuration file.

For example, **Entry Point** code will be used as client-side code and its location will be specified using <source path="path" />.

tutorialspoint
SIMPLYEASYLEARNING

A module **entry-point** is any class that is assignable to **EntryPoint** and that can be constructed without parameters. When a module is loaded, every entry point class is instantiated and its **EntryPoint.onModuleLoad()** method gets called. A sample HelloWorld Entry Point class will be as follows:

```
public class HelloWorld implements EntryPoint {
   public void onModuleLoad() {
      Window.alert("Hello, World!");
   }
}
```

Server-side Code

This is the server side part of your application and it is very much optional. If you are not doing any backend processing within your application, then you do not need this part. However, if there is some processing required at backend and your client-side application interacts with the server, then you will have to develop these components.

The next chapter will make use of all the above-mentioned concepts to create a HelloWorld application using Eclipse IDE.

tutorialspoint
SIMPLYEASYLEARNING

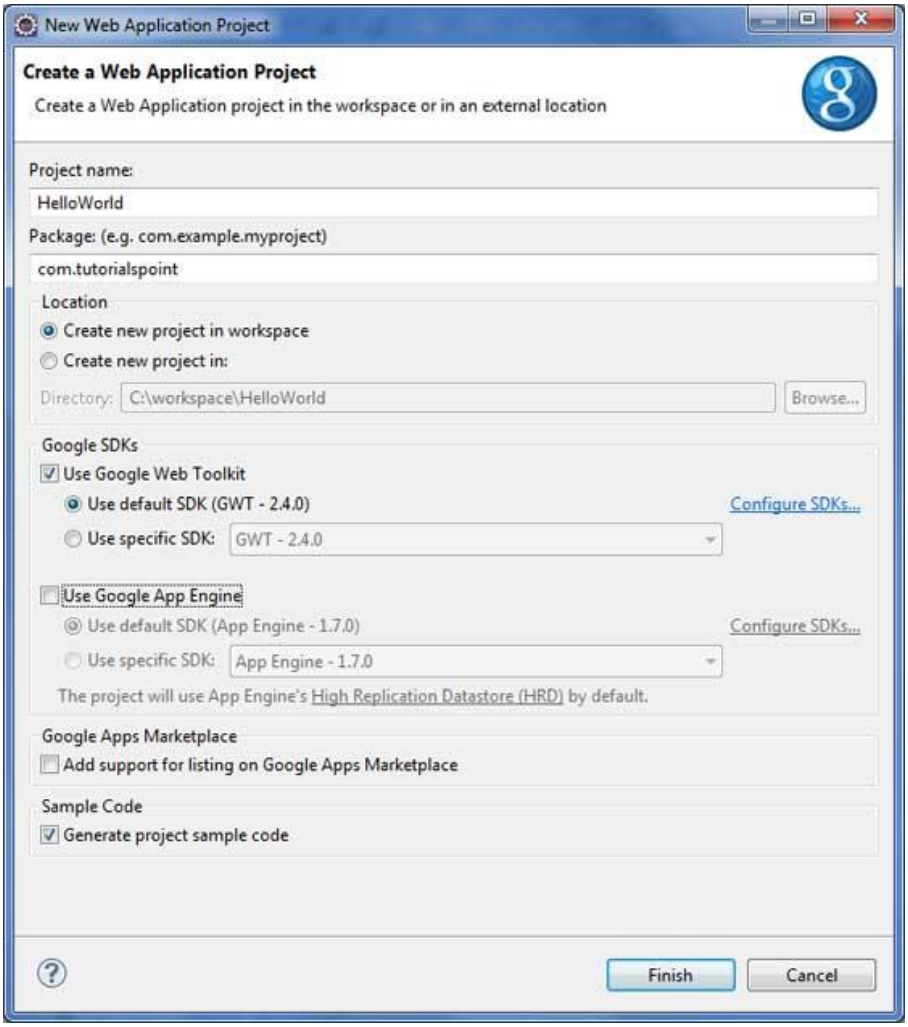
4. GWT – CREATE APPLICATION

As the power of GWT lies in **Write in Java, Run in JavaScript**, we'll be using Java IDE Eclipse to demonstrate our examples.

Let's start with a simple *HelloWorld* application:

