Google Web Toolkit (GWT)

Sang Shin
Java Technology Architect & Evangelist
Sun Microsystems, Inc.
sang.shin@sun.com
www.javapassion.com

Disclaimer & Acknowledgments

- Even though Sang Shin is a full-time employee of Sun Microsystems, the contents here are created as his own personal endeavor and thus does not necessarily reflect any official stance of Sun Microsystems on any particular technology
- Most of the slides in this presentations are created from the contents from Google Web Toolkit (GWT) website
  - http://code.google.com/webtoolkit/

Source: http://code.google.com/webtoolkit/

Agenda

- What is & Why GWT?
- Building User interface
  - GWT Widgets
  - Event Handling
  - Styling
- Remote Procedural Call (RPC)
  - Steps for building GWT RPC application
  - Serializable types
  - Handling exceptions

Source: http://code.google.com/webtoolkit/

Agenda

- JavaScript Native Interface (JSNI)
  - Motivation for JSNI
  - Accessing native JavaScript code from Java code
  - Accessing Java methods and fields from native JavaScript code
- GWT Project
  - GWT Module configuration
  - Deployment

Source: http://code.google.com/webtoolkit/

What is and Why GWT?

What is GWT?

- Java software development framework that makes writing AJAX applications easy
- Let you develop and debug AJAX applications in the Java language using the Java development tools of your choice
  - NetBeans or Eclipse
- Provides Java-to-JavaScript compiler and a special web browser that helps you debug your GWT applications
  - When you deploy your application to production, the compiler translates your Java application to browser-compliant JavaScript, CSS, and HTML

Source: http://code.google.com/webtoolkit/

Two Modes of Running GWT App

- Hosted mode
  - Your application is run as Java bytecode within the Java Virtual Machine (JVM)
  - You will typically spend most of your development time in hosted mode because running in the JVM means you can take advantage of Java's debugging facilities
- Web mode
  - Your application is run as pure JavaScript and HTML, compiled from your original Java source code with the GWT Java-to-JavaScript compiler
  - When you deploy your GWT applications to production, you deploy this JavaScript and HTML to your web servers, so end users will only see the web mode version of your application

Source: http://code.google.com/webtoolkit/

Why Use Java Programming Language for AJAX Development?

- Static type checking in the Java language boosts productivity while reducing errors.
- Common JavaScript errors (typos, type mismatches) are easily caught at compile time rather than by users at runtime.
- Code prompting/completion is widely available
- Automated Java refactoring is pretty snazzy these days.
- Java-based OO designs are easier to communicate and understand, thus making your AJAX code base more comprehensible with less documentation.

Source: http://code.google.com/webtoolkit/

Why GWT?

- No need to learn/use JavaScript language
  - Leverage Java programming knowledge you already have
- No need to handle browser incompatibilities and quirks
  - GWT handles them for you
  - Browser history
  - Forward/backward buttons
- No need to learn/use DOM APIs
  - Use Java APIs
- No need to build commonly used Widgets
  - Most of them come with GWT

Source: http://code.google.com/webtoolkit/

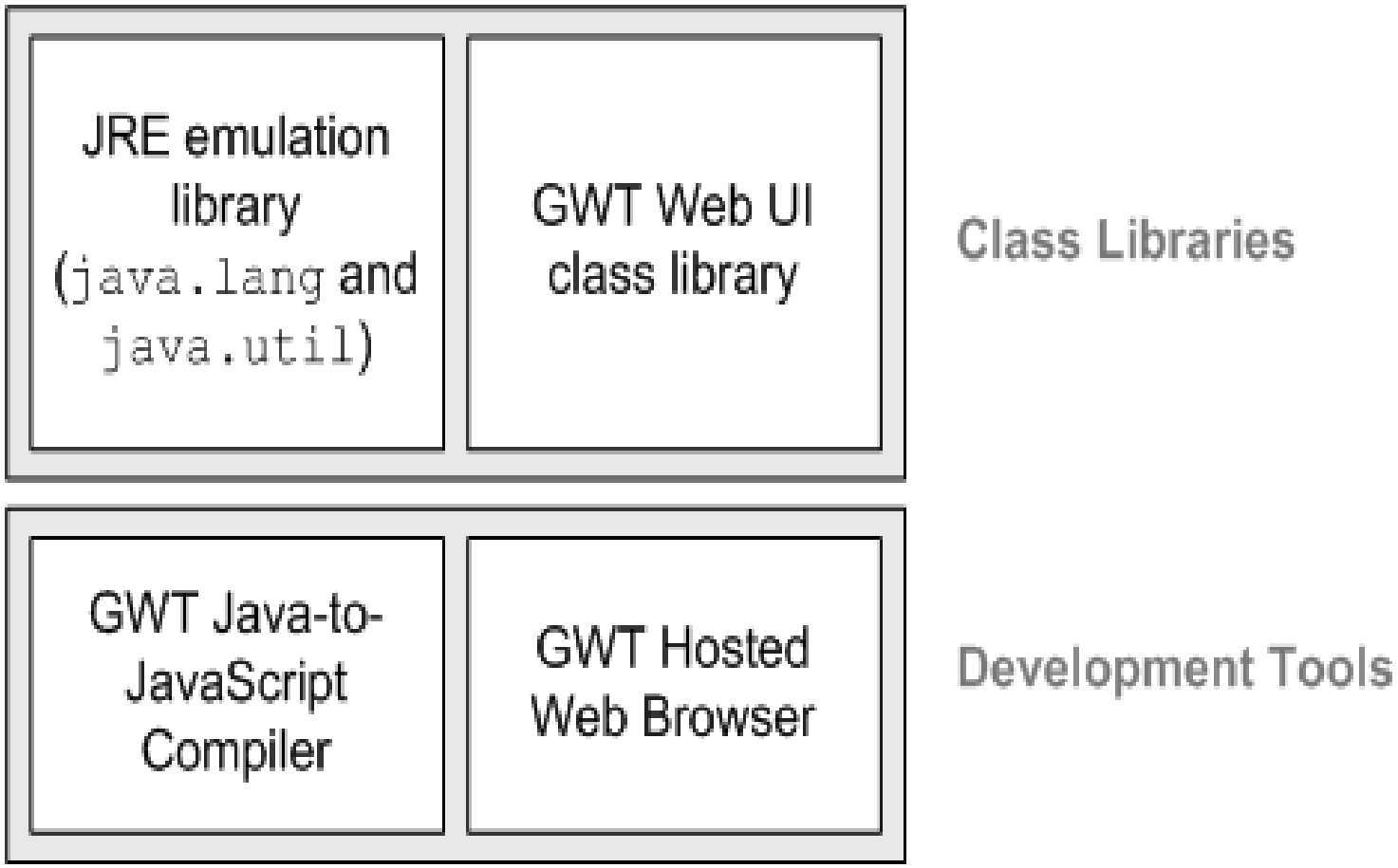
Why GWT?

