



Federico Kereki

Essential GWT

Building for the Web with
Google Web Toolkit 2

Developer's Library



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data

Kereki, Federico, 1960-

Essential GWT : building for the web with Google Web toolkit 2 / Federico Kereki.
p. cm.

Includes index.

ISBN-13: 978-0-321-70514-3 (pbk. : alk. paper)

ISBN-10: 0-321-70514-9 (pbk. : alk. paper)

1. Ajax (Web site development technology) 2. Java (Computer program language)

3. Google Web toolkit. 4. Application software--Development. I. Title.

TK5105.8885.A52K47 2011

006.7'6--dc22

2010018606

Copyright © 2011 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671-3447

ISBN-13: 978-0-321-70514-3

ISBN-10: 0-321-70514-9

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, IN.

First printing, July 2010

Editor-in-Chief

Mark Taub

Acquisitions Editor

Trina MacDonald

Development

Editor

Songlin Qiu

Managing Editor

John Fuller

Project Editor

Anna Popick

Copy Editor

Apostrophe Editing

Services

Indexer

Jack Lewis

Proofreader

Linda Begley

Editorial Assistant

Olivia Basegio

Technical

Reviewers

Jason Essington

Jim Hathaway

Daniel Wellman

Cover Designer

Gary Adair

Compositor

Rob Mauhar

Contents at a Glance

Preface **xv**

Acknowledgments **xix**

About the Author **xxi**

- 1** Developing Your Application **1**
- 2** Getting Started with GWT 2 **9**
- 3** Understanding Projects and Development **21**
- 4** Working with Browsers **31**
- 5** Programming the User Interface **55**
- 6** Communicating with Your Server **77**
- 7** Communicating with Other Servers **119**
- 8** Mixing in JavaScript **139**
- 9** Adding APIs **157**
- 10** Working with Servers **177**
- 11** Moving Around Files **195**
- 12** Internationalization and Localization **211**
- 13** Testing Your GWT Application **229**
- 14** Optimizing for Application Speed **259**
- 15** Deploying Your Application **287**
- Index **301**

This page intentionally left blank

Contents

Preface **xv**

Acknowledgments **xix**

About the Author **xxi**

1 Developing Your Application **1**

Rich Internet Applications **1**

Web 2.0 **2**

Cloud Computing **3**

The “Death of the Desktop” **4**

Advantages of GWT **4**

HTML Ubiquity and Browser Differences **4**

JavaScript Deficiencies **5**

Software Methodologies to Apply **5**

Classic Development Problems **5**

Agile Methodologies **7**

Forever Beta? **7**

Summary **8**

2 Getting Started with GWT **9**

Why Use GWT? **9**

Why Java? **10**

Some Actual Disadvantages **10**

The GWT Components **12**

Compiler **12**

JRE Emulation Library **14**

UI Library **17**

Setting Up GWT **17**

Writing Code **17**

Version Control Management **19**

Testing **19**

Running and Deploying **19**

Summary **20**

3	Understanding Projects and Development	21
	Creating a Project	21
	Using the Google Plugin for Eclipse	21
	Using the GWT Shell Script	22
	Project Structure	23
	Running Your Application: Development Mode	27
	Summary	30
4	Working with Browsers	31
	The Back Button Problem	31
	Setting Up Your HTML Page	32
	The History Class	33
	Starting Your Application	34
	Showing Forms in Pop-Ups	37
	Passing Parameters	38
	Creating a Menu	41
	Detecting the User's Browser	43
	The Classic Way	43
	The Deferred Binding Way	44
	Recognizing Older Explorers	52
	No JavaScript?	53
	Summary	53
5	Programming the User Interface	55
	Thinking About UI Patterns	55
	MVC: A Classic Pattern	56
	MVP: A More Suitable Pattern	57
	Implementing MVP	59
	Callbacks Galore	59
	Implementation Details	60
	Some Extensions	67
	Declarative UI	69
	A Basic UiBinder Example	70
	More Complex Examples	73
	Summary	76

6	Communicating with Your Server	77
	Introduction to RPC	77
	Implementation	78
	Serialization	79
	Direct Evaluation RPC	83
	RPC Patterns of Usage	84
	The World Cities Service	84
	Code Sharing	86
	Coding the Server Side Services	88
	Database-Related Widgets and MVP	94
	A Look at MVP	100
	A Country/State Cities Browser	101
	Live Suggestions	108
	Data Prevalidation	112
	Enterprise Java Beans	116
	Summary	118
7	Communicating with Other Servers	119
	The Same Origin Policy (SOP) Restriction	119
	Our City Update Application	121
	Receiving and Processing XML	125
	Using Ajax Directly	127
	Going Through a Proxy	129
	Producing and Sending XML	131
	Creating XML with Strings	132
	Creating XML Through the DOM	133
	Sending the XML Data	135
	Sending XML Through Ajax	136
	Sending XML Through a Proxy	136
	Summary	137
8	Mixing in JavaScript	139
	JSNI	139
	Basic JSNI Usage	140
	Hashing with JavaScript	142
	Animations Beyond GWT	143
	A Steampunk Display Widget	143

JSON	146
JSONP	153
Summary	155
9 Adding APIs	157
A Weather Vane	157
Getting Weather Data	157
Getting the Feed	159
Getting Everything Together	160
Getting at the Feed Data with an Overlay	161
Getting the Feed with JSNI	162
Dashboard Visualizations	162
Using the Google Visualization API	164
Handling Events	167
Working with Maps	168
Interactive Maps	168
Fixed Maps	173
Summary	175
10 Working with Servers	177
The Challenges to Meet	177
Before Going Any Further	177
Security	178
Ajax Problems	179
Cryptography	179
Hashing	180
Encrypting	180
Stateless Versus Stateful Servers	183
Common Operations	185
Logging In	185
Changing Your Password	190
Summary	193
11 Moving Around Files	195
Uploading Files	195
An Upload Form	195
A File Processing Servlet	200
Providing Feedback to the User	202

