**Internationalization Techniques**

GWT offers multiple internationalization techniques to afford maximum flexibility to GWT developers and to make it possible to design for efficiency, maintainability, flexibility, and interoperability in whichever combinations are most useful.

* [**Static string internationalization**](http://www.gwtproject.org/doc/latest/DevGuideI18n.html#DevGuideStaticStringInternationalization)  
  A family of efficient and type-safe techniques that rely on strongly-typed Java interfaces, [properties files](http://www.gwtproject.org/doc/latest/DevGuideI18n.html#DevGuidePropertiesFiles), and code generation to provide locale-aware messages and configuration settings. These techniques depend on the interfaces [Constants](http://www.gwtproject.org/javadoc/latest/com/google/gwt/i18n/client/Constants.html), [ConstantsWithLookup](http://www.gwtproject.org/javadoc/latest/com/google/gwt/i18n/client/ConstantsWithLookup.html), and [Messages](http://www.gwtproject.org/javadoc/latest/com/google/gwt/i18n/client/Messages.html).
* Static string initialization requires very little overhead at runtime and therefore is a very efficient technique for translating both constant and parameterized strings. It is also the simplest technique to implement. Static string internationalization uses standard Java properties files to store translated strings and parameterized messages, then implements strongly-typed Java interfaces to retrieve their values.
* [**Dynamic string internationalization**](http://www.gwtproject.org/doc/latest/DevGuideI18n.html#DevGuideDynamicStringInternationalization)  
  A simple and flexible technique for looking up localized values defined in a module's [host page](http://www.gwtproject.org/doc/latest/DevGuideOrganizingProjects.html#DevGuideHostPage) without needing to recompile your application. This technique is supported by the class [Dictionary](http://www.gwtproject.org/javadoc/latest/com/google/gwt/i18n/client/Dictionary.html).
* Dynamic string internationalization is slower than static string internationalization, but is very flexible. Applications using this technique look up localized strings in the module's host page; therefore, they do not need to be recompiled when you add a new locale. If you need to integrate a GWT application with an existing server-side localization system, dynamic string internationalization is the option to consider.
* **Extending or implementing Localizable**  
  Provides a method for internationalizing sets of algorithms using locale-sensitive type substitution. This is an advanced technique that you probably will not need to use directly, although it is useful for implementing complex internationalized libraries. For details on this technique, see the [Localizable](http://www.gwtproject.org/javadoc/latest/com/google/gwt/i18n/client/Localizable.html)class documentation.
* The most powerful technique is to implement the Localizable interface. Implementing Localizable allows you to go beyond simple string substitution and create localized versions of custom types. It's an advanced internationalization technique that you probably won't have to use very often.

**Process Overview: Static String Internationalization**

The process you'll follow for Static String Internationalization is straightforward.

1. First, you'll implement two Java interfaces:
   * one for string constants, the GWT Constants interface (StockWatcherConstants.java)
   * one for parameterized messages, the GWT Messages interface (StockWatcherMessages.java)

These interfaces use annotations to specify the default translation.

1. Then, for each new language you're supporting, you'll create two Java properties files:
   * one for string constants (StockWatcherConstants\_de.properties)
   * one for parameterized messages (StockWatcherMessages\_de.properties)
2. Finally, you'll replace all the strings hardcoded in the Java source code with method calls to the appropriate interface.

**Tip:** GWT provides a command-line tool, [i18nCreator](http://www.gwtproject.org/doc/latest/RefCommandLineTools.html#i18nCreator), that automates the creation of Java interfaces to access strings in properties files. This tool comes in handy, especially if you have existing localized properties files you'd like to reuse.

### Implement the Constants Interface

First create the Java interface (StockWatcherConstants) that accesses the properties files which hold each translation. The interface uses annotations to specify the default translation. This interface implements the GWT [Constants](http://www.gwtproject.org/javadoc/latest/com/google/gwt/i18n/client/Constants.html)interface. This interface is bound automatically to any StockWatcherConstants\*.properties files you create because of its name.

