

Writing **Big** Applications With Google Web Toolkit



Bruce Johnson
Google, Inc.
`bruce@google.com`



A Simpler-Than-Possible Explanation of GWT

Why AJAX Matters

GWT is Software Engineering for AJAX

Fake Q & A: A Soliloquy

Big Applications

Summary

Real Q & A

What is Google Web Toolkit (GWT)?



What is GWT?

A set of tools for building AJAX apps in the Java language

What makes GWT interesting?

Write, run, test, and debug everything in Java

Isn't that called an applet?

GWT converts your working Java source into equivalent JavaScript

GWT is a compiler?

GWT has a compiler, but the full story is even more interesting...

A Simpler-Than-Possible Explanation of GWT

Why AJAX Matters

GWT is Software Engineering for AJAX

Fake Q & A: A Soliloquy

Big Applications

Summary

Real Q & A

Updating the browser UI without switching pages

Traditionally called Dynamic HTML (DHTML)

Relies on JavaScript running to direct the UI updates

Fetching data without switching pages

Using XMLHttpRequest (XHR) to fetch data "in the background"

Viewing browsers as smart clients

Instead of HTML dumb terminals

Sharing the computational burden

Better server utilization

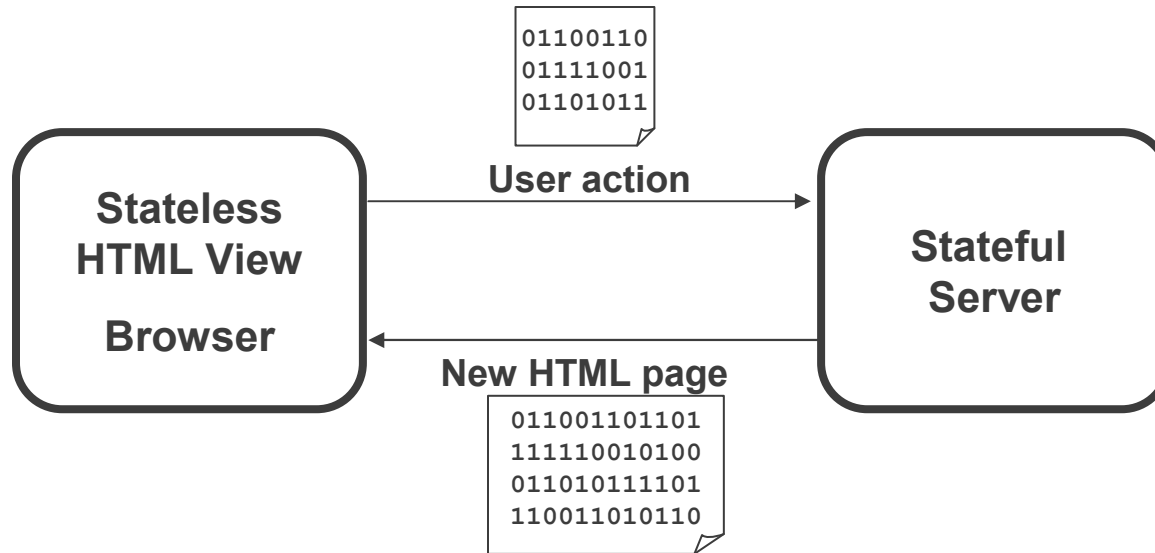
Applications that are more responsive than classic HTML

In other words, AJAX is recreating 1990s-style client/server computing without the need to install software locally