Downloading Files	204
A File Download Form	204
A Sample File Producing Servlet	207
Summary	209
12 Internationalization and Localization	211
Internationalization (i18n)	211
Resource Bundles	212
Using Constants	213
Messages	217
UiBinder Internationalization	219
Localization (l10n)	223
Summary	227
13 Testing Your GWT Application	229
Why Testing?	229
Advantages of Automatically Tested Code	230
And if a Bug Appears?	230
Unit Testing with JUnit	231
A Basic JUnit Example	231
Test Coverage with Emma	236
Testing MVP Code	238
Testing with Mock Objects	239
EasyMock	240
Integration Testing with GWTTestCase	247
Testing a View	247
Testing a Servlet	252
Acceptance Testing with Selenium	253
A Very Simple Example	255
What Can Go Wrong?	257
Summary	257
14 Optimizing for Application Speed	259
Design Patterns for Speed	259
Caching	260
Prefetching	263
Thread Simulation	266
Bundling Data	273

Speed Measurement Tools	277
Speed Tracer	278
YSlow	280
Page Speed	283
JavaScript Debuggers	285
Summary	286
15 Deploying Your Application	287
Compilation	287
Modules	289
Code Splitting	291
Deployment	297
Working with Client-Only GWT	297
Working with Client-Plus-Server GWT	297
Summary	300
Index	301

Preface

Developing modern, interactive, complex web sites has become a harder task since users' expectations are higher today. The bar has been raised by the current crop of applications such as Gmail or Google Maps, and developers are expected to work up to that level and provide similarly powerful new web sites. The style, speed, and interaction levels of modern sites practically rival those of classical desktop installed applications, and of course users don't want to go back. How do you develop such sites?

It can be said that the usage of Ajax was what started the trend toward such distinctive applications, but even given that technique, the rest of the development of web pages was the same, tools were the same, testing methods were the same, and the whole result was that the programmers' jobs had gotten much harder than needed.

(Personally, I should confess that I really never liked classic-style web development: Building large-sized applications was harder than it needed to be, JavaScript was—and still is—missing constructs geared to complex systems, the click-wait-click-wait again cycle was inevitably slow and not very interactive, and, to top it all, unless you were quite careful with your testing, your design was prone to fail on this or that browser in unexpected ways.)

GWT, in just a very few years, has grown into a powerful tool by harnessing the power of Java and its considerable programming environment and many development tools, and producing efficient and consistent output, despite the too-many and well-known incompatibilities between browsers.

Getting started with GWT isn't that hard—documentation is reasonably good, the development environment can be Eclipse or several other equally powerful IDEs, and programming is quite similar to old-fashioned Java Swing coding—so you can have your first short application up and running in a short time.

Creating production-quality, secure, internationally compliant, high-level code can be, however, a bit more complex. You need to take many factors into account, from the initial setup of your project and development of the user interface, to the final compile and deployment of your application.

Similarly, we'll also have to focus on methodologies and on software design patterns, so we can go forth in a safer, more organized way toward the complete application. For example, we'll consider how the model-view-presenter (MVP) pattern can not only enhance the design of the application, but also help run fully automatic tests, in modern Agile programming style, to attain higher quality, better tested software.

We'll be working with the latest tools and versions; not only GWT's (2.0.3 just now), but also Eclipse, Subversion, Tomcat, Apache, MySQL, and so on. Because all these tools

are open source, we can support the notion that an appropriate software stack can be built starting with GWT and ending with a full open web solution.

After my earlier confession on my dislike of classic web development strategies, I should now aver that GWT did change that for me. Working in a high-level setting, with plenty of tools, and practically forgetting about browser quirks, HTML, CSS, and JavaScript, while gaining in clarity, maintainability, and performance, has made web application creation an enjoyable task again!

The Structure of This Book

Chapters 1 through 3 deal with the basic setup for working with GWT. After considering the main reasons and objectives for using GWT, we'll study what other tools are required for serious code development, the methodology to use, and the internal aspects of projects.

Chapters 4 and 5 are the backbone for the book, for they deal with the basic design patterns that we use for building the User Interface. The code style and idioms developed here will be used throughout the rest of the book.

Chapters 6 and 7 deal with communications with servers, either through RPC (to connect with servlets) or through direct Ajax (to communicate with remote services).

Chapters 8 and 9 study how to add both JavaScript coding and third-party APIs to your application. Together with the previous two chapters, everything that's needed for mashing up services and getting information from different sources will have been covered.

Chapters 10 and 11 have to do with common server related problems, such as security aspects, and file upload and download.

Chapter 12 deals with developing GWT applications that will be used worldwide and covers both internationalization and localization.

Finally, Chapters 13 through 15 consider general themes such as testing GWT applications, optimizing their performance, and finally deploying them.

Who Should Read This Book

This book goes beyond “just learn GWT,” and is targeted to programmers who already have a basis of GWT programming and want to encompass other web applications, services, APIs, and standards as well, to produce Web 2.0-compliant Rich Internet Applications (RIAs). A previous experience with web development, possibly in a J2EE environment, will come in handy.

Having read this book through, the reader should not only be able to develop a RIA on his own by just using GWT, but he will also have a reference book to help solve the common problems that arise in such applications. Complete source code is given for all examples, so getting started is quicker.

Web Resources for This Book

The Google Web Toolkit site at <http://code.google.com/webtoolkit/> is a mandatory reference, and so is the forum at <http://groups.google.com/group/google-web-toolkit>.

The code examples for this book are available on the book's web site at www.informit.com/title/9780321705143.

Developing Your Application

Why would you use GWT? What can you develop with it and how? Before delving into specifics (as we'll be doing in the rest of the book) let's consider the answers to these questions, so you'll know what to focus on.

Developing applications with GWT can be seen as a straightforward job, but you should ask some interesting questions to unlock the way to powerful, distinct, applications. What kind of applications should you develop with GWT? (And, given the current push for Cloud Computing, you can even add "Where would you deploy your application?") How can you go about it? And, why would you use GWT?

Let's consider all these questions in sequence to start you on your way through this book, knowing your goal and the road to it.

Rich Internet Applications

When you start reading about Rich Internet Applications (RIAs), your JAB (Jargon, Acronyms, and Buzzwords) warning should go off because there are many words that are bandied about, without necessarily a good, solid definition or a clear delimitation of their meanings.