- Leverage various tools of Java programming language for writing/debugging/testing
  - For example, NetBeans or Eclipse
- JUnit integration
  - GWT's direct integration with JUnit lets you unit test both in a debugger and in a browser and you can even unit test asynchronous RPCs
- Internationalization
  - GWT internationalization support provides a variety of techniques to internationalize strings, typed values, and classes

Source: http://code.google.com/webtoolkit/

GWT Architecture

GWT Architecture



Source: http://code.google.com/webtoolkit/

GWT Architectural Components

- GWT Java-to-JavaScript Compiler
  - translates the Java programming language to the JavaScript programming language
- GWT Hosted Web Browser
  - 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
- JRE emulation library
  - GWT contains JavaScript implementations of the most widely used classes in the Java standard class library
- GWT Web UI class library

Source: http://code.google.com/webtoolkit/

The GWT Java To Javascript Compiler

- Parses the original Java code
  - Full parser, almost all Java constructs will be parsed correctly.
- Generates a full Abstract Syntax Tree (AST)
  - GWT's compiler isn't a parlor trick, it's a real code parser and cross-language compiler
- Performs optimization, dead-code elimination, dynamic analysis, and other code fixups
  - The generated JS is fast and efficient, sometimes faster than what you might write yourself
- Does not (yet) handle generics and erasure

GWT Javascript Code Generation

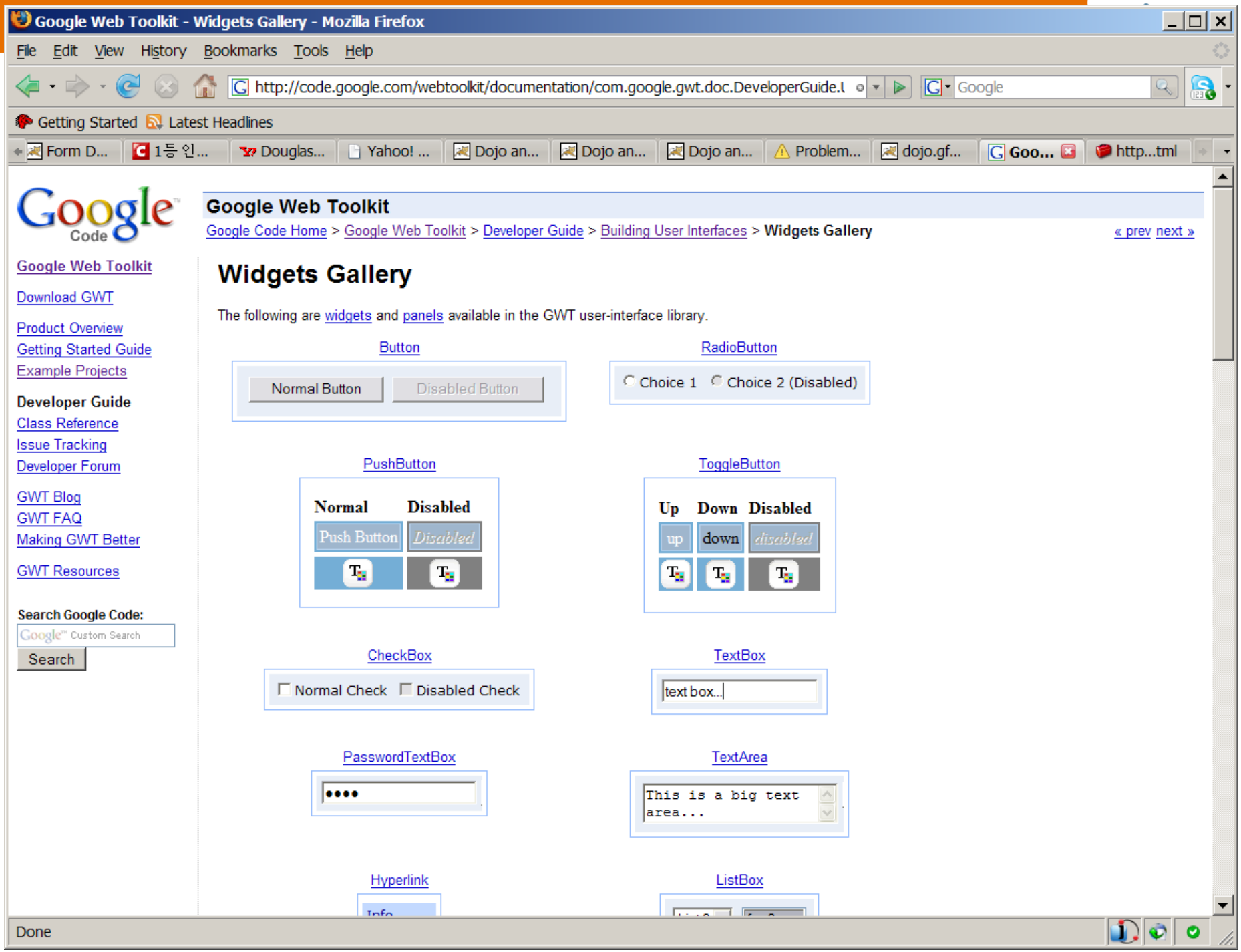
- Optimization
  - All code is optimized, including some unusual tricks (like turning some methods into wrapped statics)
- Dead-Code Elimination
  - Remove any code that cannot be reached, or always returns the same value(s)
- Javascript Generation
  - Generate Javascript, including all necessary browser-specific checks and workarounds
- Javascript obfuscation and size reduction
  - Javascript is (optionally) obfuscated to protect your trade-secrets, and to reduce it's size

Building User Interface:
Built-in GWT Widgets

GWT User Interface Classes

- Similar to those in existing UI frameworks such as Swing except that the widgets are rendered using dynamically-created HTML rather than pixel-oriented graphics
- While it is possible to manipulate the browser's DOM directly using the DOM interface, it is far easier to use Java classes from the Widget hierarchy
- Using widgets makes it much easier to quickly build interfaces that will work correctly on all browsers