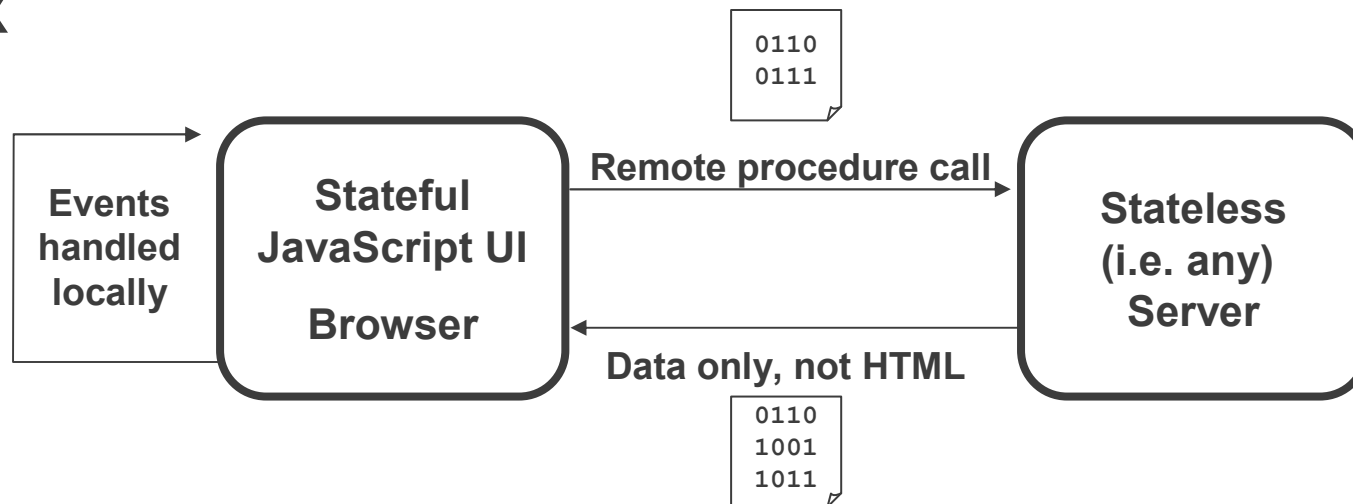
Why AJAX? Infrastructure Benefits



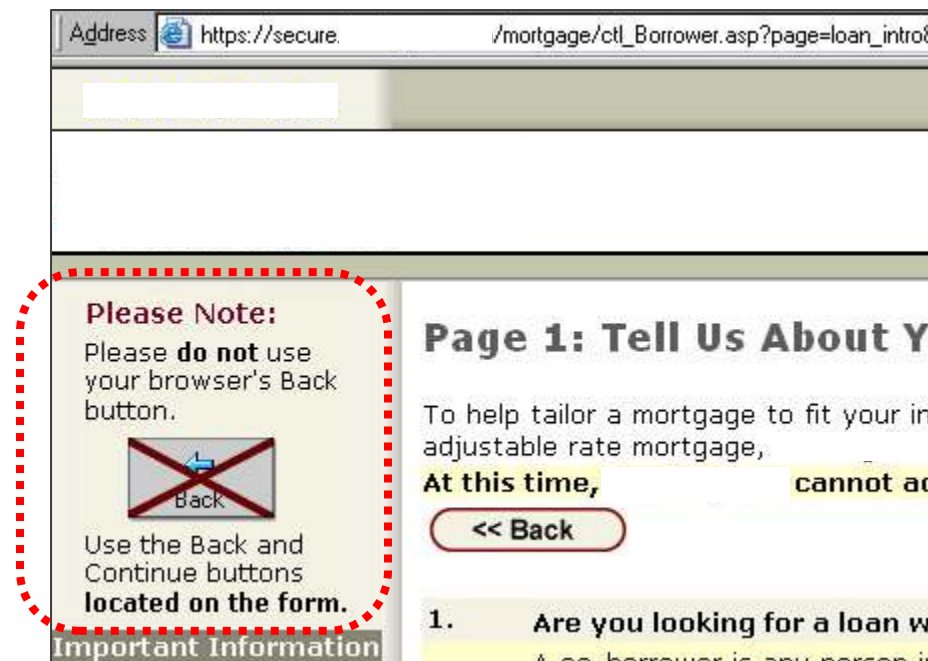
Traditional HTML



AJAX



"Do not use your browser's Back button"



What if I do click Back?
AJAX can (in theory) solve this

"Don't hit reload or we'll charge you twice!"



**What if the network hangs? What should I do?
AJAX can (in theory) solve this**

**AJAX is more than a fad
because it genuinely benefits
all* stakeholders**

**AJAX will be important
for many years to come**

***not counting developers**

A Simpler-Than-Possible Explanation of GWT

Why AJAX Matters

GWT is Software Engineering for AJAX

Fake Q & A: A Soliloquy

Big Applications

Summary

Real Q & A

Poor Usability

No history

No bookmarks

Frozen browser chrome and pegged user CPUs

Worst: Easy development and good usability are conflicting goals

Poor Browser Portability

Hard to test: every line of code is a potential portability bug

Either wrap every single browser-related call (heavy)...

Or be paranoid about every line of code (risky)

Poor Speed

Startup time is an extremely huge sacrifice...probably not worth it

Large scripts run more slowly

Worst: Maintainability and efficiency are conflicting scripting goals

typos + expandos = bug-o-s

Imagine this gem on line 5912 of your script

```
x.compnent = document.getElementById("x");  
// a spelling(!) bug that will bite much later
```

Yes, there's a reason static type checking was invented

Java didn't have this problem

And reuse is a good way to not write bugs

And don't forget code completion

And this starts to matter a lot for big projects

Poor Tool Support

Limited IDE support

Debugging = `window.alert()`

Profilers? Code coverage? Findbugs? ...

Quality Risks

New categories of runtime-only bugs

Poor JS reuse model encourages "from scratch" or copy/paste

Browsers are a moving target

Long-Term Risks

Hard to schedule (e.g. unexpected browser quirks)

Spaghetti risk

Poor documentation

Hard for large teams to work on the same code base

Hard enough to find one AJAX guru

**It is very easy to slip into making a
poorly planned AJAX investment**

**...but you'll live
with the consequences
for a long, long time**

Make great AJAX apps that are still very webby

History, Bookmarks, a working Back button...