Basically, what we build are web applications that have the look and feel of classic desktop applications but that are delivered (and "installed") over the web. Many tools have been used for this purpose, such as Java (through applets), Adobe Flash, and more recently, Microsoft Silverlight, but used in this way, all these tools are beaten, in terms of practicality, by simple HTML-based systems.

The RIAs that we will be developing are based on JavaScript and Ajax and just require an appropriate browser to run. Classic web applications were developed with a different set of tools, subjected the user to frequent waits (the hourglass cursor was often seen), and had severe restrictions as to usability, with a much clunkier feel to them than desktop installed programs.

Although some people distinguish between RIAs and the kind of interactive web applications we build, the frontiers are getting blurrier and blurrier. You could argue that Flash or Silverlight require preinstalled plugins, or that development runs along different

lines, but in terms of the final result (which is what the user experiences) differences are not so marked, and well-designed HTML/JavaScript/Ajax applications can compete for equality with applications developed with the other tools. (Also, some people opine that HTML 5 can seriously challenge Flash, up to the point of making it obsolete, but that's still to come.¹) There used to be obvious differences—the ability to store local data at the user's machine was the biggest one—but tools such as Google Gears or current developments in HTML 5 have provided this feature to web applications.²

Given its ubiquity (from desktops to netbooks, and from cell phones to tablet PCs) the browser can be considered a universal tool, and Ajax provides the best way for the creation of highly interactive applications. Of course, a few years ago there weren't many tools for doing this (GWT itself appeared in 2006) and creating heavy-lifting interactive code with just JavaScript wasn't (and still isn't) an appealing idea.³

Furthermore, given that users have been subjected for many years to web applications, and are familiar with their idioms, you are a bit ahead in terms of user interface design by keeping to a reasonable standard.

As for the language itself, using Java as a tool—even if it gets compiled into JavaScript, as GWT does—provides both a way around JavaScript's deficiencies and introduces a widely used language with plenty of development tools, which has been used over and over for all kinds of applications and has been proved to scale to large-sized applications.⁴

Web 2.0

Web 2.0 is another expression that has been bandied about a lot since its invention in 2004. Though there are way too many definitions for it, most seem to agree on the idea of using the “Web as Platform,” where all applications run in a browser instead of being preinstalled on your desktop. Furthermore, the idea of allowing users to produce their own contents (à la Wikipedia) is also included, highlighting the collaborative aspect of work, and thus bringing into the fold all kind of community and social networking sites (think Facebook or YouTube). Finally (and that's what actually works for us) the concept of *mashing* together different data sources (probably from many web services) is also included.

1. See www.ibm.com/developerworks/web/library/wa-html5webapp/ for an article of some HTML 5 features already available in current browsers.

2. Google Gears' development was practically stopped (other than support for currently available versions) by the end of 2009 because of the upcoming HTML 5 features for local storage.

3. It might be said that developing large applications with, say, Flash, isn't a walk in the park either, for different reasons to be sure, but complicating the programmer's job in any case.

4. It should be remarked that GWT isn't the only such compile-to-JavaScript solution; for example, the Python-based Pyjamas project (<http://code.google.com/p/pyjamas/>) provides Python-to-JavaScript translation, and there are many more similar tools.

GWT applications can obviously be used for producing highly interactive people sites, but they can also link together information from different origins, consuming web services with no difficulty, either connecting directly to the server or by means of proxy-based solutions. Various data formats are also not a problem; if you cannot work with such standards as XML or JSON, you can include external libraries (or roll out your own) through JSNI or Java programming. (We cover this in Chapter 8, “Mixing in JavaScript,” and Chapter 9, “Adding APIs.”)

In this context, the phrase Service-Oriented Architectures (SOA) frequently pops up. Instead of developing tightly integrated, almost monolithic, applications, SOA proposes basing your systems on a loosely integrated group of services. These services are general in purpose and can be used in the context of different applications—and, as previously mentioned, GWT is perfectly suited to “consuming” such services, dealing with different protocols and standards. (We’ll cover this in Chapter 6, “Communicating with Your Server,” and Chapter 7, “Communicating with Other Servers.”) If your company is centered on an SOA strategy, your GWT-developed applications will fit perfectly well.

Cloud Computing

Next to the idea of using the browser as the basis for the user’s experience, the most current term related to modern application development is *Cloud Computing*. This idea reflects the concept of sharing resources over the web, on demand, instead of each user having a private, limited pool of resources. In this view, software is considered a “service” (the acronym SAAS, which stands for “Software as a Service,” is often used) and a resource similar to more “tangible” ones as hardware.

(As an aside, the vulnerability of some operating systems, most notably Windows, to viruses, worms, and similar attacks, has given a push to the idea of using a simple, secure, machine and storing everything “on the web,” letting the cloud administrators deal with hackers and program infections.)

For many, this concept is yet another cycle going from centralized resources (think mainframes) to distributed processing (PCs, possibly in client/server configurations) and now to having the web as your provider. The main requirements for such an architecture involve reliable services and software, delivered through specific data centers, and running on unspecified servers; for the user, the web provides an access to a cloud of resources.

For GWT applications, your applications are basically destined from the ground up to be used “in the cloud” because of the standard restrictions imposed by browsers. Distributing an application over the web, accessing it from anywhere, and having your data stored in a basically unknown place are all characteristics of any applications you might write.⁵

5. With current (or forthcoming) standards, you might also resort to storing data locally, or to using your own private, dedicated, resources, but that’s not original and more often associated with classic desktop applications.

The “Death of the Desktop”

The trend toward Cloud Computing has even spawned a new concept: the “Death of the Desktop.” This presents rather starkly the problem of going overboard, to the limit: From the appearance of mini netbooks (with flash-based disks, slow processors, not much RAM) and iPhone-look-alike cell phones, some have reached the conclusion that desktop applications (and even desktop computers!) are on their way out. If this were true, it could be great for GWT developers, but things are a bit different.

