### Google Web Toolkit

1). The Google Web Toolkit (GWT) is a toolkit to develop Ajax web application with Java.

2). Once the Java code is finished, the GWT compiler translates the Java code into HTML and Javascript.

3). The compiler creates browser specific HTML and JavaScript to support all the major browsers correctly.

4). The generated JavaScript is automatically compressed and obfuscated, decreasing download size and making reverse engineering more difficult.

5). One of the key reasons to use any AJAX framework is to write code that will produce identical results on all browsers, which is fiendishly difficult in raw JavaScript.

6). A very small bootstrap script is also generated, which examines the execution environment and determines which browser-specific application script to load.

7). The bootstrap script is included in a standard HTML page, and when the page loads the script will resolve, download, and run the correct application script.

8). GWT supports a standard set of UI widgets, has build in support for the browser back button and a JUnit based test framework.

9). Image bundles are yet another performance-boosting feature of GWT. Defining an image bundle allows the GWT compiler to package a number of images into a single file which is accessed through a Java object.

10). This reduces the number of network round trips by getting all images for your application in a single file download. The packaging is done by the GWT compiler;

9). These include interoperation with native JavaScript, intelligent support for back button navigation within an application, accessibility support, programming delayed logic, support for JUnit testing, availability of third-party widget libraries, and more..

### Modules, Entry Points and HTML pages

### 1). GWT applications are described as so-called modules.

### 2). A module "modulename" is described by a configuration file "modulename.gwt.xml".

### 3).  Each module can define one or more Entry point classes.

### 4).  An entry point in GWT is the starting point for a GWT application similar to the main method in a standard Java program.

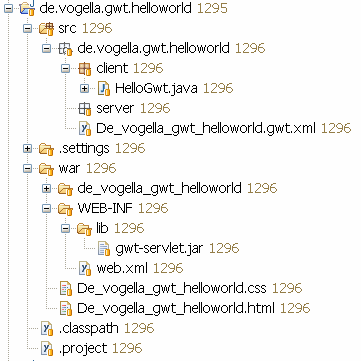
### 5). The HTML page can define "div" containers to which the GWT application can assign UI components.

### 6).  The GWT UI components can be assigned to the body of the HTML page.

### 

**GWT Project Structure**

If you investigate your example you will see the following project structure.



An GWT application consists out of the following parts

1). Module descriptor: XML file which specifies mainly the entry point of the application. It has the name of your module plus .gwt.xml.

The main purpose of the Module descriptor in \*.gwt.xml file is to define your entry-point. It also and defines standard GWT css styles which you are using.

2). war: Under the folder "war" you find the HTML page which contains the GWT application. Additional public resources, e.g. other HTML pages, images or css files should be stored here. Here you find also the standard WEB-INF folder.

3). Source code: under your default package your find a package "client" All Java code which is compiled for the client (Webbrowser) must be stored under client. For the service you would use the package "server".

**Client and Server Communication (RPC)**

1). GWT provides its own remote procedure calls (RPC's) which allow the GWT client to call server-side methods.

2).  The implementation of GWT RPC is based on the servlet technology.

3). GWT allows Java objects to be sent directly between the client and the server; which are automatically serialized by the framework.

4). With GWT RPC the communication is almost transparent for the GWT client and always asynchronous so that the client does not block during the communication.

5). The server-side servlet is usually referred to as a "service" and the a remote procedure call is referred to as "invoking a service." These object can then be used on the client (UI) side.

6). To create a GWT service you need to define the following:

* An interface which extends RemoteService that lists the service methods.
* An implementation - implements the interface and extends the RemoteServiceServlet.
* Define an asynchronous interface to your service which will be used in the client code

### 7). Create project and domain model: The data model class implements "Serializable" because GWT requires that all classes that are involved in a server and client communication have to implement the interface "Serializable".

8). **Interface:** The client server communication is based on an interface which defines the possible communication methods.

Create the following interface. By extending the RemoteService interface the GWT compiler will understand that this interface defines an RPC interface. *The annotation defines the URL for the service.* This must match the entry we will later do in "web.xml".