Leverage the Java language, developers, and technologies

IDEs, debugging, JUnit, findbugs, and profiling

Eliminate browser-specific coding with low overhead

Facilitate reuse at the Java language level via jars

Fast, simple remote procedure calls with rich semantics

Scalability...server-side session state is a big drain

Basically: the impossible...

Unless you translate Java into JavaScript :-)

Code Sample – Hello, AJAX



Demo time...

```
public class Hello implements EntryPoint {  
  
    public void onModuleLoad() {  
        Button b = new Button("Click me", new ClickListener() {  
            public void onClick(Widget sender) {  
                Window.alert("Hello, AJAX");  
            }  
        });  
  
        RootPanel.get().add(b);  
    }  
}
```

Demo

Hello, AJAX



Redefining the problem has been fruitful

Session state? All client, not a server issue

Avoids round trips for UI event handling

Deployment? No fancy server, just compiled JS

Leverage for the biggest AJAX headaches

Our Mantra: Solve the problem once & wrap it in a class

History? Create a History class

Cross-browser? Create an abstract DOM class

RPC? Create an all-Java RPC mechanism

Build (or reuse!) widgets

Written in straight Java

Code without worrying about browser portability

Separate UI style from logic

Widgets are styled with CSS

Automatically load the right CSS for your widgets

Demo

"Mail" is a desktop-style application

Demo: User Admin Dialog Box

GWT saves you round trips

Very fast startup time

Separation of concerns in the code

Keyboard support

On-the-fly font resizing

Reduce server load and improve usability

History is the first thing to go in most AJAX apps

With GWT, it's easy and works well with MVC

```
History.addHistoryListener(myController);
```

History support leads to bookmark support

http://google.com/gulp.html#beta_carroty

Demo

"KitchenSink" shows history, bookmarking, and widgets

Many solutions out there (JSON, XML-RPC, ...)

But a pure Java RPC interface sure is nice!

```
interface SpellService extends RemoteService {  
    /**  
     * Checks spelling and suggests alternatives.  
     * @param the word to check  
     * @return the list of alternatives, if any  
     */  
    String[] suggest(String word)  
}
```

Client and server speak the same language (Java)

Demo

"DynaTable" loads records dynamically

JUnit integration

Direct IDE support

Run tests in hosted mode

Easy to debug and analyze

Run tests in web mode

In-the-wild verification

Works with a farm of remote browsers

Demo

Testing the ListBox and Tree widgets

A Simpler-Than-Possible Explanation of GWT

Why AJAX Matters

GWT is Software Engineering for AJAX

Fake Q & A: A Soliloquy

Big Applications

Summary

Real Q & A

Which browsers are supported?

Firefox 1.0, 1.5, 2.0

Internet Explorer 6, 7

Safari 2.0

Opera 8.5, 9.0

**What happens when a new browser comes out?
Do I have to wait for the GWT compiler to be updated?**

Definitely no!

All browser-specific code is in user-level libraries

The JavaScript language itself has very consistent support across browsers

The DOM API is the real culprit

For backwards-compatible browsers, it's a no-brainer

For other situations, it's straightforward to change the user-level libraries

Implement a version of DOMImpl for the desired browser

Main point: GWT was designed to never be a roadblock

Isn't it really hard to debug the JavaScript that the GWT compiler produces?

If you need to (or just want to) debug the compiled output, the GWT compiler gives you multiple output options:

- style OBFUSCATED (small, efficient, and fast)
- style DETAILED (nothing is left to the imagination)
- style PRETTY (perfect if you want to actually follow the code)

The output is normal JS, so you can always use any JavaScript debugger as you would with handwritten code.

By the way, you will likely never have to do any of this. You'll be doing your debugging in Java.

What functionality is included with GWT?

User Interface

Client/Server Communication

Application Infrastructure

Internationalization

...

GWT Library Overview



AbsolutePanel, Button, ButtonBase, CellPanel, ChangeListenerCollection, CheckBox, ClickListenerCollection, ComplexPanel, Composite, DeckPanel, DialogBox, DockPanel, FileUpload, FlexTable, FlowPanel, FocusListenerAdapter, FocusListenerCollection, FocusPanel, FocusWidget, FormHandlerCollection, FormPanel, FormSubmitCompleteEvent, FormSubmitEvent, Frame, Grid, HorizontalPanel, HTML, HTMLPanel, HTMLTable, Hyperlink, Image, KeyboardListenerAdapter, KeyboardListenerCollection, Label, ListBox, LoadListenerCollection, MenuBar, MenuItem, MouseListenerAdapter, MouseListenerCollection, NamedFrame, Panel, PasswordTextBox, PopupListenerCollection, PopupPanel, RadioButton, RootPanel, ScrollListenerCollection, ScrollPanel, SimplePanel, StackPanel, TabBar, TableListenerCollection, TabListenerCollection, TabPanel, TextArea, TextBox, TextBoxBase, Tree, TreeItem, TreeListenerCollection, UIObject, VerticalPanel, Widget, WidgetCollection