Step 1 – Create Project

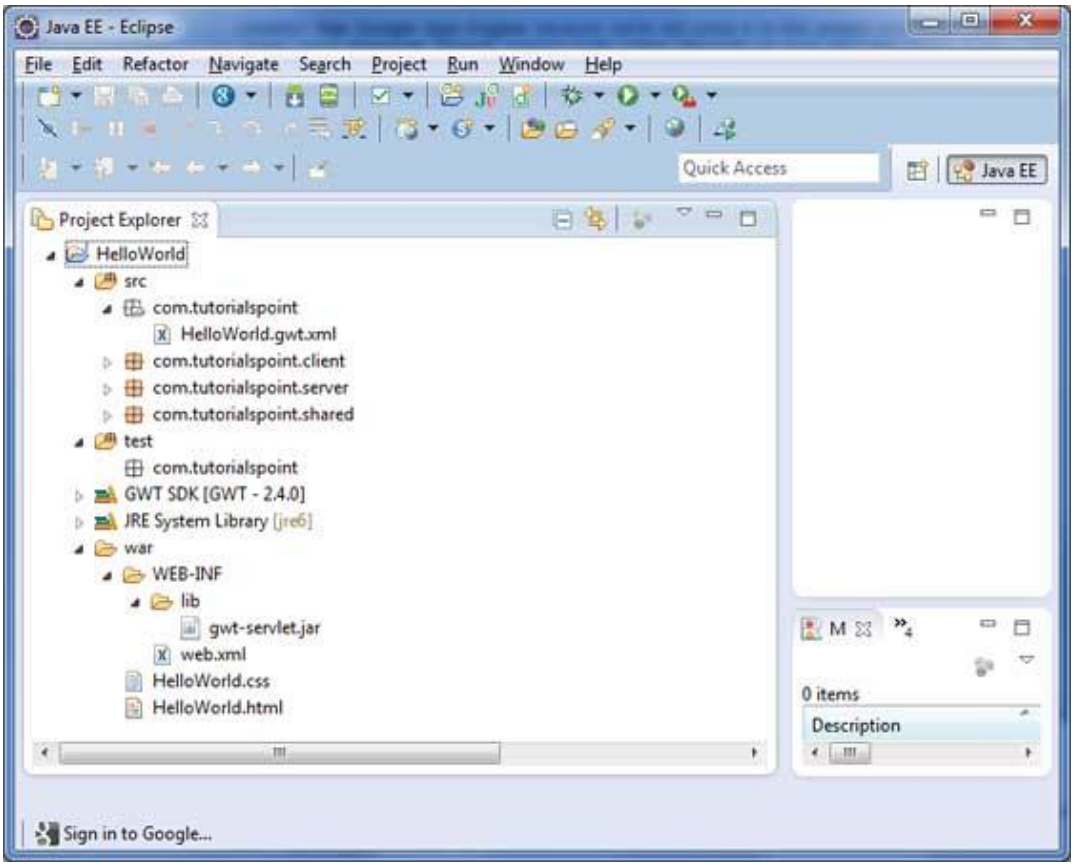
The first step is to create a simple Web Application Project using Eclipse IDE. Launch project wizard using the option Google Icon > New Web Application Project.... Now name your project as *HelloWorld* using the wizard window as follows:



tutorialspoint
SIMPLYEASYLEARNING

Unselect **Use Google App Engine** because we're not using it in this project and leave other default values (keep the **Generate Sample project code** option checked as such and click Finish Button).

Once your project is created successfully, you will have the following content in your Project Explorer:



Here is a brief description of all the important folders

Folder	Location
src	Source code (java classes) files. Client folder containing the client-side specific java classes responsible for client UI display. Server folder containing the server-side java classes responsible for server side processing.

tutorialspoint
SIMPLYEASYLEARNING

	Shared folder containing the java model class to transfer data from server to client and vice versa. HelloWorld.gwt.xml, a module descriptor file required for GWT compiler to compile the HelloWorld project.
test	Test code (java classes) source files. Client folder containing the java classes responsible to test gwt client side code.
war	This is the most important part, it represents the actual deployable web application. WEB-INF containing compiled classes, gwt libraries, servlet libraries. HelloWorld.css, project style sheet. HelloWorld.html, hots HTML which will invoke GWT UI Application.

Step 2 - Modify Module Descriptor: HelloWorld.gwt.xml

GWT plugin will create a default module descriptor file *src/com.tutorialspoint/HelloWorld.gwt.xml* which is given below. For this example, we are not modifying it, but you can modify it later based on your requirement.

```
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to='helloworld'>
  <!-- Inherit the core Web Toolkit stuff.                        -->
  <inherits name='com.google.gwt.user.User'/>

  <!-- Inherit the default GWT style sheet.  You can change       -->
  <!-- the theme of your GWT application by uncommenting          -->
  <!-- any one of the following lines.                            -->
  <inherits name='com.google.gwt.user.theme.clean.Clean'/>
  <!-- <inherits name='com.google.gwt.user.theme.chrome.Chrome'/> -->
  <!-- <inherits name='com.google.gwt.user.theme.dark.Dark'/>     -->

  <!-- Other module inherits                                      -->
```


tutorialspoint
SIMPLYEASYLEARNING

```
  <!-- Specify the app entry point class.                         -->
  <entry-point class='com.tutorialspoint.client.HelloWorld'/>

  <!-- Specify the paths for translatable code                    -->
  <source path='client'/>
  <source path='shared'/>

</module>
```

Step 3 - Modify Style Sheet: HelloWorld.css

GWT plugin will create a default Style Sheet file *war/HelloWorld.css*. Now, let us modify this file to keep our example simple and easier to understand

```
body {
   text-align: center;
   font-family: verdana, sans-serif;
}
h1 {
  font-size: 2em;
  font-weight: bold;
  color: #777777;
  margin: 40px 0px 70px;
  text-align: center;
}
```

Step 4 - Modify Host File: HelloWorld.html

GWT plugin will create a default HTML host file *war/HelloWorld.html*. Let us modify this file to keep our example simple and easier to understand

```
<html>
<head>
```


tutorialspoint
SIMPLYEASYLEARNING

```
<title>Hello World</title>
   <link rel="stylesheet" href="HelloWorld.css"/>
   <script language="javascript" src="helloworld/helloworld.nocache.js">
   </script>
</head>
<body>

<h1>Hello World</h1>
<p>Welcome to first GWT application</p>

</body>
</html>
```

You can create more static files like HTML, CSS or images in the same source directory or you can create further sub-directories and move files in those sub-directories and configure those sub-directories in module descriptor of the application.