The StockWatcherConstants interface contains methods for each of the constants in the properties files. At runtime, when one of these methods is called, the return value comes from whichever properties file that corresponds with the locale. (We'll show you how to set the locale in a minute.) If no locale is set, StockWatcher uses the default translation specified by the annotations. For example, if the locale is set to German, the stockWatcher method will return AktieWatcher; if no locale is set, the stockWatcher method will return StockWatcher.

#### Create StockWatcherConstants

1. In the client subpackage, create an interface and name it StockWatcherConstants.

In Eclipse, in the Package Explorer pane, select the package  
com.google.gwt.sample.stockwatcher.client

From the Eclipse menu bar, select File > New > Interface

Eclipse opens a New Java Interface window.

1. Fill in the New Java Interface window.

At Name enter StockWatcherConstants

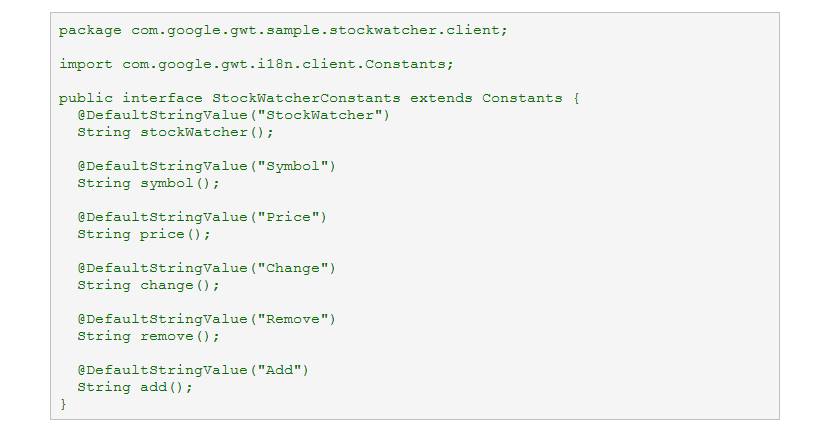
Accept the defaults for the other fields.

Press Finish

Eclipse creates stub code for the StockWatcherConstants interface.

1. Replace the stub with following code.

Notice the use of annotations to set default values.



**Implementation Note:** GWT provides another interface ([ConstantsWithLookup](http://www.gwtproject.org/javadoc/latest/com/google/gwt/i18n/client/ConstantsWithLookup.html)) which is similar to Constants except that it also contains methods for looking up a localized string dynamically by name at runtime.

### Create the German translation of constants

#### Encoding for international character sets

When you internationalize your application's interface, keep in mind that the languages you support may contain characters not in the ASCII character set. Therefore, both in the HTML host page (StockWatcher.html), and the Java properties files that contain the translations, you must set the encoding to UTF-8.

1. Check the encoding for the application host page.

Open StockWatcher.html.

If the encoding is not already set to UTF-8, copy this code into the <head> element.

<meta http-equiv="content-type" content="text/html; charset=UTF-8">

#### Create StockWatcherConstant\_de.properties

1. In the client subpackage, create a Java properties file.

At Enter or select the parent folder, select  
StockWatcher/src/com/google/gwt/sample/stockwatcher/client

At File name, enter  
StockWatcherConstants\_de.properties

1. Change the encoding of the file to UTF-8.

Select the file and then from the Eclipse menu bar, select File > Properties or right-click.

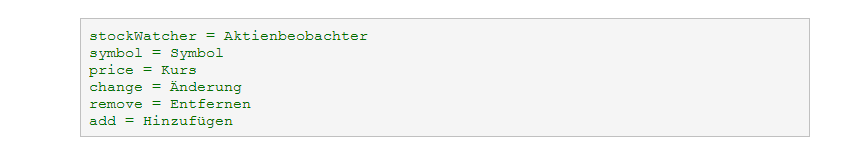
Eclipse opens the Properties window.

At Text file encoding, select Other UTF-8. Apply and Save the change.

**Note:** Depending on your Eclipse configuration, when you apply the changes, you might get this warning: UTF-8 conflicts with the encoding defined in the content type (ISO-8859-1). Do you wish to set it anyway? You can ignore the warning and apply the change.

1. Add the mappings for the static text in the German user interface.

Copy and paste the following text into the StockWatcherConstant\_de.properties file.