User Interface

History, DeferredCommand, Localizable, Constants, Dictionary, ConstantsWithLookup, Messages

Usability and I18N

DOMException, XMLParser, Attr, CDATASection, CharacterData, Comment, Document, DocumentFragment, Element, EntityReference, NamedNodeMap, Node, NodeList, ProcessingInstruction, Text

XML

AsyncCallback, IsSerializable, RemoteService, RemoteServiceServlet

RPC

JSONArray, JSONBoolean, JSONException, JSONNull, JSONNumber, JSONObject, JSONParser, JSONString, JSONValue

JSON

Header, Request, RequestBuilder, RequestCallback, RequestException, Response, URL

HTTP

How big are GWT apps? Doesn't the compiler produce bloated script?

For tiny bits of functionality (say, < 100 lines) of handwritten JS, you might be better off writing it by hand. Beyond that, compiler size and speed optimizations will ultimately win.

Examples of compiler optimizations

- Dead code removal
- Type tightening
- Polymorphism removal
- Inlining
- JS code gen compression

(See next slide for experimental data)

Leverage: Wicked Cool Optimizations



Tough decision not to support reflection and class loading

Worth it! Three words: Whole program optimization

For example, type tightening to eliminate polymorphism

```
Shape s = new Circle(2); // radius of 2
double a = s.getArea();
```

can become

```
Circle s = new Circle(2); // radius of 2
double a = (s.radius * s.radius * Math.PI);
```

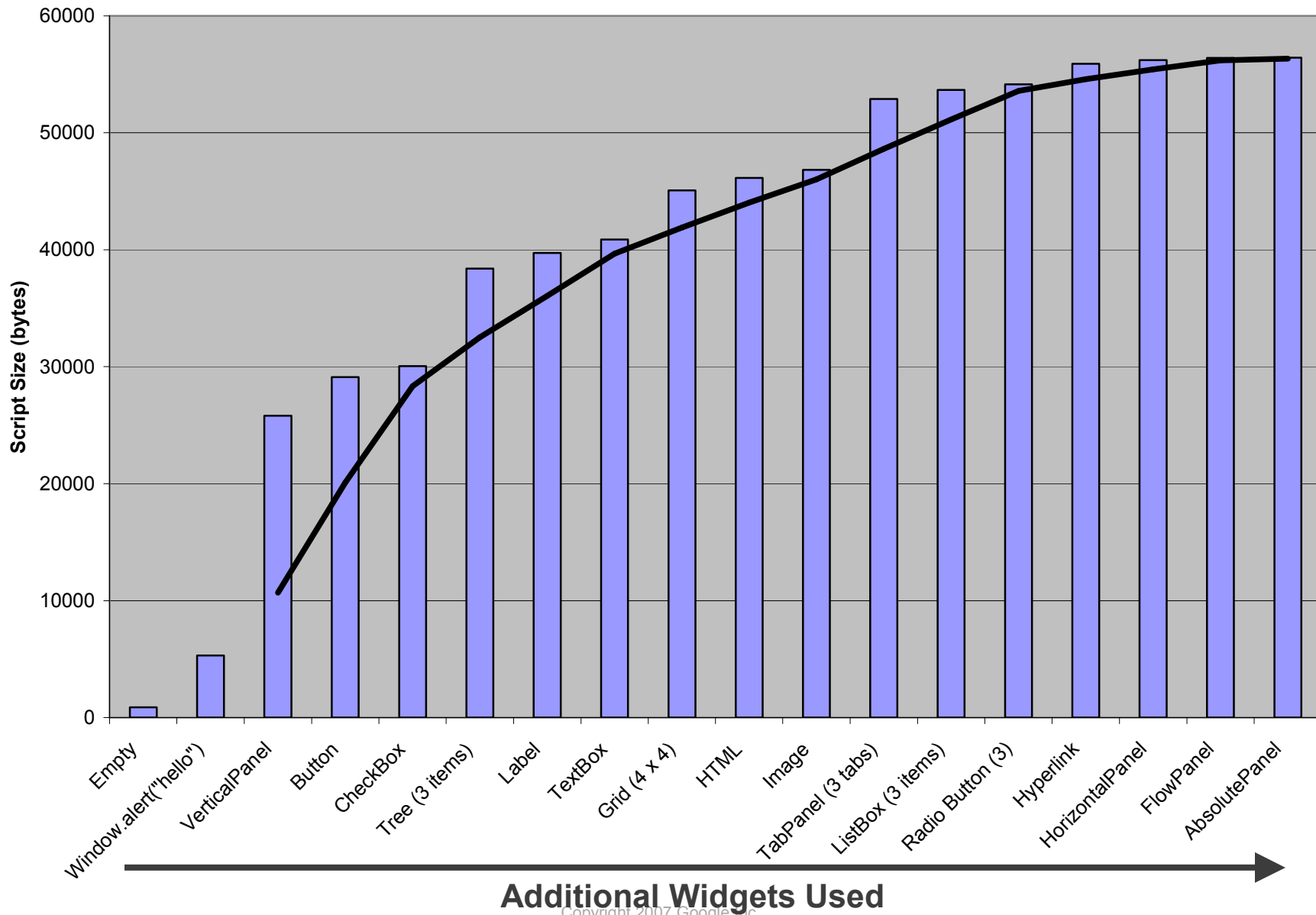
which, if Circle has no side effects, can become

```
double a = 12.5663706143591;
```