Step 5 - Modify Entry Point: HelloWorld.java

GWT plugin will create a default Java file *src/com.tutorialspoint/HelloWorld.java*, which keeps an entry point for the application.

Let us modify this file to display "Hello,World!"

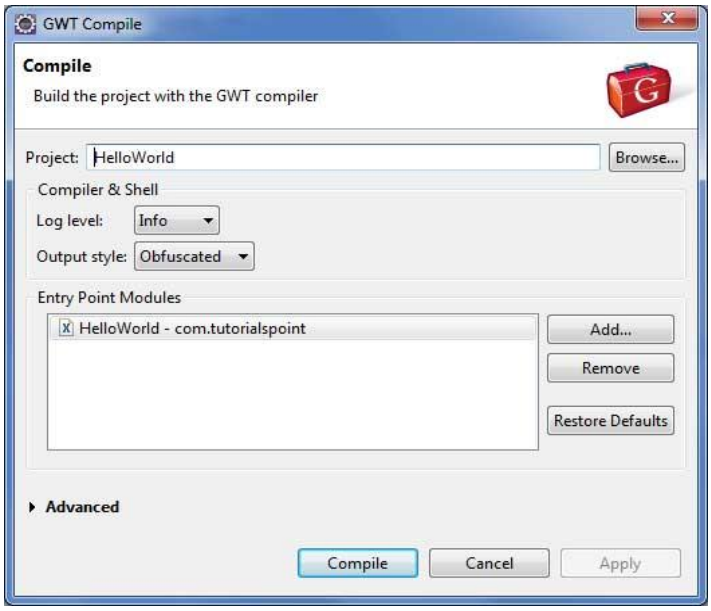
```
package com.tutorialspoint.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.Window;

public class HelloWorld implements EntryPoint {
   public void onModuleLoad() {
        Window.alert("Hello, World!");
   }
}
```


tutorialspoint
SIMPLYEASYLEARNING

You can create more Java files in the same source directory to define either entry points or to define helper routines.

Step 6 - Compile Application

Once you are ready with all the changes done, its time to compile the project. Use the option Google Icon > GWT Compile Project... to launch GWT Compile dialogue box as shown below



Keep the default values intact and click Compile button. If everything goes fine, you will see following output in Eclipse console

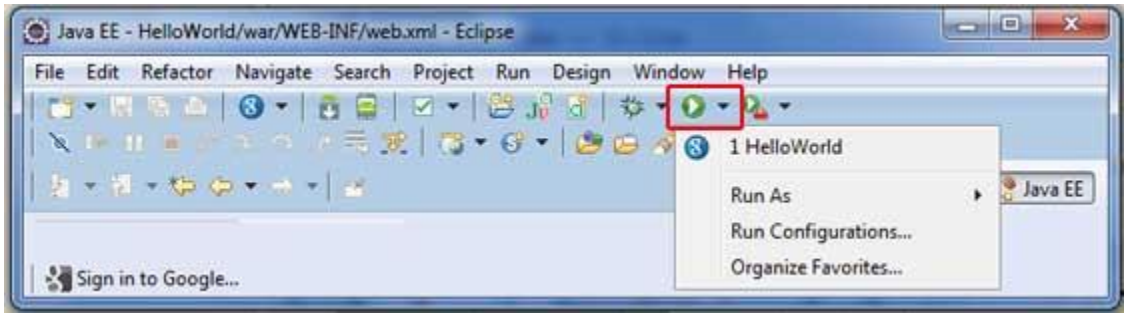
```
Compiling module com.tutorialspoint.HelloWorld
   Compiling 6 permutations
      Compiling permutation 0...
      Compiling permutation 1...
      Compiling permutation 2...
      Compiling permutation 3...
      Compiling permutation 4...
      Compiling permutation 5...
   Compile of permutations succeeded
```


tutorialspoint
SIMPLYEASYLEARNING

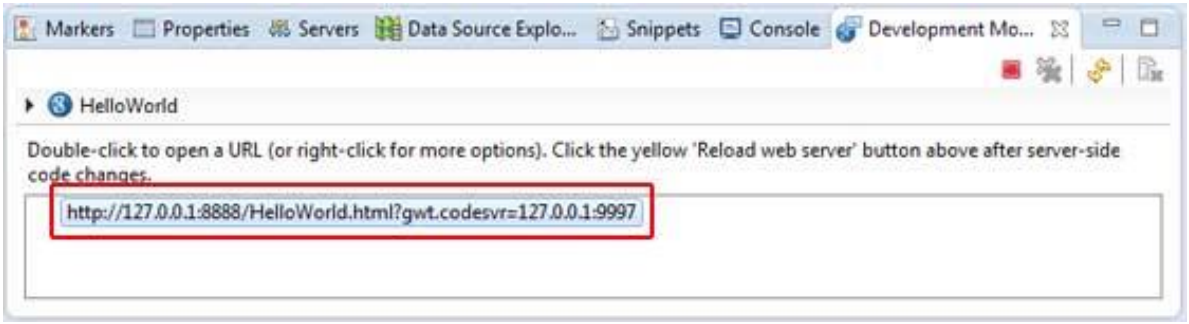
```
Linking into C:\workspace\HelloWorld\war\helloworld
   Link succeeded
   Compilation succeeded -- 33.029s
```

Step 7 - Run Application

Now click on Run application menu and select **HelloWorld** application to run the application.