This interface needs to be available on the client side, therefore it must be placed in the client package

@RemoteServiceRelativePath("userService")

public interface MyUserService extends RemoteService {

List<MyUser> getUserList();

MyUser getUser(String id);

}

### 9).GWT uses asynchronous communication therefore you also need to create the asynchronous version of this interface. The name of this interface must be the interface name concatenated with "Async".

### The implementation of this asynchronous interface will be automatically created by the GWT compiler.

### public interface MyUserServiceAsync {

### void getUserList(AsyncCallback<List<MyUser>> callback);

### void getUser(String id, AsyncCallback<MyUser> callback);

### }

### 10). Create the server

For the server implementation create a package "de.vogella.gwt.helloworld.server". Create the following class which extends RemoteServiceServlet.

### public class MyUserServiceImpl extends RemoteServiceServlet implements

### MyUserService {

### Implemeting methods….

### }

### 11). Callback : To receive a callback a class needs to be implemented which can then react on a failure or success in the communication. The interface AsyncCallback defines these two methods: "OnSuccess" and "OnFailure".

### Keep in mind that the server communication is asynchronous. If you call your server your GWT application will continue and at some undefined point in time it will receive the data from the server. Make sure that you do not assume in your coding that the call to the server is finished.

*12). Google offers a cloud computing infrastructure called*[*Google App Engine*](http://appengine.google.com/)*(App Engine) for creating and running web applications. App Engine allows the dynamic allocation of system resources for an application based on the actual demand.*

13). Serializability of a class is enabled by the class implementing the java.io.Serializable interface. Classes that do not implement this interface will not have any of their state serialized or deserialized. All subtypes of a serializable class are themselves serializable. The serialization interface has no methods or fields and serves only to identify the semantics of being serializable.

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

<module rename-to=*'cypers'*>

<!-- 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.standard.Standard'* />

<!-- <inherits name='com.google.gwt.user.theme.chrome.Chrome'/> -->

<!-- <inherits name='com.google.gwt.user.theme.dark.Dark'/> -->

<inherits name=*'eu.maydu.gwt.validation.ValidationLibrary'* />

<inherits name=*'com.bradrydzewski.gwt.calendar.Calendar'* />

<inherits name=*'com.bradrydzewski.gwt.calendar.theme.ical.iCal'* />

<inherits name=*'com.allen\_sauer.gwt.dnd.gwt-dnd'* />

<!-- Specify the app entry point class. -->

<entry-point class=*'com.ignis.cypers.client.Cypers'* />

<!-- Specify the paths for translatable code -->

<source path=*'client'* />

<source path=*'model'* />

<source path=*'shared'* />

</module>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<!-- The HTML 4.01 Transitional DOCTYPE declaration-->

<!-- above set at the top of the file will set -->

<!-- the browser's rendering engine into -->

<!-- "Quirks Mode". Replacing this declaration -->

<!-- with a "Standards Mode" doctype is supported, -->

<!-- but may lead to some differences in layout. -->

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />

<!-- -->

<!-- Consider inlining CSS to reduce the number of requested files -->

<!-- -->

<link type="text/css" rel="stylesheet" href="Cypers.css">

<!-- -->

<!-- Any title is fine -->

<!-- -->

<title>CYPERS</title>

<!-- -->

<!-- This script loads your compiled module. -->

<!-- If you add any GWT meta tags, they must -->

<!-- be added before this line. -->

<!-- -->

<script type="text/javascript" language="javascript" src="cypers/cypers.nocache.js"></script>

</head>

<!-- -->

<!-- The body can have arbitrary html, or -->

<!-- you can leave the body empty if you want -->

<!-- to create a completely dynamic UI. -->

<!-- -->

<body>

<!-- OPTIONAL: include this if you want history support -->

<iframe src="javascript:''" id="\_\_gwt\_historyFrame" tabIndex='-1' style="position:absolute;width:0;height:0;border:0"></iframe>

</body>

</html>

## GWT Panels - Summary

#### Panels Are Layout Widgets

Panels are widgets which can contain other widgets. As such, they are used for layout and only rarely have data associated with them directly - the DisclosurePanel for example, can have data in the header, and the TabPanel's tabs. Some are based on an HTML table element and some are based on a div. Here's a table of which are which.