Despite several impressive opinions and pronouncements from people all over the industry, the trend toward more powerful machines, with CPUs, memory, and I/O facilities that put to shame the supercomputers of just a few years ago, doesn’t seem to be slowing down. Even if you are enamored with the latest netbooks or high-powered cell phones, you should accept that working all the time with minimal screens isn’t the way that things can get done at a company. (And for gaming or graphic-intensive usages, small machines aren’t so hot either; they may do, however, for business-oriented applications.) In any case, GWT can help you because you can use its layout facilities and CSS styling to produce applications for just about any device out there.

Also, remove the rosy glasses for an instant. Cloud computing offers several advantages (and GWT applications can be considered to be right in the middle of that concept) but also presents problems, so you need to plan accordingly. Aside from the obvious difficulty of dealing with possibly flaky web connections, security and compatibility can be stumbling blocks. (On the other hand, scalability is well handled; there are plenty of large sites, with hundreds or thousands of servers, proving that web applications can scale well.) The important point is, with or without desktops, GWT provides some ways around these kind of problems, and we’ll study this in upcoming chapters.⁶

Advantages of GWT

Why would you develop with GWT? Shouldn’t directly using JavaScript make more sense? How do you manage with browser quirks? Let’s consider the reasons for GWT.

HTML Ubiquity and Browser Differences

The first reason for GWT applications is the ubiquity of HTML. Even if some time ago browsers for, say, cell phones, weren’t as capable as their desktop brethren, nowadays you can basically find the exact same capabilities in both. In terms of GWT, this is a boon because it means that a well-designed application can run and look pretty in devices from 3 inches to 25 inches.⁷

6. And, of course, these inconveniences haven’t stopped anyone from developing HTML-based applications!

7. Don’t expect to get the screen design right the first time; managing to build clear, small screen browser applications is more an art than a science.

This availability is somehow tempered because today's browsers are not created equal—but you certainly knew that if you designed web pages on your own! When Microsoft's Internet Explorer ruled the roost, having practically 100% of the browser market, this wasn't a noticeable problem. However, today browser usage statistics point to a different status quo: Mozilla Firefox and Safari, among others, have started carving larger and larger niches in the market, and in some countries (mostly European) they have outnumbered Internet Explorer. The current trend is toward applying web standards, and that bodes well for web developers. In any case, GWT is quite adept at solving browser quirks and differences, so the point may be considered moot for the time being.

JavaScript Deficiencies

Even assuming fully standard-compliant browsers, the fact remains that JavaScript, no matter how powerful, isn't a good language from the specific point of view of software engineering. Because this isn't a book on JavaScript, we won't delve in its main problems, but using it for large-sized application development can be, to say the least, a bit complicated.

This language isn't well adapted either to development by large groups of people, and the tools it provides for system development aren't that adequate, so the programmer must add extra code to bridge the distance between a modern object-oriented design and its actual implementation.

One solution that has been applied is the usage of different libraries that provide a higher-level way of using the language.⁸ GWT solves this problem in a radically different way, by enabling the use of the higher level Java language, for which there are plenty of modern development, testing, and documentation tools.

Software Methodologies to Apply

For classic application development, many well-known methodologies exist, but in the context of modern web development, you should definitely use some techniques.

Classic Development Problems

If you learned to develop systems years ago, you were surely exposed to the Waterfall Model or some other methodologies directly based on it. In this model for the development process, progress is seen as flowing like a waterfall from stage to stage, through

8. You could consider Google's "Closure" library (see <http://code.google.com/closure/>) used for Gmail's development, or Yahoo!'s YUI library (see <http://developer.yahoo.com/yui/>), jQuery (<http://jquery.com/>), Dojo (www.dojotoolkit.org/), Prototype (www.prototypejs.org/), MooTools (<http://mootools.net/>), and many others. The functionality of these libraries isn't always the same, but there's considerable overlap between them, showing the problems they set out to solve are real and well known.

well-defined phases (see Figure 1.1) starting with the Analysis of Requirements, following with the Design of the Solution and its Implementation, then to Testing (or Quality Assurance), and finally to Installation and future Maintenance.

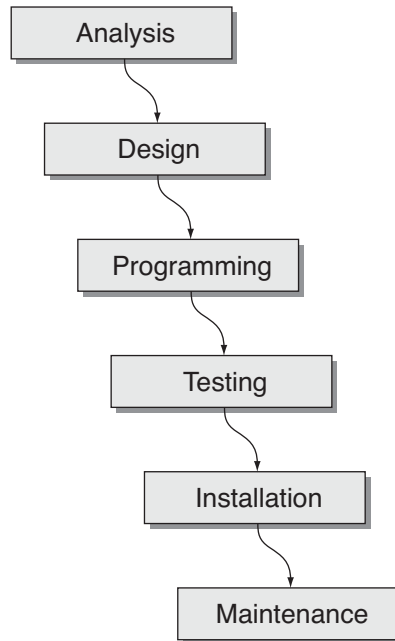


Figure 1.1 The classic Waterfall Model isn't the best possible for GWT development.

This model is flawed in several ways (and of course, there are some fixes for that) but its main problem is its orientation to highly regimented industries such as Construction, in which late changes can be quite costly to implement, usually requiring tearing down what was done and practically starting anew.

Another point—and an important one—is that you cannot expect users to be fully aware of what they require; it is sometimes said “Users don't know what they want, but they know what they don't want.”⁹ Classical methodologies do not take this into consideration, and might thus incur important costs, because newly discovered or determined requirements can invalidate a previous design.

Finally, it's difficult to predict where difficulties will occur; problems with functionality are usually found “on the go,” and if going back to change something to help future development is too costly, you can face a dilemma: Spend money and time revising your

9. “I'll know it when I see it” is another way of expressing this.

design, or keep your substandard design, and spend money and time later trying to make your software do tasks it wasn't well designed to do.

It has been said that the Waterfall Model, and similar ones, are based on the old "Measure Twice, Cut Once" saw, but you cannot actually apply this when you don't actually know what's being measured! (And, furthermore, what happens if requirements change along the way, and by the time you finish with development, the problem has actually changed?) Modern, agile technologies try to take this into account and work in a radically different way, and that's the way you should use with GWT.