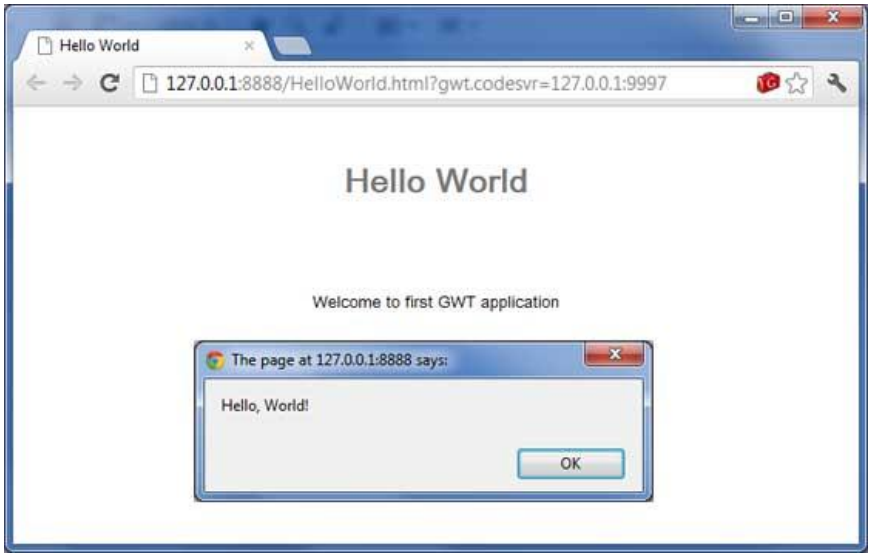
If everything is fine, you must see GWT Development Mode active in Eclipse containing a URL as shown below. Double click the URL to open the GWT application.



Because you are running your application in development mode, you will also need to install GWT plugin for your browser. Simply follow the onscreen instructions to install the plugin.

If you already have GWT plugin set for your browser, then you should be able to see the following output

tutorialspoint
SIMPLYEASYLEARNING



Congratulations! You have implemented your first application using Google Web Toolkit (GWT).

tutorialspoint
SIMPLYEASYLEARNING

5. GWT - DEPLOY APPLICATION

This tutorial will explain you how to create an application "**war"** file and how to deploy that in Apache Tomcat Websever root.

If you understood this simple example, then you will also be able to deploy a complex GWT application following similar steps.

Now, let us have a working Eclipse IDE along with GWT plugin place and follow certain steps to create a GWT application:

Step	Description
1	Create a project with a name *HelloWorld* under a package *com.tutorialspoint* as explained in the *GWT - Create Application* chapter.
2	Modify *HelloWorld.gwt.xml*, *HelloWorld.css*, *HelloWorld.html* and *HelloWorld.java* as explained below. Keep rest of the files unchanged.
3	Compile and run the application to make sure business logic is working as per the requirements.
4	Finally, zip the content of the war folder of the application in the form of war file and deploy it in Apache Tomcat Webserver.
5	Launch your web application using appropriate URL as explained below in the last step.

Following is the content of the modified module descriptor **src/com.tutorialspoint/HelloWorld.gwt.xml**.

```
<?xml version="1.0" encoding="UTF-8"?>
<module rename-to='helloworld'>
  <!-- Inherit the core Web Toolkit stuff.                        -->
  <inherits name='com.google.gwt.user.User'/>

  <!-- Inherit the default GWT style sheet.                       -->
  <inherits name='com.google.gwt.user.theme.clean.Clean'/>
```


tutorialspoint
SIMPLYEASYLEARNING

```
  <!-- Specify the app entry point class.                         -->
  <entry-point class='com.tutorialspoint.client.HelloWorld'/>

  <!-- Specify the paths for translatable code                    -->
  <source path='client'/>
  <source path='shared'/>

</module>
```

Following is the content of the modified Style Sheet file **war/HelloWorld.css**.

```
body {
   text-align: center;
   font-family: verdana, sans-serif;
}
h1 {
  font-size: 2em;
  font-weight: bold;
  color: #777777;
  margin: 40px 0px 70px;
  text-align: center;
}
```

Following is the content of the modified HTML host file **war/HelloWorld.html**.

```
<html>
<head>
<title>Hello World</title>
   <link rel="stylesheet" href="HelloWorld.css"/>
   <script language="javascript" src="helloworld/helloworld.nocache.js">
   </script>
</head>
```


tutorialspoint
SIMPLYEASYLEARNING

```
<body>

<h1>Hello World</h1>
<div id="gwtContainer"></div>

</body>
</html>
```

I modified the HTML a little bit from the previous example. Here I created a placeholder ... where we will insert some content using our entry point java class. So let us have the following content of Java file **src/com.tutorialspoint/HelloWorld.java**.

```
package com.tutorialspoint.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.RootPanel;

public class HelloWorld implements EntryPoint {
   public void onModuleLoad() {
      HTML html = new HTML("<p>Welcome to GWT application</p>");

      RootPanel.get("gwtContainer").add(html);
   }
}
```

Here we created a basic widget HTML and added it inside the div tag having the id="gwtContainer". We will study different GWT widgets in the coming chapters.