##### Note: Suffixing a properties file

If you've never dealt with internationalization before, you may be wondering why the \_de suffix is appended to German properties file. The suffix \_de is the standard language tag for the German language (**De**utsch). Languages tags are abbreviations that indicate a document or application's locale. In addition to specifying the language, they can also contain a subtag indicating the region of a locale. For example, the language tag for French-speaking Canada is fr\_CA.

In GWT, properties files indicate the locale with a language code suffix (just like Java resource bundles). The exception is the properties file for the default locale. When no locale is explicitly set at runtime, the properties file with no language code suffix is used. For StockWatcher, you've specified the default translation with annotations instead of using a default properties file.

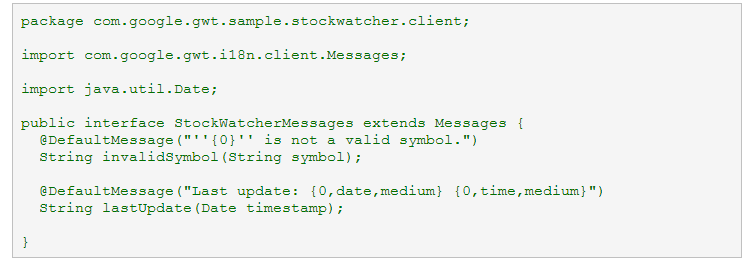
### Implement the Messages interface

First create the Java interface (StockWatcherMessages). It accesses the properties files which hold the translations of each parameterized message. This interface implements the GWT [Messages](http://www.gwtproject.org/javadoc/latest/com/google/gwt/i18n/client/Messages.html) interface. Unlike the StockWatcherConstants interface, the methods in this interface contain parameters. When you call these methods, the arguments you pass will replace the placeholders you left in the strings in the properties files. This interface is bound automatically to any StockWatcherMessages\*.properties files you create because of its name.

#### Internationalizing date formats

Parameterized messages are not limited to pop-up alerts and error messages. Any place in the application where you set text on a Label widget has the potential to be a parameterized message. For example, in StockWatcher the timestamp is a parameterized message; not only do you pass in the value of the date, but the date format can vary by locale.

1. In the client subpackage, create an interface and name it StockWatcherMessages.
2. Replace the stub with following code.



#### Tips on formatting parameterized messages

##### Specifying the number of arguments

Notice that the message strings all have an {0} embedded in them. These are placeholders that will be replaced at runtime by arguments passed to our StockWatcherMessages interface methods.

If you have a string that needs more than one argument, number the placeholders sequentially.  
For example: myString = First parm is {0}, second parm is {1}, third parm is {2}.

##### Handling quoted text

If your messages contains single quotes ('), as many of those in StockWatcher do, you'll need to replace them with two consecutive single quotes in the Java properties files. In general, the formatting rules for GWT messages are the same that apply to Java's [MessageFormat](http://java.sun.com/j2se/1.5.0/docs/api/java/text/MessageFormat.html) class.

### Create the German translation of parameterized messages

1. Create a Java properties file.

At Enter or select the parent folder, select  
StockWatcher/src/com/google/gwt/sample/stockwatcher/client

At File name, enter  
StockWatcherMessages\_de.properties

1. Change the encoding of the file to UTF-8.
2. Add the mappings for the parameterized text in the English user interface.

Copy and paste the following text into the StockWatcherMessages\_de.properties file.

lastUpdate = Letzte Aktualisierung: {0,date,medium} {0,time,medium}

invalidSymbol = ''{0}'' ist kein gültiges Aktiensymbol.

### Replacing hardcoded strings with generated localized ones

The next step in internationalizing StockWatcher is to replace all hardcoded strings within the source code with method calls to one of the two new interfaces.

#### Replacing strings hardcoded in the HTML host page

Currently the StockWatcher application has one string that isn't generated programmatically: the title, StockWatcher. It's an HTML <h1> heading in the host page (StockWatcher.html).

In the Build a Sample GWT Application tutorial we wanted to show you that is possible to mix static HTML elements with those generated by StockWatcher on the same page. It was also a fast and easy way of putting static text around the stock table. However, now that you are internationalizing StockWatcher, you can see that this is not the most flexible strategy.