Source: http://code.google.com/webtoolkit/


Google Web Toolkit - Widgets Gallery - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://code.google.com/webtoolkit/documentation/com.google.gwt.doc.DeveloperGuide.
Google
Getting Started Latest Headlines
Google Code
Google Web Toolkit
Google Code Home > Google Web Toolkit > Developer Guide > Building User Interfaces > Widgets Gallery
« prev next »
Google Web Toolkit
Download GWT
Product Overview
Getting Started Guide
Example Projects
Developer Guide
Class Reference
Issue Tracking
Developer Forum
GWT Blog
GWT FAQ
Making GWT Better
GWT Resources
Search Google Code:
Google™ Custom Search
Search
Widgets Gallery
The following are widgets and panels available in the GWT user-interface library.
Button
Normal Button
Disabled Button
RadioButton
Choice 1
Choice 2 (Disabled)
PushButton
Normal
Disabled
Push Button
Disabled
ToggleButton
Up
Down
Disabled
up
down
disabled
CheckBox
Normal Check
Disabled Check
TextBox
text box...
PasswordTextBox
TextArea
This is a big text area...
Hyperlink
ListBox
Done

Google Web Toolkit - Widgets Gallery - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://code.google.com/webtoolkit/documentation/com.google.gwt.doc.DeveloperGuide.

Getting Started Latest Headlines

Form D... 1등 인... Douglas... Yahoo! ... Dojo an... Dojo an... Dojo an... Problem... dojo.gf... Goo... http...tml

PasswordTextBox

TextArea

This is a big text area...

Hyperlink

Info
Buttons
Menus
Images
Layouts

ListBox

List 0
List 0
List 1
List 2
List 3
List 4
foo0
bar0
baz0
toto0
tintin0

MenuBar

Style Fruit Term
Bold
Italicized
More »
Code
~~Strikethrough~~
Underlined

Tree

foo@example.com
Inbox
Drafts
Templates

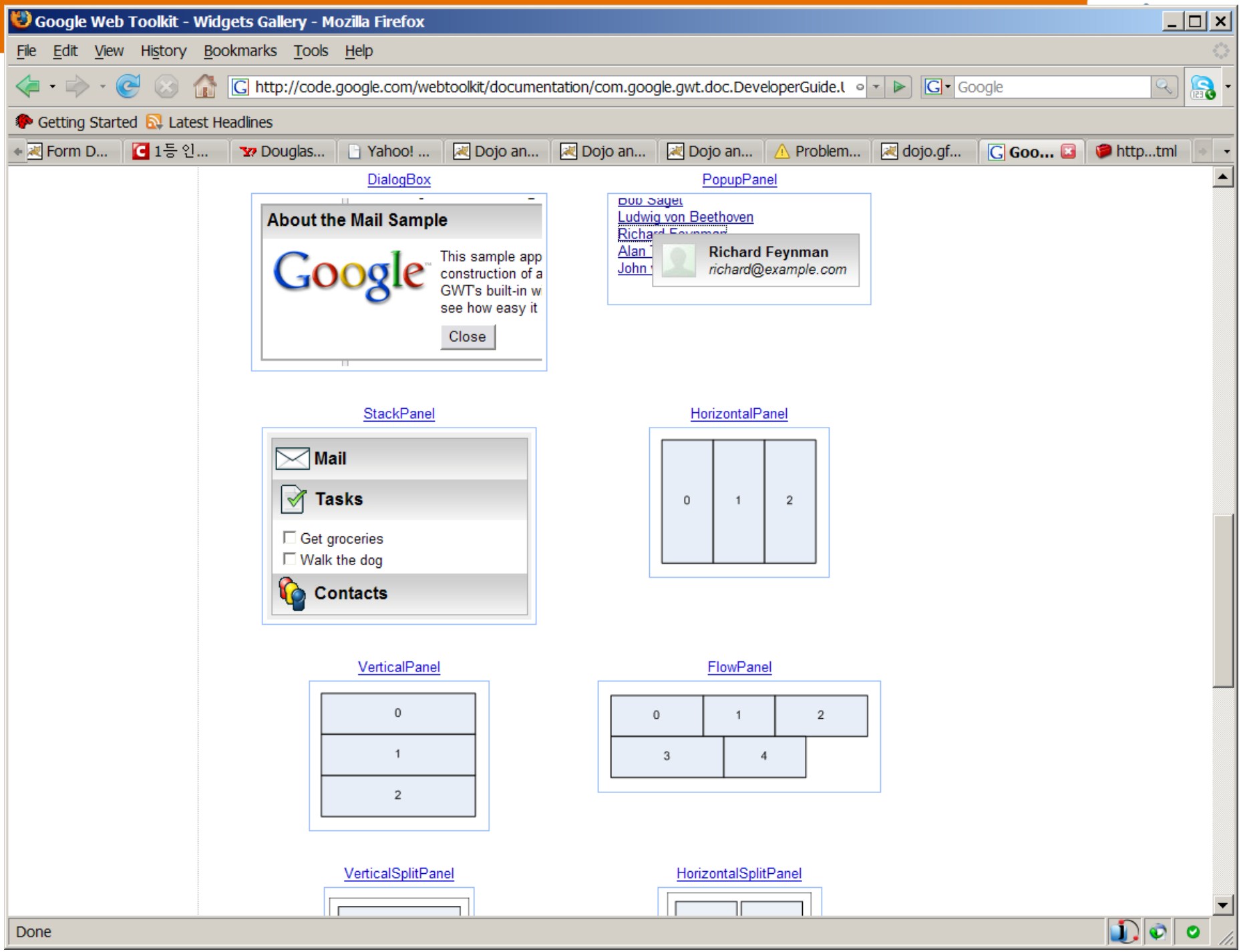
Table

sender	email
markboland05	mark@example.com
Hollie Voss	hollie@example.com
boticario	boticario@example.com
Emerson Milton	emerson@example.com
Healy Colette	healy@example.com
Brigitte Cobb	brigitte@example.com
Elba Lockhart	elba@example.com

TabBar

1634 1640 1642 1662

Done


Google Web Toolkit - Widgets Gallery - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://code.google.com/webtoolkit/documentation/com.google.gwt.doc.DeveloperGuide.
Google
Getting Started
Latest Headlines
DialogBox
About the Mail Sample
This sample app
construction of a
GWT's built-in w
see how easy it
Close
PopupPanel
Ludwig von Beethoven
Richard Feynman
richard@example.com
StackPanel
Mail
Tasks
Get groceries
Walk the dog
Contacts
HorizontalPanel
0
1
2
VerticalPanel
0
1
2
FlowPanel
0
1
2
3
4
VerticalSplitPanel
HorizontalSplitPanel
Done

Google Web Toolkit - Widgets Gallery - Mozilla Firefox

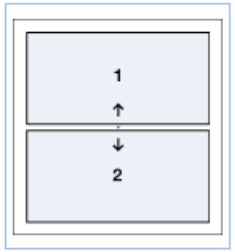
File Edit View History Bookmarks Tools Help

http://code.google.com/webtoolkit/documentation/com.google.gwt.doc.DeveloperGuide.