Imagine those sorts of optimizations across your entire app

In JavaScript, reducing size and increasing speed are complementary goals, which makes optimizations *really* fun

Compilation: Only Pay for What You Use



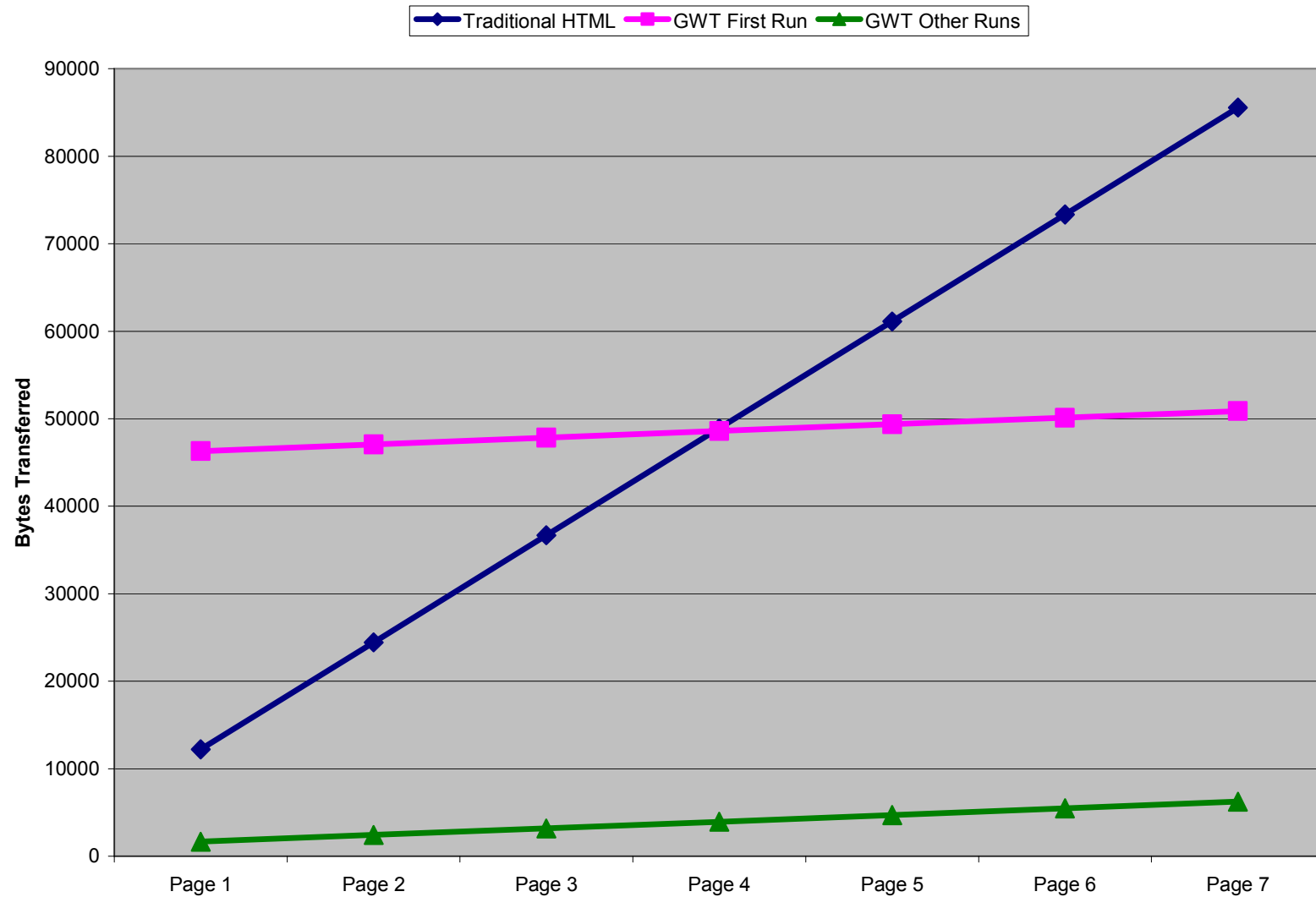
How fast are GWT apps? Surely I could write faster apps by hand!