Agile Methodologies

Several software development methodologies seek to reduce the time between the requirement analysis phase and the development phase to develop at least parts of the system in shorter times, using possibly an iterative method to advance to the final application. Prototypes are frequently used to bridge the distance between the user and the developer, helping both to understand what's actually required. Instead of attempting to do a whole system at once, development is parceled in smaller subsystems. The user is involved all the time, instead of providing his input (in the form of requirements) only at the beginning and then dealing with the system after its installation.

All these suggestions are currently applied in Agile Software Methodologies (born in 2001) that emphasize collective (i.e., users plus programmers) development of systems, in highly iterative steps, with frequent verification and (if needed) adaptation of the written code.

Agile Methodologies usually break a complex system into several short stages, substituting short, easily measured and controlled iterations, for long-term (and hard to do) planning. Each iteration (usually shorter than a month) involves a mini development cycle that includes all the stages associated with a Waterfall Model but finishes with giving the users a working product with increasing functionality that serves not only as a measure of advance, but also as an aid to determine if changes are needed. The delivered software is used as the main measurement for progress, instead of depending on a Gantt chart or other documents.

GWT is perfectly suited to such methodologies, because it can offer iterative development, rapid prototyping (and here tools such as UiBinder, which we will study, can help quickly develop appropriate interfaces), and automated testing. The latter point is particularly important: Given that development can (and will) go back and forth, and code used in a previous iteration can be modified several times along the complete development process, it's important to check whether old functionality hasn't been lost and whether bugs have been introduced. GWT has tools that provide for both unit testing (at the lowest level) and acceptance testing (at the user level).

Forever Beta?

As a side effect of the iterative development process, it's usually hard to define what constitutes a "version" of the final system. Because practically every iteration produces new

functionality, and the final goal isn't as well defined as with classic methodologies (in which the complete roadmap is laid out at the beginning and then preferably left unchanged) with iterative development, you deliver the system in many small steps, rather than in large ones.

In this context, it's not unknown for systems to be considered in "perpetual beta"; beta testing refers to the tests done by actual users with a system that is close to the full product but not necessarily complete. (An extreme case of this is Google's Gmail, which was considered to be at beta level from 2004 to 2009!) With GWT, you can provide functionality increases in short steps, and the web model enables for easy distribution of the updated code.¹⁰

Summary

We touched upon several considerations that impact web application development. In the rest of the book, we will be elaborating on them and provide specific techniques to help you develop company-sized RIAs with the expected levels of quality and functionality.

10. This could be said, of course, of any web-based application not necessarily written with GWT; the point is that GWT helps you work this way.

Index

Numbers and Symbols

007. See **Bond, James**

\$ (' a '), as selector, **142**

A

Acceptance testing, with Selenium

example of, 255–256

overview, 253–255

potential problems, 257

"Access Control Specification," 121

Accounting, security and, 178

Accuweather, 158

actualUrl value, 175

addCity(...) function

coding server side services and, 90

WorldService remote service and, 85

addValueChangeListener method, 97

aDirectory, compilation and, 287, 288, 289

Agile Software Methodologies, 7, 10

Ajax

caching and, 260

ExternalTextResource types and,
277

receiving/processing XML and,
127–128

security controls and, 179

sending XML via, 136

stateless server-side coding and,
183–184

AjaxLoader API, 160–161`<all>...</all>` construct, 45

Albany, a city in NY, USA, 108, 175

Alt+Backspace/Back button problem

creating menus, 41–43

displaying forms in pop-ups, 37–38

History class, 33–34

overview, 31–32

passing parameters, 38–41

setting up HTML page, 32

starting application, 34–37

American Museum of Natural History, 168–170, 173–174`andReturn(...)` method, 245–246`animateAllLinks(...)` function, 143

Ángel S. Adami, 158

Animations, JSNI and, 143

Annotations, 215

Antique browsers, 52–53, 120–121

`<any>...</any>` construct, 45**Apache**

client-only GWT deployment and, 297

Commons Lang component, 183, 200–202

Tomcat servlet container, deployment and, 297–300

`appendChild(...)` method, 134–135**Application development**

with client-only GWT 2, 297

with client-plus-server GWT 2, 297–300

code splitting, 291–297

compilation, 287–289

getting started, 20

modules, 289–291

overview, 287

summary, 300

Application development

GWT advantages, 4–5

overview, 1

Rich Internet Applications and, 1–4

software methodologies for, 5–8

summary, 8

Application Programming Interfaces (APIs), adding

dashboard visualizations, 162–168

overview, 157

summary, 175

weather vane, 157–162

working with maps. *See* Maps**Application speed, optimizing**design patterns for. *See* Design patterns, for speedmeasurement tools for. *See* Speed measurement tools

overview, 259

Application testing

acceptance testing with Selenium, 253–257

integration testing with `GWTTestCase`, 247–253JUnit and. *See* JUnit testing

overview, 229

reasons for, 229–231

summary, 257

ARCFOUR, 180–183

AreaChart objects, 164, 167

arguments, JSNI and, 140

Assembly language, 139

`assertArrayEquals(...)` methods, 235`assertFalse(...)` methods

EasyMock and, 245

`GWTTestCase` and, 251

JUnit testing and, 236

AsyncCallback function
 callbacks and, 59
 coding server side services and, 89
 JSONP and, 154–155

Authentication, security and, 178

Authorization, security and, 178

Automatic testing of GWT. See Application testing

Automatically tested code, 229–231

Availability, security and, 178

B

Back button/Alt+Backspace problem
 creating menus, 41–43
 displaying forms in pop-ups, 37–38
 History class, 33–34
 overview, 31–32
 passing parameters, 38–41
 setting up HTML page, 32
 starting application, 34–37

baseUrl method, 136

Beta testing, 7–8

Bond, James, 144–145, 226

Browser recognition
 classic way, 43–44
 deferred binding way, 44–47
 disabled JavaScript and, 53
 of older IE, 52–53
 overview, 43