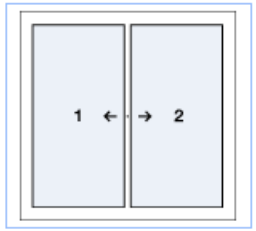
Getting Started Latest Headlines

Form D... 1등 인... Douglas... Yahoo! ... Dojo an... Dojo an... Dojo an... Problem... dojo.gf... Goo... http...tml

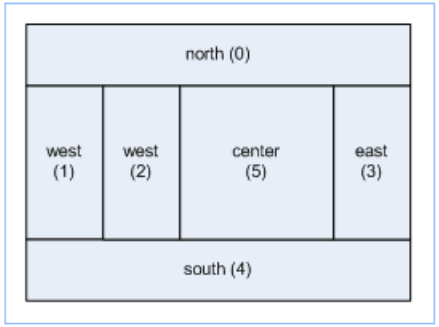
VerticalSplitPanel



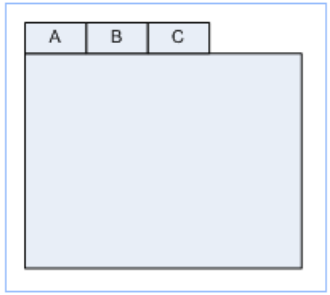
HorizontalSplitPanel



DockPanel



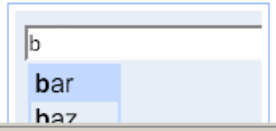
TabPanel



RichTextArea

DisclosurePanel

SuggestBox



Done

Building User Interface: Custom Composite Widget

Custom Composite Widget

- Composite widget is a specialized widget that can contain another component (typically, a panel)
  - You can easily combine groups of existing widgets into a composite widget that is itself a reusable widget
- The most effective way to create new widgets

Source: http://code.google.com/webtoolkit/

Example: Composite Widget

```
public static class OptionalTextBox extends Composite implements
      ClickListener {

  private TextBox textBox = new TextBox();
  private CheckBox checkBox = new CheckBox();

  /**
   * Constructs an OptionalTextBox with the given caption on the check.
   *
   * @param caption the caption to be displayed with the check box
   */
  public OptionalTextBox(String caption) {
     // Place the check above the text box using a vertical panel.
     VerticalPanel panel = new VerticalPanel();
     panel.add(checkBox);
     panel.add(textBox);

     // Set the check box's caption, and check it by default.
     checkBox.setText(caption);
     checkBox.setChecked(true);
     checkBox.addClickListener(this);

     // continued in the next slide
```

Source: http://code.google.com/webtoolkit/

Example: Composite Widget

```
    // All composites must call initWidget() in their constructors.
    initWidget(panel);

    // Give the overall composite a style name.
    setStyleName("example-OptionalCheckBox");
}

public void onClick(Widget sender) {
    if (sender == checkBox) {
        // When the check box is clicked, update the text box's enabled state.
        textBox.setEnabled(checkBox.isChecked());
    }
}
```

Source: http://code.google.com/webtoolkit/

Building User Interface:
Event Handling

Events and Listeners

- Events in GWT use the "listener interface" model similar to other user interface frameworks (like Swing)
  - A listener interface defines one or more methods that the widget calls to announce an event
  - A class wishing to receive events of a particular type implements the associated listener interface - called Event Listener - and then passes a reference to itself to the widget to "subscribe" to a set of events

Source: http://code.google.com/webtoolkit/

Example: Event Listener

```
public class ListenerExample extends Composite implements ClickListener {
  private FlowPanel fp = new FlowPanel();
  private Button b1 = new Button("Button 1");
  private Button b2 = new Button("Button 2");

  public ListenerExample() {
    setWidget(fp);
    fp.add(b1);
    fp.add(b2);
    b1.addClickListener(this);
    b2.addClickListener(this);
  }

  // Event listener method from the ClickListener interface
  public void onClick(Widget sender) {
    if (sender == b1) {
      // handle b1 being clicked
    } else if (sender == b2) {
      // handle b2 being clicked
    }
  }
}
```

Source: http://code.google.com/webtoolkit/

Building User Interface:
Styling through CSS

Steps To Follow

1. Create CSS file (which contains styles)
   - KitchenSink.css
2. Specify the CSS file in the module configuration file
3. Use the styles in the Java code

Source: http://code.google.com/webtoolkit/

Step 1: Create a CSS file Example: KitchenSink.css

```
.ks-List .ks-SinkItem {
  width: 100%;
  padding: 0.3em;
  padding-right: 16px;
  cursor: pointer;
  cursor: hand;
}

.ks-List .ks-SinkItem-selected {
  background-color: #C3D9FF;
}

.ks-images-Image {
  margin: 8px;
}

.ks-images-Button {
  margin: 8px;
  cursor: pointer;
  cursor: hand;
}
```

Source: http://code.google.com/webtoolkit/

Step 2: Specify the CSS file in the Module Configuration File: KitchenSink.gwt.xml

```
<module>
<inherits name='com.google.gwt.user.User'/>
<entry-point
   class='com.google.gwt.sample.kitchensink.client.KitchenSink'/>
<stylesheet src='KitchenSink.css'/>
</module>
```

Source: http://code.google.com/webtoolkit/

Step 3: Add/Remove Styles to Widgets in the Java Code: SinkList.java

```
public void setSinkSelection(String name) {
  if (selectedSink != -1)
    list.getWidget(selectedSink).removeStyleName("ks-SinkItem-selected");

  for (int i = 0; i < sinks.size(); ++i) {
    SinkInfo info = (SinkInfo) sinks.get(i);
    if (info.getName().equals(name)) {
      selectedSink = i;
      list.getWidget(selectedSink).addStyleName("ks-SinkItem-selected");
      return;
    }
  }
}
```