Likely to be true for very small apps

Unlikely to be true for bigger apps due to compiler and class library optimizations

(See next slide for experimental data)

Efficiency: Bandwidth and Startup Time



**Does GWT have to control the entire page?
I can't rewrite my app from scratch!**

GWT does not force you to start over!

Attach code to existing pages with a <meta> tag

```
<html>
  ..<meta name="gwt:module" content="..." />
  ..<h1>Welcome to GWTravel Services</h1>
  ..<div id="reservationWizard">
  ..</html>
```

Your Java source is as loosely-coupled as you need it to be

```
Panel p = RootPanel.get("reservationWizard");
Wizard wiz = new ReservationWizard();
p.add(wiz);
```

Works with any HTML-generating server approach

A Simpler-Than-Possible Explanation of GWT

Why AJAX Matters

GWT is Software Engineering for AJAX

Fake Q & A: A Soliloquy

Big Applications

Summary

Real Q & A

The obvious question that rarely gets asked

What exactly are we trying to optimize for?

Download speed?

Are we supporting dial-up users?

Startup time?

First run? Subsequent runs? How fast, exactly?

Some particular size cutoff?

Size-on-wire? Size-in-cache?

Is the cutoff arbitrary or based on measured effects?

Funny: compare script size to the size of your images

Ahead-of-time script compression

C6BD1564339FC70220.cache.html (119 KB)

C6BD1564339FC70220.cache.html.gz (39 KB)

Our "big" app instantly became 3 times smaller

The last step of your build should be to zip GWT output

Classic HTML can't use compression so well

Data changes frequently

HTML changes rarely

Mixing them forces compression into the critical path

GWT supports aggressive script caching

Combine a small "selection script"...

→ `KitchenSink.nocache.html`

`Expires: <pretty soon>`

With a larger compiled script...

→ `md5.cache.html`

`Expires: <when the sun explodes>`

Viola! Perfect caching!

Never re-fetch the big script *unless* it has changed

Never *fail* to re-fetch the big script when it *has* changed

If you're confident that it's going to be a big app...

The default choice should be client-side MVC

Only tricky part is making your model async

Then again, not so bad...

```
myModel.requestNthItem(14) ;  
...  
class MyView implements MyModelListener {  
    void onNthItemReceived(int n, Item item) {  
...
```

MVC also fits perfectly with GWT history

- 1. Start by assuming you have a single page and you're building a traditional client-side MVC app (remember client/server? :-)**
- 2. Add code as if you'll never hit a brick wall**
- 3. Make sure your app implements history well**
- 4. Evaluate the size and speed of your app**
 - A. If you're happy, goto 2
 - B. If you're unhappy, do all the stuff on the previous slides
 - C. If you're still unhappy, see the next slide

Never feel obligated to keep your GWT app to a single page

History smoothes this over

Startup is fast to make page switching inexpensive

When wrangled by GWT, IFRAMES aren't so evil

Divide big chunks into IFRAMES that your controller shows/hides them as necessary



Consolidate multiple small RPCs

Build composite structures and large-grained APIs

Good rule of thumb: minimize HTTP round-trips

Server replies with more data than was requested

Create UI lazily

Fits naturally with history and MVC

Spread the cost of widget creation across user-time

See KitchenSink for an example

A Simpler-Than-Possible Explanation of GWT

Why AJAX Matters

GWT is Software Engineering for AJAX

Fake Q & A: A Soliloquy

Big Applications

Summary

Actual Q & A

Internationalization support

Highly optimized

Externalized string ids are checked during compilation

Automatic, dynamic dependency inclusion

Slurp in external CSS

Slurp in external JS

Everything is cross-browser

IE6+, FF 1.0.x, FF 1.5.x, Safari 2.0.x, Opera 9.x

Your choice of development platforms

Mac OS X, Linux, Windows

Your choice of IDEs

IntelliJ IDEA, Eclipse, NetBeans, JCreator, JBuilder

GWT is Open Source



Licensed under Apache 2.0

Source is available via svn on Google Code project hosting

Our charter document is "Making GWT Better"

Mission statement

Design axioms

Community forums

How to build GWT from source

Code style

Submitting patches

Transparent development (published minutes, roadmap)

Great participation

100,000+ downloads of the release candidates for GWT 1.3

Great discussion on G-W-T and G-W-T-C lists

200+ developers on the contributors list

Patches are rolling in!