Once you are ready with all the changes done, let us compile and run the application in development mode as we did in **GWT - Create Application** chapter. If everything is fine with your application, then this will produce the following result:

tutorialspoint
SIMPLYEASYLEARNING

Create WAR File

Now our application is working fine and we are ready to export it as a war file.

Follow the following steps:

- Go into your project's **war** directory **C:\workspace\HelloWorld\war**
- Select all the files & folders available inside war directory.
- Zip all the selected files & folders in a file called *HelloWorld.zip*.
- Rename *HelloWorld.zip* to *HelloWorld.war*.

Deploy WAR file

- Stop the tomcat server.
- Copy the *HelloWorld.war* file to **tomcat installation directory > webapps folder.**
- Start the tomcat server.
- Look inside webapps directory, there should be a folder **helloworld** got created.
- Now HelloWorld.war is successfully deployed in Tomcat Webserver root.

Run the Application

Enter a url in the web browser: **http://localhost:8080/HelloWorld** to launch the application.

Server name (localhost) and port (8080) may vary as per your tomcat configuration.

tutorialspoint
SIMPLYEASYLEARNING

tutorialspoint
SIMPLYEASYLEARNING

6. GWT - STYLE WITH CSS

GWT widgets rely on cascading style sheets (CSS) for visual styling. By default, the class name for each component is **gwt-<classname>**.

For example, the Button widget has a default style of *gwt-Button* and similar way the TextBox widgest has a default style of *gwt-TextBox*.

In order to give all buttons and text boxes a larger font, you could put the following rule in your application's CSS file

```
.gwt-Button  { font-size: 150%; }
.gwt-TextBox { font-size: 150%; }
```

By default, neither the browser nor GWT creates default **id** attributes for widgets. You must explicitly create a unique id for the elements which you can use in CSS. In order to give a particular button with id **my-button-id** a larger font, you could put the following rule in your application's CSS file

```
#my-button-id { font-size: 150%; }
```

To set the id for a GWT widget, retrieve its DOM Element and then set the id attribute as follows:

```
Button b = new Button();
DOM.setElementAttribute(b.getElement(), "id", "my-button-id")
```

CSS Styling APIs

There are many APIs available to handle CSS setting for any GWT widget. Following are few important APIs which will help you in your day to day web programming using GWT:

S.No.	API & Description
1	public void setStyleName(java.lang.String style) This method will clear any existing styles and set the widget style to the new CSS class provided using *style*.
2	public void addStyleName(java.lang.String style)

tutorialspoint
SIMPLYEASYLEARNING

	This method will add a secondary or dependent style name to the widget. A secondary style name is an additional style name that is,so if there were any previous style names applied they are kept.
3	public void removeStyleName(java.lang.String style) This method will remove given style from the widget and leaves any others associated with the widget.
4	public java.lang.String getStyleName() This method gets all of the object's style names, as a space-separated list.
5	public void setStylePrimaryName(java.lang.String style) This method sets the object's primary style name and updates all dependent style names.

For example, let's define two new styles which we will apply to a text

```
.gwt-Big-Text{
   font-size:150%;
}
.gwt-Small-Text{
   font-size:75%;
}
.gwt-Red-Text{
   color:red;
}
```

Now you can use *setStyleName(Style)* to change the default setting to new setting. After applying the following rule, a text's font will become large

```
txtWidget.setStyleName("gwt-Big-Text");
```

We can apply a secondary CSS rule on the same widget to change its color as follows:

```
txtWidget.addStyleName("gwt-Red-Text");
```

Using the above method, you can add as many styles as you like to apply on a widget. If you remove the first style from the button widget, then the second style will still remain with the text.

tutorialspoint
SIMPLYEASYLEARNING

```
txtWidget.removeStyleName("gwt-Big-Text");
```

Primary & Secondary Styles

By default, the *primary style* name of a widget will be the default style name for its widget class for example *GWT-Button* for Button widgets. When we add and remove style names using AddStyleName() method, those styles are called secondary styles.

The final appearance of a widget is determined by the sum of all the secondary styles added to it, plus its primary style. You set the primary style of a widget with the *setStylePrimaryName(String)* method. To illustrate, let's say we have a Label widget. In our CSS file, we have the following rules defined:

```
.MyText {
  color: blue;
}

.BigText {
  font-size: large;
}

.LoudText {
  font-weight:  bold;
}
```

Let's suppose we want a particular label widget to always display blue text, and in some cases, use a larger, bold font for added emphasis.

We could do something like this:

```
// set up our primary style
Label someText = new Label();
someText.setStylePrimaryName("MyText");
...

// later on, to really grab the user's attention
```


tutorialspoint
SIMPLYEASYLEARNING

```
someText.addStyleName("BigText");
someText.addStyleName("LoudText");
...

// after the crisis is over
someText.removeStyleName("BigText");
someText.removeStyleName("LoudText");
```

Associating CSS Files

There are multiple approaches for associating CSS files with your module. Modern GWT applications typically use a combination of CssResource and UiBinder. We are using only first approach in our examples.