Source: http://code.google.com/webtoolkit/

Adding/Removing Styles

- Style can be added to or removed from widgets via
  - <Widget>.addStyleName(“<name-of-style>”);
  - <Widget>.removeStyleName(“<name-of-style>”);
- Multiple styles can be added to a widget

Source: http://code.google.com/webtoolkit/

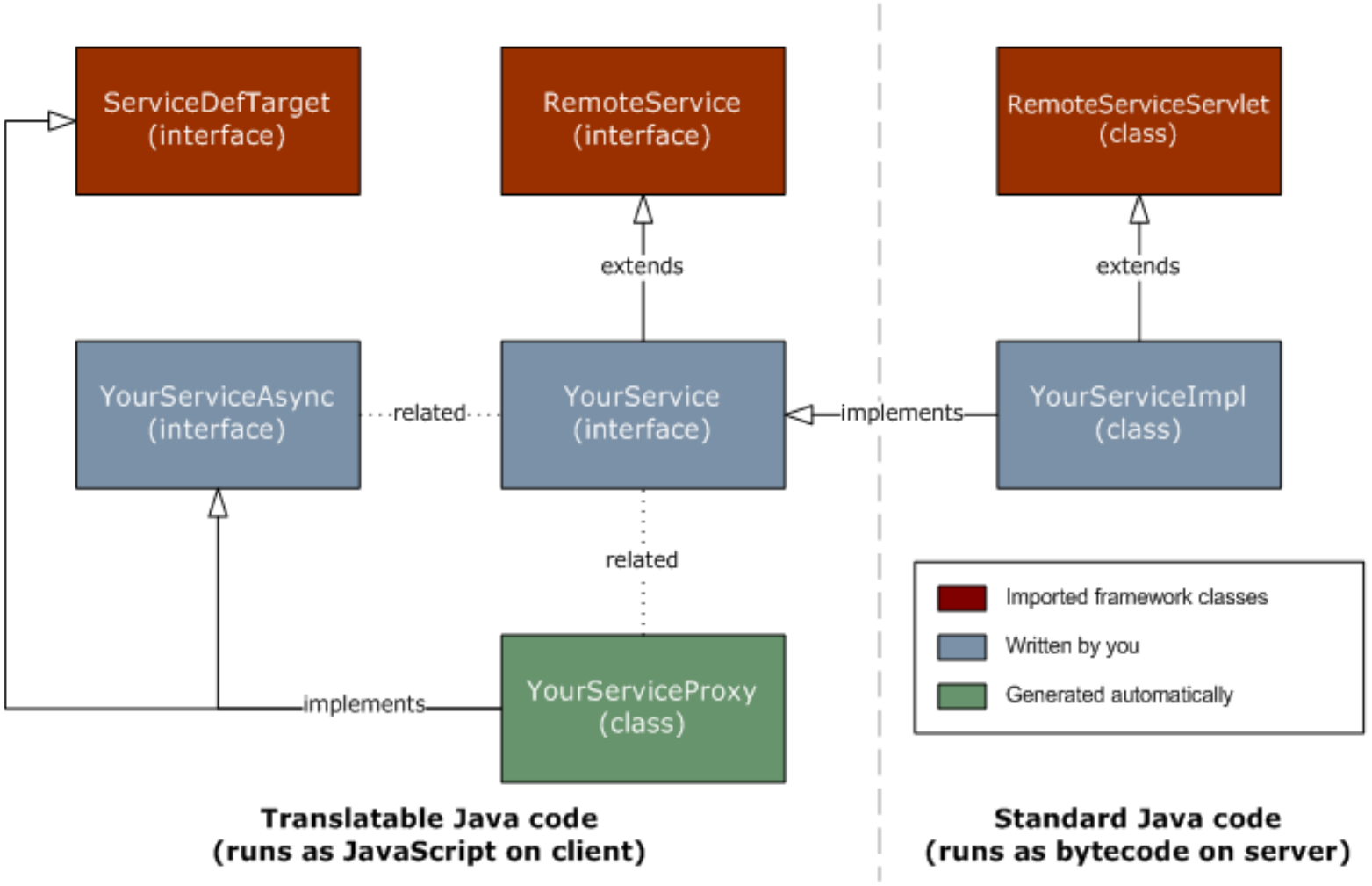
Remore Procedure Call (RPC)

What is and Why GWT RPC?

- Mechanism for interacting with the server by invoking a method
  - Example: Used for fetching data from the server
- Makes it easy for the client and server to pass Java objects back and forth over HTTP
  - Marshaling and unmarshaling of Java objects are handled by the GWT
- When used properly, RPCs give you the opportunity to move all of your UI logic to the client (leaving business logic on the server)
  - Could result in greatly improved performance, reduced bandwidth, reduced web server load, and a pleasantly fluid user experience

Source: http://code.google.com/webtoolkit/

GWT RPC Plumbing Architecture



Source: http://code.google.com/webtoolkit/

Remore Procedure Call (RPC)

Sub-topic: Steps for Implementing GWT RPC

Steps for Implementing GWT RPC

1. Write two service interface's (client & server)
   - Synchronous interface
   - Asynchronous interface - has to pass async. callback object
2. Implement the service at the server side
   - Service class implements Service interface and extends RemoteServiceServlet class
3. Configure the service in the module configuration file
   - Needed for running the app in hosted mode
4. Make a call from the client

Source: http://code.google.com/webtoolkit/

1. Write Two Service Interface Files

- Synchronous interface

```
public interface MyHelloService extends RemoteService {
  public String sayHello(String s);
}
```

- Asynchronous interface

```
// Has to be named as <Synchronous-interface>Async.
// Has to pass AsyncCallback object as the last parameter.
// The return type is always void.

interface MyHelloServiceAsync {
  public void sayHello(String s, AsyncCallback callback);
}
```

Source: http://code.google.com/webtoolkit/

2. Implement the Service

- Extends RemoteServiceServlet and implements the service interface

```
public class MyHelloServiceImpl extends RemoteServiceServlet
                                 implements MyHelloService {

    // Provide implementation logic.
    public String sayHello(String s) {
        return "Hello, " + s + "!";
    }

}
```