Browser(s)
 based measurement tools. *See* Speed measurement tools
 Country/State cities, 101–108
 differences, 4–5
 GWT Developer Plugin to operate, 28–30
 HTMLUnit, 247

older, 52–53, 120–121
 security, SOP restriction and, 119–121
 "Web as Platform" and, 2
 XML parser, 125–127

Browsers, working with
 Back button problem. *See* Back button/Alt+Backspace problem
 code generation, 47–52
 detecting user's. *See* Browser recognition overview, 31
 summary, 53

BufferedReader function, 129

Bug prevention, 229–231. See also Security; Security, servers and

Bundles, resource
 annotations and, 215
 internationalization and, 212–213
 localization and, 224–227
 UiBinder-based internationalization and, 219–223
 using constants and, 213–214

Bundling data, for application speed, 273–277

Button function, JSON and, 147–150

C

Caching

application speed and, 260–263
 prefetching and, 264–265

Callbacks

EasyMock testing and, 245–246
 enabling/disabling Login button and, 67–69
 GWTTestCase testing and, 252–253
 JSON usage and, 149–151
 JSONP and, 153–155
 Login button, 186–187
 MVP implementation and, 59–60

Callbacks (*continued*)

- Presenter, 98
 - uploading files and, 197–200
- callback=yourownfunction(...)
 - parameter, 153
- calledName variable, 242
- Calls, from JavaScript, 140–141**
- Capture<.> class, 244–245
- Cascading Style Sheets (CSS), 113–115**
- Challenges, security**
 - AAA for, 178
 - Ajax problems, 179
 - full SSL security and, 177–178
 - overview, 177
- changePassword(...) method, 192–193
- Changing passwords, security and, 190–193**
- CheckStyle plugin, 18**
- Cities Browsing class, 292–293**
- Cities updating application, 121–125**
- CITIES_AT_A_TIME constant, 268–269
- CitiesBrowserView.ui.xml file, 102–104, 106–108
- CITIES_DELAY_IN_MS constant, 268–269
- CITIES_PAGE_SIZE constant, 103, 106–107
- City browser application**
 - sample form, 101
 - Selenium testing and, 255
 - thread simulation and, 266–270
- <city> element
 - city update application and, 123–124
 - creating XML and, 132
- City input form, 112–113**
- cityExists(...) function
 - coding server side services and, 91
 - WorldService remote service and, 85

- cityList function, 127

Classes

- java.lang package, 14
- java.sql package, 15
- java.util package, 15–16
- Classic browser detection, 43–44**
- Classical methodologies, 5–7**
- classname, JSNI and, 140
- clearAllCities(...) method, 123
- clearCities(...) method, 105–106
- Client-only GWT, deployment with, 297**
- Client-plus-server GWT, deployment with, 297–300**
- ClientBundle interface, 273–277**
- ClientCityData classes**
 - code sharing and, 86–88
 - coding server side services and, 90, 92
- Cloaking, 10**
- Closure Library, 5**
- Cloud Computing, 3**
- Code, automatically tested, 229–231**
- Code generators, 47–52**
- Code Inlining**
 - compiler and, 13
 - JSON usage and, 152
- Code sharing, 86–88**
- Code splitting**
 - application deployment and, 291–297
 - compiler and, 12
- Code writing, 17–18**
- codeDecode(...) method, 181–182
- Codes, pattern, 225–227**
- Command objects, 41–43, 47–52**
- Command pattern, 267, 270**
- Common operations, security**
 - changing password, 190–193
 - logging in, 185–190

Communicating with other servers

- city update application, 121–125
- overview, 119
- producing XML, 131–135
- receiving/processing XML, 125–131
- sending XML, 135–137
- SOP restriction and, 119–121
- summary, 137

Communicating with your server

- introduction to RPC. *See* Remote Procedure Calls (RPC), introduction
- RPC patterns of usage. *See* Remote Procedure Calls (RPC), usage

Compilation, deployment and, 287–289**Compile process, in GWT 2, 287–289****Compile Reports tool, 293–296****Compiler, Java-to-JavaScript, 12–14****-compileReport, compilation option, 287****Complex UiBinder examples**

- dealing with constructors, 74–75
- presetting properties, 73
- using your own widgets, 73–74
- working with complex layouts, 75

Components

- compiler, 12–14
- JRE Emulation Library, 14–16
- overview, 12
- UI library, 17

Components tab, 281–282**Composite widgets**

- Country/State, 101–108
- interactive maps and, 171
- MVP and, 95

Confidentiality, security and, 178**Constant Folding, 13****Constants interface, 213–214****ConstantsWithLookup interface**

- internationalization and, 213–214
- translating error codes and, 215–217

Constructors

- code sharing and, 86–88
- dealing with, 74–75
- invoking Java, 141

Controller role, in MVC, 56–57**<coords> element**

- city update application and, 123–125
- creating XML and, 132–134

Copy Propagation, 13**Country/State cities browser, 101–108****CountryState object, 260–263****CountryStateView widgets**

- Country/State cities browser and, 102–104, 107–108
- UiBinder code and, 95–97

createElement(...) method, 134–135**CreateMock(...) methods, 243****createTextNode(...) method, 134–135****Creating XML**

- overview, 131–132
- with strings, 132–133
- through DOM, 133–135

Cryptography

- encryption, 180–183
- hashing, 180
- hashing with JavaScript, 142–143
- overview, 179

CssResource elements, 274**Currencies, localization and, 226****currentRow variable, 268–269****CustomFieldSerializer class, 80**

D**Darwin, Charles, 144, 225, 233–234****Darwin (cities), 122–124, 252–253, 256**

Dashboard visualizations

- Google Visualization API, 164–167
- handling events, 167–168
- overview, 162–164

Data

- bundling, 273–277
- prevalidation, 112–116

Data tables, 165–167**Data transfer object (DTO), 186–189****Database-related widgets, 94–100****DataResource elements, 274–276****Date and Time formats, 224–226****Dates, serialization of, 80****Dead Code Elimination, 12****Dead For Now (DFN) code splitting, 291****"Death of the Desktop" concept, 4****Debuggers, JavaScript, 285–286****Declarative UI**

- basic UiBinder example, 70–73
- complex UiBinder examples, 73–75
- overview, 69