- Using a <link> tag in the host HTML page.
- Using the <stylesheet> element in the module XML file.
- Using a **CssResource** contained within a **ClientBundle**.
- Using an inline <ui:style> element in a **UiBinder** template.

GWT CSS Example

This example will take you through simple steps to apply different CSS rules on your GWT widget. Let us have a working Eclipse IDE along with GWT plug in place and follow the following steps to create a GWT application:

Step	Description
1	Create a project with a name *HelloWorld* under a package *com.tutorialspoint* as explained in the *GWT - Create Application* chapter.
2	Modify *HelloWorld.gwt.xml*, *HelloWorld.css*, *HelloWorld.html* and *HelloWorld.java* as explained below. Keep rest of the files unchanged.
3	Compile and run the application to verify the result of the implemented logic.

Following is the content of the modified module descriptor **src/com.tutorialspoint/HelloWorld.gwt.xml**.

```
<?xml version="1.0" encoding="UTF-8"?>
```


tutorialspoint
SIMPLYEASYLEARNING

```
<module rename-to='helloworld'>
  <!-- Inherit the core Web Toolkit stuff.                        -->
  <inherits name='com.google.gwt.user.User'/>

  <!-- Inherit the default GWT style sheet.                       -->
  <inherits name='com.google.gwt.user.theme.clean.Clean'/>

  <!-- Specify the app entry point class.                         -->
  <entry-point class='com.tutorialspoint.client.HelloWorld'/>

  <!-- Specify the paths for translatable code                    -->
  <source path='client'/>
  <source path='shared'/>

</module>
```

Following is the content of the modified Style Sheet file **war/HelloWorld.css**.

```
body{
   text-align: center;
   font-family: verdana, sans-serif;
}
h1{
   font-size: 2em;
   font-weight: bold;
   color: #777777;
   margin: 40px 0px 70px;
   text-align: center;
}
.gwt-Button{
   font-size: 150%;
```


tutorialspoint
SIMPLYEASYLEARNING

```
   font-weight: bold;
   width:100px;
   height:100px;
}
.gwt-Big-Text{
   font-size:150%;
}
.gwt-Small-Text{
   font-size:75%;
}
```

Following is the content of the modified HTML host file **war/HelloWorld.html** to accommodate two buttons.

```
<html>
<head>
<title>Hello World</title>
   <link rel="stylesheet" href="HelloWorld.css"/>
   <script language="javascript" src="helloworld/helloworld.nocache.js">
   </script>
</head>
<body>

<div id="mytext"><h1>Hello, World!</h1></div>
<div id="gwtGreenButton"></div>
<div id="gwtRedButton"></div>

</body>
</html>
```

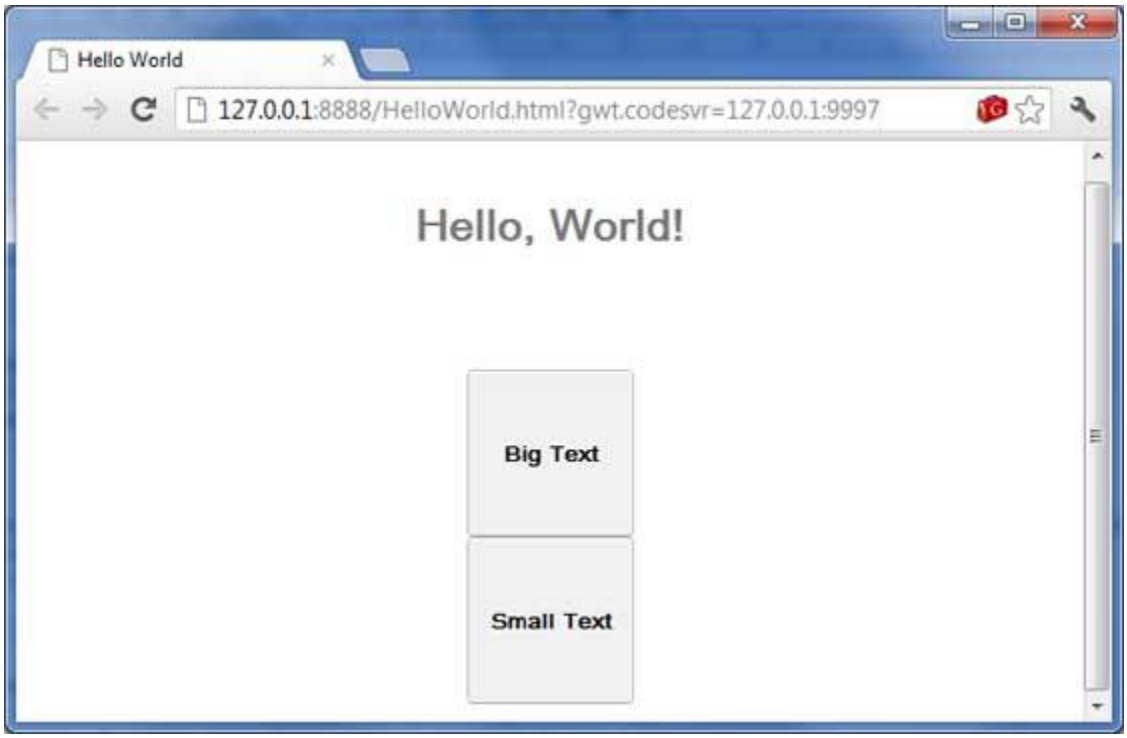
Let us have the following content of Java file **src/com.tutorialspoint/HelloWorld.java** which will take care of adding two buttons in HTML and will apply custom CSS style.