Source: http://code.google.com/webtoolkit/

3. Configure the Service in the Module Configuration File

```
<module>
   <inherits name="com.google.gwt.user.User"/>
   <entry-point class="com.google.gwt.sample.hello.client.Hello"/>
   <servlet path='/hellorpc'
    class='com.google.gwt.sample.hello.server.MyHelloServiceImpl'/>
</module>
```

Source: http://code.google.com/webtoolkit/

4. Make a call from Client

a) Instantiate an client proxy (an object of the type of asynch. service interface) using GWT.create()

b) Specify a service entry point URL for the service proxy using ServiceDefTarget

c) Create an asynchronous callback object to be notified when the RPC has completed

d) Make the call from the client

Source: http://code.google.com/webtoolkit/

a. Instantiate Service Interface using GWT.create()

```
public void menuCommandEmptyInbox() {

  // (a) Create the client proxy. Note that although you are creating the
  // object instance of the service interface type, you cast the result to the asynchronous
  // version of the interface. The cast is always safe because the generated proxy
  // implements the asynchronous interface automatically.
  //
  MyEmailServiceAsync emailService =
                        (MyEmailServiceAsync) GWT.create(MyEmailService.class);
```

Source: http://code.google.com/webtoolkit/

b. Specify a service entry point URL for the service proxy using ServiceDefTarget

```
public void menuCommandEmptyInbox() {

  // (a) Create the client proxy. Note that although you are creating the
  // service interface proper, you cast the result to the asynchronous
  // version of the interface. The cast is always safe because the generated proxy
  // implements the asynchronous interface automatically.
  //
  MyEmailServiceAsync emailService =
                    (MyEmailServiceAsync) GWT.create(MyEmailService.class);

  // (b) Specify the URL at which our service implementation is running.
  // Note that the target URL must reside on the same domain and port from
  // which the host page was served.
  //
  ServiceDefTarget endpoint = (ServiceDefTarget) emailService;
  String moduleRelativeURL = GWT.getModuleBaseURL() + "email";
  endpoint.setServiceEntryPoint(moduleRelativeURL);
```

Source: http://code.google.com/webtoolkit/

c. Create an asynchronous callback object to be notified when the RPC has completed

```
public void menuCommandEmptyInbox() {

  ...

 // (c) Create an asynchronous callback to handle the result.
 //
 AsyncCallback callback = new AsyncCallback() {
   public void onSuccess(Object result) {
     // do some UI stuff to show success
   }

   public void onFailure(Throwable caught) {
     // do some UI stuff to show failure
   }
 };
```

Source: http://code.google.com/webtoolkit/

d. Make the Call

```
public void menuCommandEmptyInbox() {

  ...

  // (d) Make the call. Control flow will continue immediately and later
  // 'callback' will be invoked when the RPC completes.
  //
  emailService.emptyMyInbox(fUsername, fPassword, callback);
}
```

Source: http://code.google.com/webtoolkit/

Remore Procedure Call (RPC)

Sub-topic: Serializable Types

Serializable Types

- Method parameters and return types must be serializable
- Java data types that are already serializable
  - primitives, such as char, byte, short, int, long, boolean, float, or double
  - String, Date, or wrapper types such as Character, Byte, Short, Integer, Long, Boolean, Float, or Double
  - An array of serializable types (including other serializable arrays)
  - A serializable user-defined class
  - A class has at least one serializable subclass

Source: http://code.google.com/webtoolkit/

Serializable User-defined Classes

- A user-defined class is serializable if
  - it is assignable to IsSerializable, either because it directly implements the interface or because it derives from a superclass that does, and
  - all non-transient fields are themselves serializable.

Source: http://code.google.com/webtoolkit/

Remore Procedure Call (RPC)

Sub-topic: Handling Exceptions

Handling Exceptions

- Making RPCs opens up the possibility of a variety of errors
  - Networks fail, servers crash, and problems occur while processing a server call
- GWT lets you handle these conditions in terms of Java exceptions
- RPC-related exceptions
  - Checked exceptions
  - Unexpected exceptions

Source: http://code.google.com/webtoolkit/

Checked Exceptions

- Service interface methods support throws declarations to indicate which exceptions may be thrown back to the client from a service implementation
- Callers should implement AsyncCallback.onFailure(Throwable) to check for any exceptions specified in the service interface

Source: http://code.google.com/webtoolkit/

Unchecked Exceptions

- An RPC may not reach the service implementation at all. This can happen for many reasons:
  - the network may be disconnected
  - a DNS server might not be available
  - the HTTP server might not be listening
- In this case, an InvocationException is passed to your implementation of AsyncCallback.onFailure(Throwable)

Source: http://code.google.com/webtoolkit/

JavaScript Native Interface (JSNI)

Why JSNI?

- Sometimes it's very useful to mix handwritten JavaScript into your Java source code
  - For example, the lowest-level functionality of certain core GWT
- Leverage various existing JavaScript toolkits
  - Dojo toolkits, Prototype, Rico, etc.
- Should be used sparingly
  - JSNI code is less portable across browsers, more likely to leak memory, less amenable to Java tools, and hard for the compiler to optimize
- Web equivalent of inline assembly code

Source: http://code.google.com/webtoolkit/

What Can You Do with JSNI?

- Implement a Java method directly in JavaScript
- Wrap type-safe Java method signatures around existing JavaScript
- Call from JavaScript into Java code and vice-versa
- Throw exceptions across Java/JavaScript boundaries
- Read and write Java fields from JavaScript
- Use hosted mode to debug both Java source (with a Java debugger) and JavaScript (with a script debugger, only in Windows right now)

Source: http://code.google.com/webtoolkit/

JavaScript Native Interface (JSNI): Accessing Native JavaScript Methods from Java Code

How to add JavaScript Code via JSNI?