Default value, 215, 218**defaultLocale attribute, 220****@DefaultStringValue(...) function, 215****Deferred binding replacement technique**

- browser detection with, 44–47
- dashboard visualizations and, 164
- using constants and, 214

Deferred commands, 270–273**delayTestFinish(...) call, 252–253****Demeter, Law of, 62****Dependency Injection, 56****description attribute, 221–223****deserialize(...) method, 82****Design patterns, for speed**

- bundling data, 273–277

caching, 260–263

overview, 259

prefetching, 263–266

thread simulation. *See* Thread simulation**Desktops, 4****Developing GWT applications. *See* Application development****Development mode, 27–30****DFN (Dead For Now) code splitting, 291****Direct Evaluation RPC (deRPC), 83–84****disableLogin(...) method, 67****Display interface**

- changing password, 191
- city update application, 123
- file download form, 205
- interactive maps, 168–169
- uploading files, 197

displayCities(...) method

- Country/State cities browser and, 105–107
- processing XML using Ajax and, 128
- producing/sending XML and, 132

displayEmptyCities(...) method, 105–107**displayNews(...) method, 152****<div>**

- interactive maps and, 170
- UiBinder and, 220–221
- widgets and, 145

<div id> function, 144–146**\$doc, 142****Document Object Model (DOM)**

- creating XML through, 133–135
- GWTTestCase and, 251

doGet(...) method

- file producing servlet and, 207–208
- providing feedback and, 202–204

Dojo Toolkit, 5, 11
DOM. See **Document Object Model (DOM)**
DomEvent.fireNativeEvent(...)
 method, 251
doPost(...) **method, 201–202**
Downloading files
 file download form, 204–207
 file producing servlet, 207–208
 overview, 204
-draftCompile, compilation option, 287, 289
DragonFly debugger, in Opera, 285–286
drawZoomAndCenter(...) **method, 172–173**
DTO (Data transfer object), 186–189
Dummy objects, 240

E

-ea, compilation option, 287
EasyMock testing, 19, 240–247
EclEmma plugin, 19, 236–238
Eclipse
 debugger, and JSNI, 140
 JUnit testing and, 234, 236–238
 for writing code, 17–18
Einstein, Albert, 234
EjbAccess remote servlet, 116–117
--enable-extension-timeline-api
 parameter, 278
enableLoginButton(...) **method, 67–68**
Encryption
 defined, 179
 security and, 180–183
Enterprise Java Beans (EJB), 116–118
<entry-point> **element**
 modules and, 290
 project structure and, 25–26

Enumerations, serialization of, 80
Environment object
 changing passwords and, 192
 EasyMock testing and, 240–247
 MVP implementation and, 60–63, 66–67
Error codes, translating, 215–217
Exceptions
 GWT 2 and, 13
 java.lang package, 14
 JavaScript code and Java, 141
 java.util package, 15
execute(...) **method, 272–273**
Extensible Markup Language (XML)
 city update application and, 123–125
 creating, overview, 131–132
 creating through DOM, 133–135
 creating with strings, 132–133
 receiving and processing, 125–131
 sending, overview, 131–132, 135–136
 sending through Ajax, 136
 sending through proxy, 136–137
ExternalTextResource elements, 274–275, 277
-extra, compilation option, 287
Extreme Programming (XP), 229

F

fail(...) **methods**
 EasyMock testing and, 243
 JUnit testing and, 236
Fake objects, 240
Feed, weather, 159–160
Feedback information, 202–204
File processing servlet, 200–202
File producing servlet, 206

Files, moving

- downloading, 204–208
- overview, 195
- summary, 209
- uploading. *See* Uploading files

FileUpload form, 195–200**final attributes, 80****finishTest(...) call, 252–253****Firebug**

- debugger, 285–286
- Page Speed and, 283–285
- YSlow and, 280–282

Firefox

- cross scripting request and, 135
- Firebug, 280–286
- SOP restriction and, 121

firstResultPosition attribute, 147**Fixed maps, 173–175****Flash library, 164****FlexTable function, 170****Floating point numbers, 13****FormPanel parameters, 196–198****Forms**

- city browser, 101, 255
- ClientBundle sampler application, 276
- file download, 204–207
- file upload, 195–200
- passing parameters to, 38–41
- in pop-ups, 37–38
- UiBinder-based internationalization, 219–223

Full code size value, 295

G

Generators, code, 47–52**Generic resource bundles, 212–213****GenericServiceReturnDto class, 187–188****GeoNames, 158****GET calls**

- file download form and, 205–206
- processing XML using Ajax and, 127–128
- providing feedback and, 203–204
- SOP restriction and, 121

getAge(...) method, 152**getAndDisplayCities(...) method, 269, 273****getAttributeNode(...) method, 127****getCities(...) function, 85****getCityName(...) method, 109****getCityPopulation(...) method**

- city update application and, 123
- producing/sending XML and, 131–132

getCountries(...) function

- caching and, 261

- database-related widgets and, 97

- WorldService remote service and, 85

getCountryState(...) method, 102, 104–107**getDescription(...) method, 141****getDisplay() method, 131–132****getDocumentElement(...) method, 125–127****getElementsByTagName(...) method, 127****getFeed(...) routine, 162****getFormat(...) method, 225****getFromUrl(...) method, 129****getLatitude(...) method, 171****getLongitude(...) method, 171****getModel(...) method**