tutorialspoint
SIMPLYEASYLEARNING

```
package com.tutorialspoint.client;

import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.HTML;
import com.google.gwt.user.client.ui.RootPanel;

public class HelloWorld implements EntryPoint {
   public void onModuleLoad() {

   // add button to change font to big when clicked.
   Button Btn1 = new Button("Big Text");
   Btn1.addClickHandler(new ClickHandler() {
      public void onClick(ClickEvent event) {
         RootPanel.get("mytext").setStyleName("gwt-Big-Text");
      }
   });

   // add button to change font to small when clicked.
   Button Btn2 = new Button("Small Text");
   Btn2.addClickHandler(new ClickHandler() {
      public void onClick(ClickEvent event) {
         RootPanel.get("mytext").setStyleName("gwt-Small-Text");
      }
   });

   RootPanel.get("gwtGreenButton").add(Btn1);
   RootPanel.get("gwtRedButton").add(Btn2);
```

```
    }
}
```

Once you are ready with all the changes done, let us compile and run the application in development mode as we did in **GWT - Create Application** chapter. If everything is fine with your application, this will produce following result:



Now try clicking on the two buttons displayed and observe "Hello, World!" text which keeps changing its font upon clicking on the two buttons.

tutorialspoint
SIMPLYEASYLEARNING

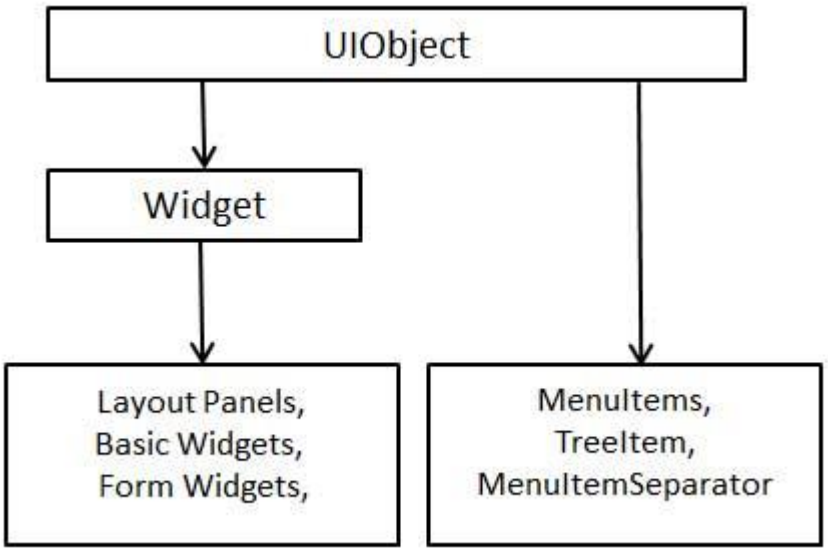
7. GWT - BASIC WIDGETS

Every user interface considers the following three main aspects:

- **UI elements** : These are the core visual elements the user eventually sees and interacts with. GWT provides a huge list of widely used and common elements varying from basic to complex, which we will cover in this tutorial.
- **Layouts:** They define how UI elements should be organized on the screen and provide a final look and feel of the GUI (Graphical User Interface). This part will be covered in Layout chapter.
- **Behavior:** These are events which occur when the user interacts with UI elements. This part will be covered in Event Handling chapter.

GWT UI Elements

The GWT library provides classes in a well-defined class hierarchy to create complex web-based user interfaces. All classes in this component hierarchy has been derived from the **UIObject** base class as shown below:



Every Basic UI widget inherits properties from a Widget class which in turn inherits properties from UIObject. Tree and Menu will be covered in "Complex widgets" tutorial later.

tutorialspoint
SIMPLYEASYLEARNING

S.No.	Widget & Description
1	GWT UIObject Class This widget contains text, not interpreted as HTML using a element, causing it to be displayed with block layout.
2	GWT Widget Class This widget can contain HTML text and displays the html content using a element, causing it to be displayed with block layout.

GWT – UIObject Class

Introduction

The class **UIObject** is the superclass for all user-interface objects. It simply wraps a DOM element, and cannot receive events. It provides direct child classes like Widget, MenuItem, MenuItemSeparator, TreeItem.

- All UIObject objects can be styled using CSS.
- Every UIObject has a primary style name that identifies the key CSS style rule that should always be applied to it.
- More complex styling behaviour can be achieved by manipulating an object's secondary style names.

Class Declaration

Following is the declaration for **com.google.gwt.user.client.ui.UIObject** class:

```
public abstract class UIObject
   extends java.lang.Object
```

Field

Following are the fields for com.google.gwt.user.client.ui.UIObject class:

- **public static final java.lang.String DEBUG_ID_PREFIX** -- The element ID that you specify will be prefixed by the static string DEBUG_ID_PREFIX.

tutorialspoint
SIMPLYEASYLEARNING

Class Constructors

S.No.	Constructor & Description
1	UIObject() This creates a UIObject for the child classes.