- When accessing the browser's window and document objects from JSNI, you must reference them as $wnd and $doc, respectively
  - Your compiled script runs in a nested frame, and $wnd and $doc are automatically initialized to correctly refer to the host page's window and document

Source: http://code.google.com/webtoolkit/

Writing Native JavaScript Methods

- JSNI methods are declared native and contain JavaScript code in a specially formatted comment block between the end of the parameter list and the trailing semicolon
  - /*-{ <JavaScript code }-*/
- JSNI methods are be called just like any normal Java method
- They can be static or instance methods

Source: http://code.google.com/webtoolkit/

Example: Native JavaScript Methods

```
public static native void alert(String msg) /*-{
  $wnd.alert(msg);
}-*/;
```

Source: http://code.google.com/webtoolkit/

JavaScript Native Interface (JSNI): Accessing Java Methods & Fields from JavaScript

Invoking Java methods from JavaScript

- Calling Java methods from JavaScript is somewhat similar to calling Java methods from C code in JNI. In particular, JSNI borrows the JNI mangled method signature approach to distinguish among overloaded methods.
- [instance-expr.]@class-name::method-name(param-signature)(arguments)

Source: http://code.google.com/webtoolkit/

Accessing Java Fields from JavaScript

- Static and instance fields can be accessed from handwritten JavaScript
- [instance-expr.]@class-name::field-name

Source: http://code.google.com/webtoolkit/

GWT Module Configuration

Configuration Settings of a GWT Project (defined in *.gwt.xml file)

- Inherited modules
- An entry point class name; these are optional, although any module referred to in HTML must have at least one entry-point class specified
- Source path entries
- Public path entries
- Deferred binding rules, including property providers and class generators

Source: http://code.google.com/webtoolkit/

Module XML (*.gwt.xml) File Format

- Modules are defined in XML files whose file extension is .gwt.xml
- Module XML files should reside in the project's root package

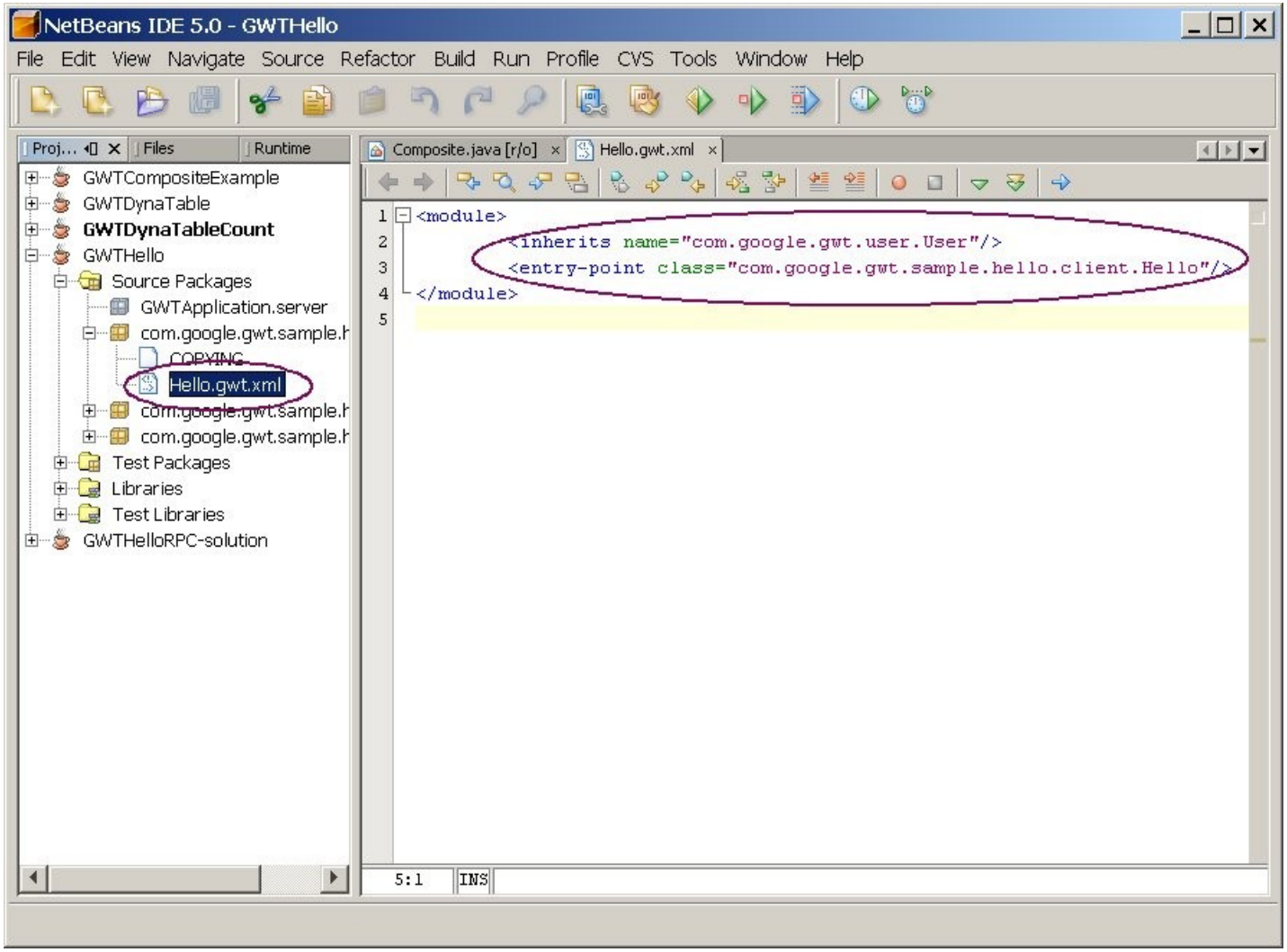
Source: http://code.google.com/webtoolkit/

Entry-Point Classes

- 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

Source: http://code.google.com/webtoolkit/

Example: Entry-Point class of GWTHello



Source: http://code.google.com/webtoolkit/

Source Path

- Modules can specify which subpackages contain translatable source, causing the named package and its subpackages to be added to the source path
  - Only files found on the source path are candidates to be translated into JavaScript, making it possible to mix client-side and server-side code together in the same classpath without conflict
- When module inherit other modules, their source paths are combined so that each module will have access to the translatable source it requires
- If no <source> element is defined in a module XML file, the client subpackage is implicitly added to the source path as if <source path="client"> had been found in the XML

Source: http://code.google.com/webtoolkit/

Public Path

- Modules can specify which subpackages are public, causing the named package and its subpackages to be added to the public path
- 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 net effect is that user-visible URLs need not include a full package name.
- When module inherit other modules, their public paths are combined so that each module will have access to the static resources it expects.