Documentation Included



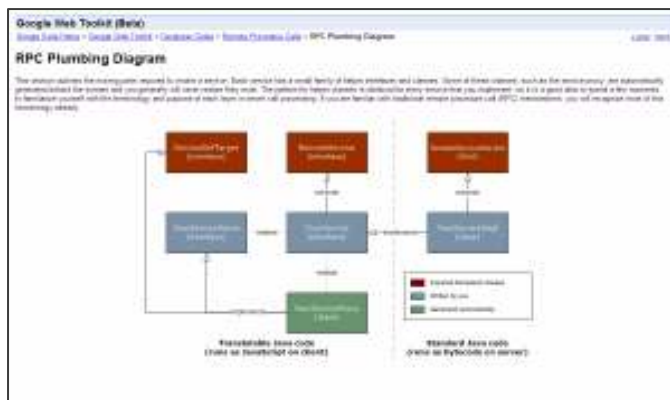
Getting Started Guide



Widget Gallery



Developer Guide



Class Reference



Community and Support

- 7000+ members on the developer forum

- Books and articles

- Meta-sites (gwtsite.com, gwtPowered.org)

Libraries and Applications

- GWT Widgets on SourceForge

- 75 GWT-related projects on Google Code project hosting

- Diverse products built with GWT

 - Google Base (base.google.com)

 - Google Image Labeler (images.google.com/imagelabeler)

 - Web-based conferencing (dimdim.com)

 - Texas Hold 'em with live chat (gpokr.com)

