## Why GWT?

### GWT is a framework for writing AJAX interactive web applications.

### Write Java code, not JavaScript

One of the most unique features of GWT is that you code in Java. While in development, you can run your Java code in a special "hosted mode" browser with all of the mature, sophisticated Java debugging tools you always have available. For deployment, this Java code is run through a compiler which translates this Java code to cross-browser JavaScript.

Although this is significant, especially in shops with a lot of Java experience, I don't believe this is a truly killer feature of GWT. The important thing is that the developer is shielded from the complexities of writing JavaScript code that works in all browsers. Other AJAX frameworks provide a layer of JavaScript that provides this abstraction. GWT does this by translating standard Java code, but *how* this is abstracted is less important than the simple fact that it *is* abstracted.

### Simple RPC mechanism

GWT has its own mechanism for remote procedure calls between the browser and the server. This works a lot like vanilla Java RMI - define interfaces and implementations for your server functions, and code will be generated to allow your client code to call them as if they were simple local methods. This blows away all the work of defining XML or JSON data formats for requests and responses. Just code the function for the server, call the function (still in Java) from the client code, and all of the marshalling, unmarshalling, network communication, etc. is done for you.

GWT also supports other RPC methods, such as XML or JSON over HTTP. You can write server code as standard SOAP or REST web services and GWT will be able to use them directly. This will take longer to code than the GWT RPC mechanism, but allows you to use legacy web services or services intended to also be used by non-GWT clients. A good middle-ground approach may be to prototype using GWT RPC, and re-implement as web services later.

### Optimized cross-browser JavaScript

### browsers

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. Why roll your own code to handle all of those quirks, when a bunch of framework authors have already done it for you? GWT handles this, just like any good framework. As a bonus, the generated JavaScript is automatically compressed and obfuscated, decreasing download size and making reverse engineering more difficult.

Download sizes are kept even smaller by a unique GWT feature, namely...

### Deferred binding

Writing code that works on any browser is great. But there is a cost - the framework has to include code for every browser. You may be viewing the page in IE, but you also downloaded code specific to Firefox, Safari, and Opera, which will never be executed. Deferred binding is the GWT solution to this dilemma.

The first step in deferred binding happens during compilation. The GWT compiler creates a different JavaScript file for each browser type, containing all of your application code optimized for just that one browser. A very small bootstrap script is also generated, which examines the execution environment and determines which browser-specific application script to load. 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.

There is also provision for taking advantage of caching. The bootstrap file is intended to never be cached (and is helpfully suffixed with "nocache.js" as a reminder). The implementation scripts are intended to be cached, and have file names that are a hash code of their contents. This means that if you re-compile, but there are no changes in a particular file, it will have the same name and will allow the browser to use the already cached version. If the file changes, it's filename will change and therefore force a reload when it is next requested. All of this is done without any developer help (the generated bootstrap script deals with the filenames automatically). Pretty neat, huh?

I've only talked about deferring binding based on browser type so far, but this is a general mechanism and you can define other context-dependent variances as well. The most common use of this is...

### Localization

Localizing a GWT application is very straightforward and very similar to how it is done in standard Java applications. In the simple case, just create properties files with locale suffixes, access them by key through a GWT interface, and the locale-specific string will be used. The cool part is that GWT uses the deferred binding mechanism to make sure that only those properties for the current user's locale are ever downloaded.

As with browser-specific code, this starts at compile time. For each combination of browser AND locale, an application script is generated. So, if you have, say, 4 browsers and 3 different locales, you will have 12 files generated - such as FireFox in English, FireFox in French, IE in English, etc. The bootstrap file will examine the client environment and load the application file specific to the detected browser and locale.

### Image bundles

It should be obvious by now that the Google team put a lot of effort into ensuring the highest possible performance.

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.

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;

all the developer has to do is define an annotated interface with a function to access each image.

### Embeds well in existing sites

Some frameworks work best only if they are in charge of the whole page. GWT "plays well with others", in that it can generate entire pages or elements within a page equally well. The GWT scripts work by creating HTML elements inside a specified element on a page during the onLoad() event. To embed a GWT application in an existing page, you therefore only need to include the script file and add an id to the element you want to contain the application (which can be a div, a td, the body, anything). Couldn't be easier.

### Conclusion

That's a lot of good reasons to choose GWT, and that's just scratching the surface. There are many other features that make GWT a compelling choice. 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. GWT is also under continuous development, with the upcoming 1.6 version to include improvements like a faster hosted mode server, faster string handling, better compiler performance, and easier deployment to standard JEE WAR files. The Google team has done a very good job of providing a no-compromise framework that provides a fast, rich, and consistent user experience while keeping the developer focused on the application rather than the technology. It's definitely worth taking for a test drive.

Now, if I could just get that song out of my head...