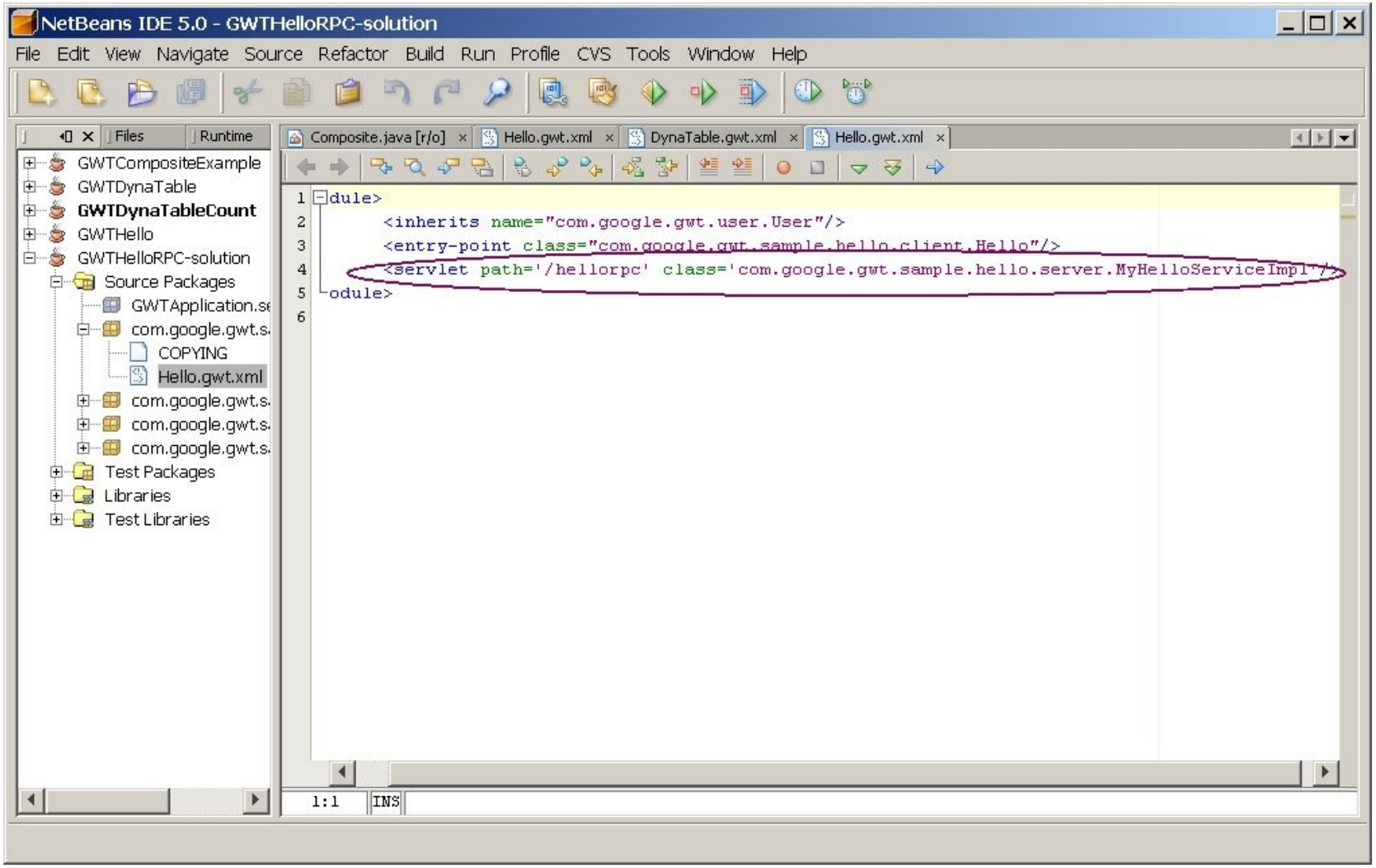
Source: http://code.google.com/webtoolkit/

Servlet Path

- For convenient RPC testing, this element loads a servlet class mounted at the specified URL path
- The URL path should be absolute and have the form of a directory (for example, /spellcheck)
  - Your client code then specifies this URL mapping in a call to ServiceDefTarget.setServiceEntryPoint(String)
- Any number of servlets may be loaded in this manner, including those from inherited modules.

Source: http://code.google.com/webtoolkit/

Example: Servlet Path



Source: http://code.google.com/webtoolkit/

GWT Project Structure

Standard GWT Project Layout

- GWT projects are overlaid onto Java packages such that most of the configuration can be inferred from the classpath and your module definitions (*.gwt.xml files)
- Standard GWT Project layout
  - com/example/cal/ - The project root package contains module XML files
  - com/example/cal/client/ - Client-side source files and subpackages
  - com/example/cal/server/ - Server-side code and subpackages
  - com/example/cal/public/ - Static resources that can be served publicly

Source: http://code.google.com/webtoolkit/

Example: Calendar GWT Application (Root directory)

- com/example/cal/Calendar.gwt.xml
  - A common base module for your project that inherits com.google.gwt.user.User module
- com/example/cal/CalendarApp.gwt.xml
  - Inherits the com.example.cal.Calendar module (above) and adds an entry point class
- com/example/cal/CalendarTest.gwt.xml
  - A module defined by your project

Source: http://code.google.com/webtoolkit/

Example: Calendar GWT Application (client and server directories)

- com/example/cal/client/CalendarApp.java
  - Client-side Java source for the entry-point class
- com/example/cal/client/spelling/SpellingService.java
  - An RPC service interface defined in a subpackage
- com/example/cal/server/spelling/SpellingServiceImpl.java
  - Server-side Java source that implements the logic of the spelling service

Source: http://code.google.com/webtoolkit/

Example: Calendar GWT Application (public directory)

- com/example/cal/public/Calendar.html
  - An HTML page that loads the calendar app
- com/example/cal/public/Calendar.css
  - A stylesheet that styles the calendar app
- com/example/cal/public/images/logo.gif
  - A logo

Source: http://code.google.com/webtoolkit/

Resources

Resources

- http://code.google.com/webtoolkit/
  - Main site for all things GWT
- http://googlewebtoolkit.blogspot.com/
  - GWT team blog
- http://code.google.com/p/google-web-toolkit/
  - GWT source code and issue tracker site
- Slides from Presentation at TSSJS (PDF)
  - http://tinyurl.com/3y8kmx
- Video from Google Developer Day
  - http://tinyurl.com/2rkq29

Google Web Toolkit (GWT) Basics

Sang Shin
Java Technology Architect
Sun Microsystems, Inc.
sang.shin@sun.com
www.javapassion.com