Class Methods

S.No.	Method & Description
1	void addStyleDependentName(java.lang.String styleSuffix) Adds a dependent style name by specifying the style name's suffix.
2	void addStyleName(java.lang.String style) Adds a secondary or dependent style name to this object.
3	static void ensureDebugId(Element elem, java.lang.String id) Ensure that elem has an ID property set, which allows it to integrate with third-party libraries and test tools.
4	protected static void ensureDebugId(Element elem, java.lang.String baseID, java.lang.String id) Set the debug id of a specific element.
5	ensureDebugId(java.lang.String id) Ensure that the main Element for this UIObject has an ID property set, which allows it to integrate with third-party libraries and test tools.
6	int getAbsoluteLeft() Gets the object's absolute left position in pixels, as measured from the browser window's client area.

tutorialspoint
SIMPLYEASYLEARNING

7	int getAbsoluteTop() Gets the object's absolute top position in pixels, as measured from the browser window's client area.
8	Element getElement() Gets a handle to the object's underlying DOM element.
9	int getOffsetHeight() Gets the object's offset height in pixels.
10	int getOffsetWidth() Gets the object's offset width in pixels.
11	protected Element getStyleElement() Template method that returns the element to which style names will be applied.
12	java.lang.String getStyleName() Gets all of the object's style names, as a space-separated list.
13	protected static java.lang.String getStyleName(Element elem) Gets all of the element's style names, as a space-separated list.
14	java.lang.String getStylePrimaryName() Gets the primary style name associated with the object.
15	protected static java.lang.String getStylePrimaryName(Element elem) Gets the element's primary style name.
16	java.lang.String getTitle() Gets the title associated with this object.
17	boolean isVisible()

tutorialspoint
SIMPLYEASYLEARNING

	Determines whether or not this object is visible.
18	static boolean isVisible(Element elem) Determines whether element is visible or not.
19	protected void onEnsureDebugId(java.lang.String baseID) Called when the user sets the id using the ensureDebugId(String) method.
20	void removeStyleDependentName(java.lang.String styleSuffix) Removes a dependent style name by specifying the style name's suffix.
21	void removeStyleName(java.lang.String style) Removes a style name.
22	protected void setElement(Element elem) Sets this object's browser element.
23	protected void setElement(Element elem) Sets this object's browser element.
24	void setHeight(java.lang.String height) Sets the object's height.
25	void setPixelSize(int width, int height) Sets the object's size, in pixels, not including decorations such as border, margin, and padding.
26	void setSize(java.lang.String width, java.lang.String height) Sets the object's size.
27	protected static void setStyleName(Element elem, java.lang.String styleName) Clears all of the element's style names and sets it to the given style.

tutorialspoint
SIMPLYEASYLEARNING

28	protected static void setStyleName(Element elem, java.lang.String style, boolean add) This convenience method adds or removes a style name for a given element.
29	void setStyleName(java.lang.String style) Clears all of the object's style names and sets it to the given style.
30	protected static void setStylePrimaryName(Element elem, java.lang.String style) Sets the element's primary style name and updates all dependent style names.
31	void setStylePrimaryName(java.lang.String style) Sets the object's primary style name and updates all dependent style names.
32	void setTitle(java.lang.String title) Sets the title associated with this object.
33	void setVisible(boolean visible) Sets whether this object is visible.
34	static void setVisible(Element elem, boolean visible) Sets whether this element is visible
35	void setWidth(java.lang.String width) Sets the object's width.
36	java.lang.String toString() This method is overridden so that any object can be viewed in the debugger as an HTML snippet.
37	void unsinkEvents(int eventBitsToRemove) Removes a set of events from this object's event list.

tutorialspoint
SIMPLYEASYLEARNING

Methods Inherited

This class inherits methods from the following classes:

- java.lang.Object

GWT – Widget Class

Introduction

The class **Widget** is the base class for the majority of user-interface objects. Widget adds support for receiving events from the browser and being added directly to panels.

Class Declaration

Following is the declaration for **com.google.gwt.user.client.ui.Widget** class:

```
public class Widget
   extends UIObject
      implements EventListener
```

Field

Following are the fields for **com.google.gwt.user.client.ui.Widget** class:

- **public static final java.lang.String DEBUG_ID_PREFIX** -- The element ID that you specify will be prefixed by the static string DEBUG_ID_PREFIX.

Class Constructors

S.No.	Constructor & Description
1	Widget() This creates a Widget for the child classes.

tutorialspoint
SIMPLYEASYLEARNING

Class Methods

S.No.	Method & Description
1	protected <H extends EventHandler> HandlerRegistration addDomHandler(H handler, DomEvent.Type<H> type) Adds a native event handler to the widget and sinks the corresponding native event.
2	protected <H extends EventHandler> HandlerRegistration addHandler(H handler, GwtEvent.Type<H> type) Adds this handler to the widget.
3	protected void delegateEvent(Widget target, GwtEvent<?> event) Fires an event on a child widget.
4	protected void doAttachChildren() If a widget implements HasWidgets, it must override this method and call onAttach() for each of its child widgets.
5	protected void doDetachChildren() If a widget implements HasWidgets, it must override this method and call onDetach() for each of its child widgets.

tutorialspoint
SIMPLYEASYLEARNING

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

tutorialspoint
SIMPLYEASYLEARNING