An easy way to generate this heading is by replacing the text inside the <h1> element with a GWT Label widget and calling its setText(String) method. Remember, GWT widgets cannot be embedded directly into the HTML host page; so first wrap it with a Root panel.

1. Open StockWatcher.html.

Associate a Root panel with the <h1> heading by assigning it an id of "appTitle".

Delete the text in the <h1> heading.

<body>

<img src="images/GoogleCode.png"/>

~~<h1>StockWatcher</h1>~~

<h1 id="appTitle"></h1>

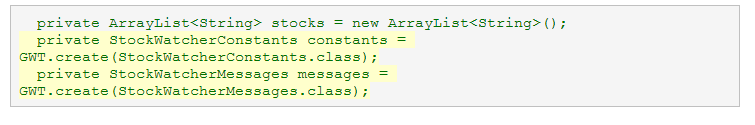
Now, you should be able to set all of StockWatcher's localized strings at runtime.

#### Replacing strings set programmatically

Go through the StockWatcher class and replace all the strings that are hardcoded text.

1. Create instances of the StockWatcherConstants and StockWatcherMessages interfaces.

In the StockWatcher class, add the following pair of instance fields.

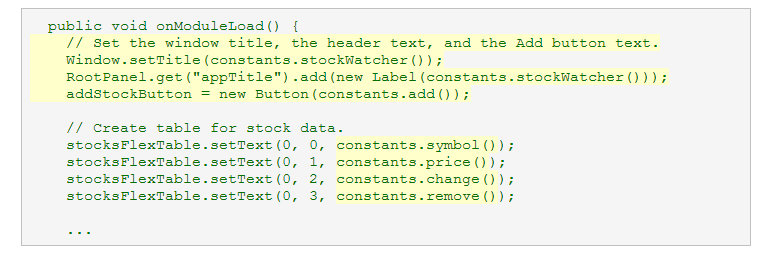


1. Because these are interfaces, not classes, you can't instantiate them directly. Instead, you use the GWT.create(Class) method. Then you'll be able to use these interfaces' accessor methods to retrieve the appropriate strings.
2. Eclipse flags GWT and suggests you include the import declaration.

import com.google.gwt.core.client.GWT;

1. Replace all the hardcoded strings with method calls to the constants class.

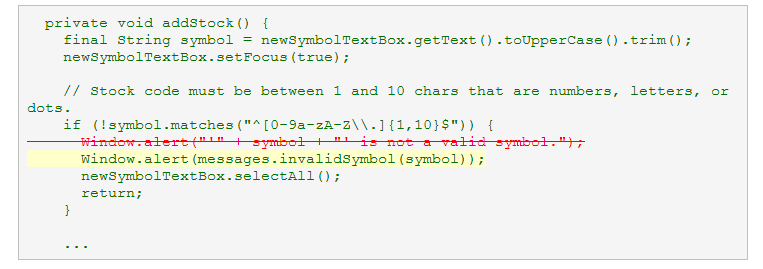
Get the values of the window title, the application title, the Add Stock button, and the column headers of the flex table from the constants properties files.



1. Replace the parameterized error message.

In the addStock method, replace the alert message for an invalid stock code entry.

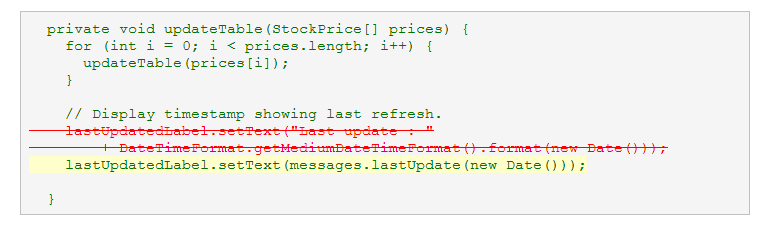
Change:



5. Move the logic for determining the date format to the Messages interface.

In the updateTable(StockPrice[] prices) method, replace the variable timestamp with a call to its lastUpdate method.

Change:



## 3. Localizing StockWatcher

At this point you have created two localized versions of StockWatcher's user interface. But how does GWT know which one to load at runtime? GWT uses client properties to produce customized JavaScript compilations of your GWT application using a mechanism called deferred binding. To pick the correct localized version of StockWatcher to serve at runtime, GWT evaluates the client property, *locale*.