|  |  |  |  |
| --- | --- | --- | --- |
| **Panel Name** | **Type** | **Extends** | **Lets you** |
| AbsolutePanel | DIV | ComplexPanel | Position widgets absolutely |
| CellPanel | TABLE | ComplexPanel | [Abstract] Subclass your own cell panels |
| ComplexPanel | - | Panel | [Abstract] Subclass panels with more than one widget |
| DeckPanel | DIV | ComplexPanel | Display widgets - one visible at a time |
| DisclosurePanel | TABLE | Composite | Show/hide a details pane |
| DockPanel | TABLE | CellPanel | Add widgets N, S, E, W around a central cell |
| FlowPanel | DIV | ComplexPanel | Add widgets as if to a normal DIV |
| FocusPanel | DIV | SimplePanel | Add focus to non-focusable widgets |
| FormPanel | DIV | SimplePanel | Submit a form to a server |
| Frame | IFRAME | - | Add an IFRAME to the application |
| HorizontalPanel | TABLE | CellPanel | Add a chain of cells horizontally |
| HorizontalSplitPanel | DIV | SplitPanel | Move the border between two cells |
| HTMLPanel | DIV |  | Add HTML, then access the elements that have IDs |
| Panel | - | Widget | [Abstract] Base for all panels |
| ScrollPanel | DIV | SimplePanel | Stack child widgets, displaying contents of only one |
| StackPanel | TABLE | SimplePanel | Add a vertical chain of widgets |
| SimplePanel | DIV | Panel | [Abstract] One-widget panel |
| TabPanel | TABLE | VerticalPanel | Add virtual card-index dividers |
| VerticalPanel | TABLE | CellPanel | Add a chain of cells vertically |
| VerticalSplitPanel | DIV | SplitPanel | Move the border between two cells |

## GWT Widgets - Summary

#### Widgets For Getting And/Or Displaying Data

The widgets in this section are mainly used to input or output data. You could, for example, use the FlexTable to lay out your screen, but more usually you would use it for tables of data.

|  |  |
| --- | --- |
| **Widget Name** | **Description** |
| Button | The user clicks it, the application does something |
| CheckBox | Displays and/or gets input for a true/false condition |
| Composite | [Abstract]A class to let you build your own widgets |
| FileUpload | Allows the user to upload files. Must be inside a FormPanel |
| FlexTable | A table with varying numbers of cells in a row. Expands as required. |
| Grid | A fixed-size table of cells which can hold text, HTML or widgets |
| Hidden | A hidden text field for use on a form |
| HTML | A widget which holds raw HTML |
| Hyperlink | A widget which lets the user jump to somewhere else in the application |
| Image | Holds an image for display |
| Label | A one-line text box |
| ListBox | A list of available selections as a list or dropdown |
| MenuBar | A dropdown menu bar holding MenuItems or another MenuBar |
| MenuItem | An item in a MenuBar - it either fires a command, or cascades into a submenu |
| PopupPanel | An alert panel within the appliation |
| PushButton | A button which can be styled |
| RadioButton | One of a set of mutually exclusive options |
| RichTextArea | Like a text area but with text fomatting |
| SuggestBox | A TextBox which has a dropdown list of suggestions |
| TabBar | A kind of horizontal menu you can make look like tabs |
| TextArea | A multi-line box for entering text |
| TextBox | A single-line box for entering text |
| ToggleButton | A click to latch/click to release button |
| Tree | A hierarchical node-based display widget |

## GWT Listeners - Summary

|  |  |
| --- | --- |
| ChangeListener | Text has changed |
| ClickListener | Something has been clicked |
| EventListener | A catch-all for when ANYthing happens |
| FocusListener | Something has got or lost focus |
| HistoryListener | The user/program has navigated history, or the program has added to the history |
| KeyboardListener | Keypress, or down or up |
| LoadListener | Something has loaded |
| MouseListener | The mouse has done something in your widget (yuk!) |
| MouseWheelListener | The mouse wheel has moved |
| PopupListener | A popup has popped up or down |
| ScrollListener | Scrolling has occurred |
| TableListener | A cell has been clicked |
| TreeListener | A tree item has been selected or its state has changed |
| WindowCloseListener | Window closing - Are you sure? Really sure? Positive? Please don't let me die! |
| WindowResizeListener | The window is a different size |