Tools, Tools, Tools

- IntelliJ IDEA plug-in for GWT

- WindowBuilderPro GUI designer for GWT

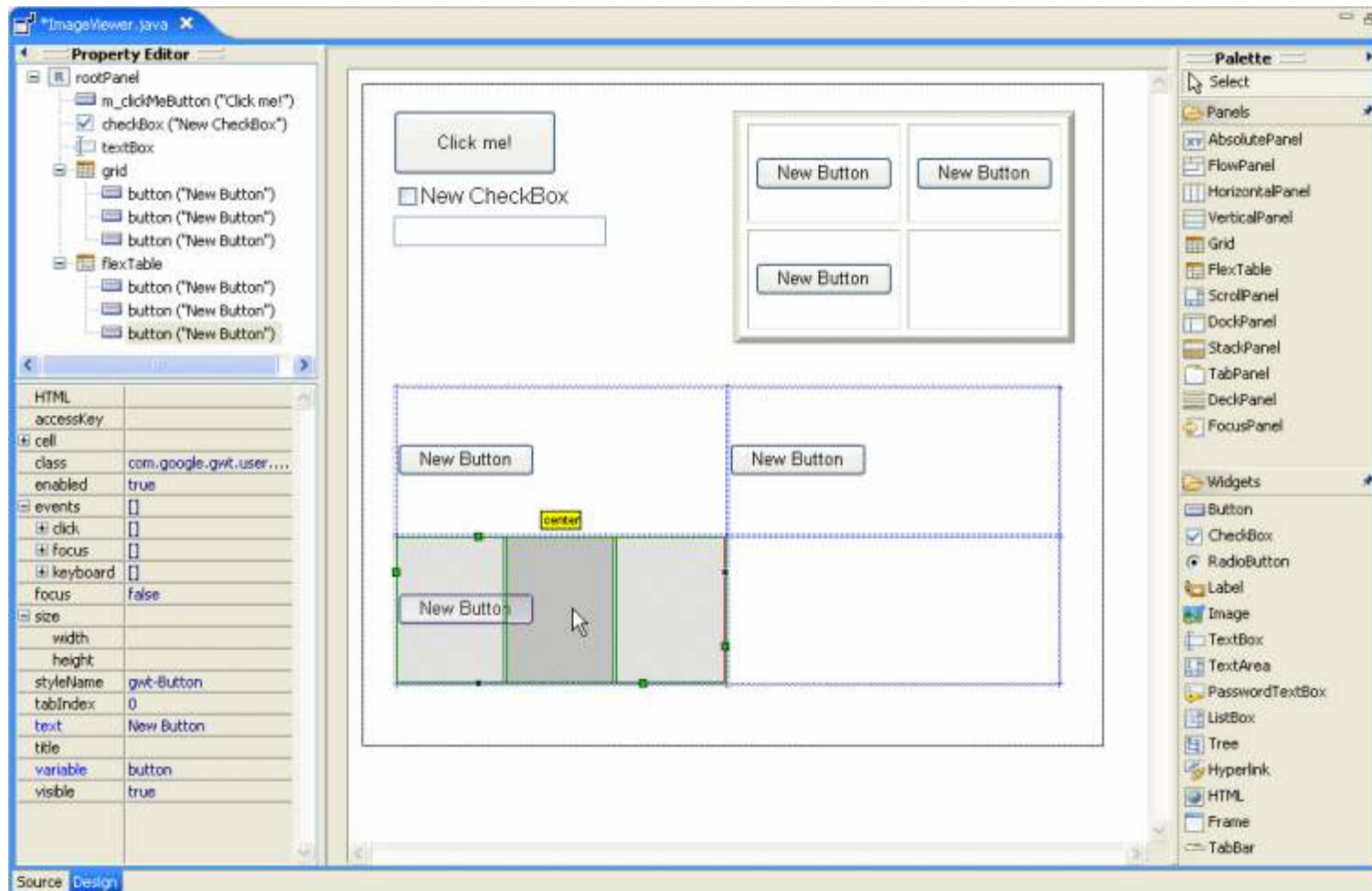
- VistaFei for GWT

- Googlipse, an open source Eclipse plug-in for GWT

Did I Mention Tools?



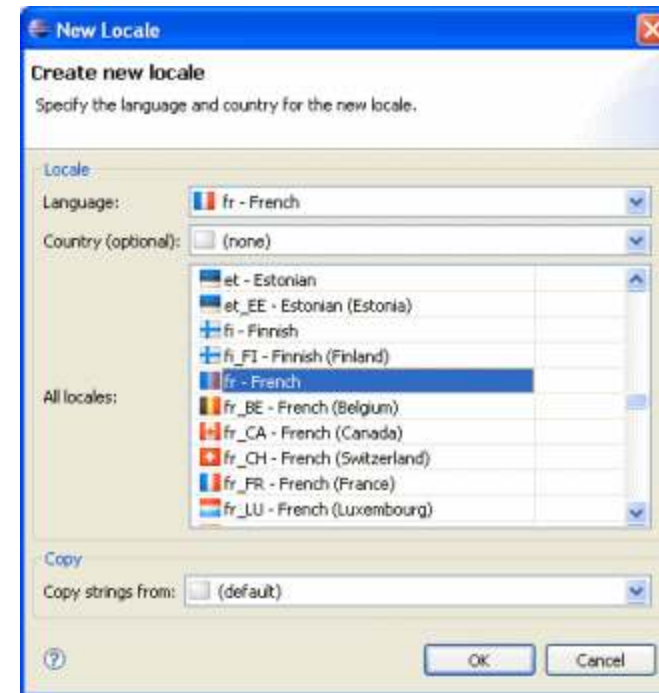
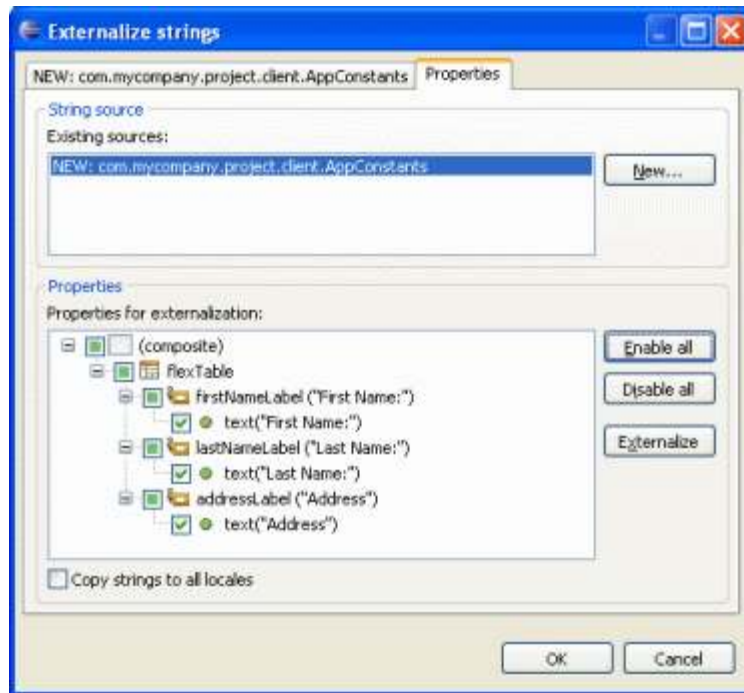
Instantiations GWT Designer WYSIWYG Layout



Did I Mention Tools?



Instantiations GWT Designer Internationalization



Leverage is needed to use AJAX well with minimum risk

PhD in browser quirks is no longer a hiring prereq

Turn AJAX development into software engineering

GWT rewards using good engineering practices

We will share our best work and ideas with you, and we hope you will return the favor

Much more to come...see you online!

A Simpler-Than-Possible Explanation of GWT

Why AJAX Matters

GWT is Software Engineering for AJAX

Fake Q & A: A Soliloquy

Big Applications

Summary

Actual Q & A

Q&A