### Internationalization and Deferred Binding

You saw in the Build a Sample GWT Application tutorial that GWT uses deferred binding to generate different permutations of your application, each one targeting a different web browser. At runtime, the GWT bootstrap code delivers the appropriate permutation depending on which browser the end-user is using. These browser-specific compilations are created because *user agent* is a GWT client property. In the same way, GWT represents the locale as a client property. This means that the GWT compiler will generate custom versions of internationalized applications representing each supported locale.

When there are multiple client properties, GWT generates a unique compilation for every combination of possible client property values. So, for example, if GWT supports 5 web browsers and you translate an application into 4 different languages, the GWT compiler produces a total of 20 different versions of your application. However, each user of your application is served only the code in the permutation matching his or her particular combination of web browser and locale.

### Identifying StockWatcher's supported locales

You tell the GWT compiler that StockWatcher now supports the German (de) locale by extending the set of values of the client property, *locale*.

1. Identify German as a supported language.

Open StockWatcher.gwt.xml and add the following property.

<entry-point class='com.google.gwt.sample.stockwatcher.client.StockWatcher'/>

<extend-property name="locale" values="de"/>

</module>

1. Refresh StockWatcher in development mode.

The English language version loads by default.

1. Load the German version.

To the end of the URL, append &locale=de

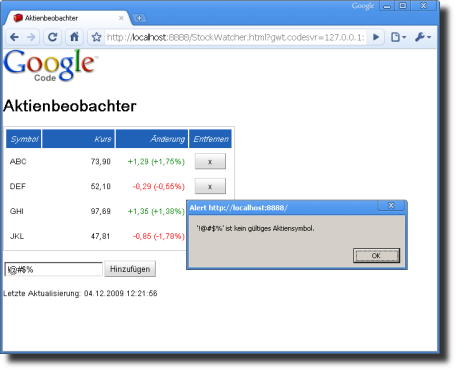
http://localhost:8888/StockWatcher.html?gwt.codesvr=127.0.0.1:9997&locale=de

All the constants should display in German.

1. Enter a stock code with an invalid character.

The error message should display in German.

Notice, too, that the date and currency formats are localized.



1. Compile StockWatcher and open it in production mode.

The web browser displays StockWatcher's default interface.

1. Test the German interface.

Append the locale to the URL ?locale=de

The web browser displays StockWatcher's German interface.

1. Look at the files generated.

You should see files for twice as many permutations as you did before you created the German-language interface.

### Determining the user's locale

At runtime, how do you determine the user's locale? You might do as many websites do, and present the user with a list of languages or locales to choose from manually. You could also have the web server check the Accept-Language field in the browser's HTTP request to determine the correct locale. If you do this, however, be sure to provide a way for the user to override the value in the Accept-Language field and select their language preference.

### Setting the locale

Now that StockWatcher is internationalized, how does GWT know which locale to load at runtime? The answer is that it uses the value of the client property, locale. You can set a this client property two ways:

* Add an HTML <meta> tag to the <head> of the application host page, containing the property name and value:  
  <meta name="gwt:property" content="locale=de">
* Append the client property value to the query string of the URL:  
  http://www.example.org/myapp.html?locale=de

If you specify a client property (such as locale) in both a <meta> tag and the URL, the URL value takes precedence.

### Preserving the locale across multiple browser pages

The locale settings for a GWT module apply only for that particular instance of that particular module. This means that if your application contains links to other GWT host pages or non-GWT web pages on your site, the locale setting does not carry over to those pages. Thus, if you want to preserve the user-specified locale, you'll either need to pass the locale in the query string of all links in your GWT application, or you'll need to store the locale setting somewhere on the server, which can then insert the appropriate <meta> tag into the host pages of any loaded GWT modules.

## More about internationalization

At this point you've used Static String Internationalization to generate a German-language permutation of your GWT application. You've implemented localization so that the appropriate permutation loads based on the user's locale.

To learn more about all three internationalization techniques, the GWT I18N module, or how to use GWT i18nCreator to reuse existing localized properties files, see the Developer's Guide, [Internationalization](http://www.gwtproject.org/doc/latest/DevGuideI18n.html).