- EasyMock testing and, 243

- MVP implementation and, 60

- getModuleName(...) method, 250
- getName(...) method, 141
- getName(...) method
 - EasyMock testing and, 245–246
 - file processing servlet and, 202
- GetNewsCallback(...) function, 149–150
- getNodeValue(...) method, 127
- getPassword(...) method, 245–246
- getSelections(...) method, 168
- getSessionKey(...) method, 187–189
- getSize(...) method, 202
- getSomething(...) method
 - logging in and, 185–187
 - MVP implementation and, 62–63, 67
- getStates(...) function
 - database-related widgets and, 97
 - WorldService remote service and, 85
- getSummary(...) method, 152
- getText(...) call, 277
- goBack(...) method, 59–60
- Google AJAX Feed API, 159
- Google Chart API, 163
- Google Chrome
 - Speed Tracer and, 278–280
 - Tomcat-deployed application, 297–300
- Google Gears, 2
- Google Maps, 168
- Google Plugin for Eclipse
 - coding server side services and, 89
 - GWT project creation with, 21–22
 - UiBinder templates and, 70
 - for writing code, 17–18
- Google Testing Blog, 229
- Google Visualization API, 163, 164–167
- Google Web Toolkit 2. See **GWT 2 (Google Web Toolkit 2), getting started**
- google.feeds variable, 162
- goto. statements, 257
- Grade tab, 281
- GreetingServiceImpl class, 81
- GWT 2 (Google Web Toolkit 2), getting started**
 - advantages/disadvantages, 9–11
 - components, 12–17
 - defined, 9
 - setting up, 17–20
 - summary, 20
- GWT advantages**
 - HTML ubiquity/browser differences, 4–5
 - Java, 10
 - JavaScript, 5
 - overview of, 9–11
- GWT AjaxLoader API**
 - getting weather feed and, 160
 - steps for using, 160–161
- GWT Developer Plugin, 28–30**
- gwt.ajaxloader.jar files, 160–161
- GWT.create(...)
 - creating widgets with, 74
 - invoking messages with, 217
- GWT.getHostpageBaseURL(...) function, 128
- GWT.runAsync(...) method, 291–293
- gwttest directory, 23
- GWTTestCase, integration testing and
 - overview, 247
 - setup times, 254
 - testing login view, 247–251
 - testing servlets, 252–253
- gwt.xml files
 - creating modules and, 290–291
 - Google Visualization API and, 164
 - GWT AjaxLoader API and, 160–161
 - GWTTestCase testing and, 252

H

Hashing

- changing passwords and, 191–193
- defined, 179
- with JavaScript, 142–143
- logging in and, 185–190
- security and, 180

`hashword.length(...)` method, **180**

Hints mode, 279–280

History class, 32, 33–34

Host, 119–120

HTML (HyperText Markup Language)

- setting up page, 32
- ubiquity of, 4–5
- widgets, 46–47

HTMLPanel function, 147–149

HTMLUnit web browser, 247

Humble Dialog (Humble Object), 56

Hýbl, Cestmír, 144, 146

Hyperlink widgets, 43

HyperText Markup Language. See HTML (HyperText Markup Language)

I

i18n. See Internationalization (i18n)

IE. See Internet Explorer (IE)

ImageBundle interface, 273–274

ImageResource elements, 274–275

IncrementalCommand function, 271

<inherits> element, 25

initialize methods, 81

initializeWithString(...) method, 40

instance objects, 140

instance.@classname::field, 141

Integration testing, with GWTTestCase
overview, 247

testing login view, 247–251

testing servlets, 252–253

Interactive maps, 168–173

Internationalization (i18n)

- annotations tricks, 215
- bundling data and, 274
- messages and, 217–219
- overview, 211–212
- resource bundles and, 212–213
- summary, 227
- translating error codes, 215–217
- UiBinder, 219–223
- using constants, 213–214

Internet Explorer (IE)

- recognizing old versions of, 52–53
- SOP restriction and, 120–121

IsSerializable interface, 86–88

J

jar file, 290

Java

- advantages of, 10
- JavaScript interaction with, 139–141
- server-side code, 88–94
- UiBinder and, 72–73

Java Cryptography Architecture (JCA), 180

Java-to-JavaScript compiler, 12–14

Java Virtual Machine parameters, 140–141

java.io package, 14

java.lang package, 14–15

JavaScript

- debuggers, 285–286
- deficiencies of, 5
- disabled, 53
- Java interaction with, 139–141
- stateless server-side coding and, 183–184

JavaScript library

dashboard visualizations and, 164
loading, 160, 161

JavaScript, mixing in

JSNI and. *See* JavaScript Native Interface (JSNI)

JSON. *See* JavaScript Object Notation (JSON)

JSONP, 153–155

overview, 139

summary, 155

JavaScript Native Interface (JSNI)

basic usage of, 140–141

browser detection and, 44

getting feed with, 162

hashing with, 142–143

overview, 139–140

Steampunk display widgets and, 143–146

JavaScript Object Notation (JSON)

feed data, 161

news reader completion using, 148–153

news reader view using, 147–148

overview, 146–147

weather information and, 158–159

JavaScript Object Notation with Padding (JSONP), 153–155

JavaScriptException objects, 141

JavaScriptObject function, 170

java.sql package, 15

java.util package, 15–16

JCA (Java Cryptography Architecture), 180

Jetty web server, 79

Johnston, Paul, 142

jQuery JavaScript Library, 5, 11, 143

JRE Emulation Library

java.io package, 14

java.lang package, 14–15

java.sql package, 15

java.util package, 15–16

JSLint, 282
JSMin, 282

JSNI. *See* JavaScript Native Interface (JSNI)

JSON. *See* JavaScript Object Notation (JSON)

JSONP (JavaScript Object Notation with Padding), 153–155

JSONParser methods, 151

JsonpRequestBuilder class, 154–155

JsonUtils.escapeValue(...) method, 147

JsonUtils.unsafeEval(...) method, 151

JUnit testing

basic example of, 231–236

EasyMock and, 240–247

with mock objects, 239–240

MVP code testing, 238–239

overview, 19, 231

test coverage with Emma, 236–238

K

@Key(...) annotation

key attribute and, 220

resource bundles and, 215

key attribute, 219–223

Keys

annotations tricks and, 215

resource bundles and, 212–213

translating error codes and, 216

KeyValueType class

EclEmma coverage test with, 236–238

JUnit testing of, 231–236

module for, 290–291

L

l10n. See **Localization (l10n)**

Launcher, improved, 37–38

Layouts, complex, 75

Lazy evaluation, 100–101

Least recently used (LRU) logic, 262

Libraries

- Closure, 5
- Flash, 164
- JavaScript, 160, 161, 164
- jQuery, 5, 11, 143
- JRE Emulation, 14–16

Lincoln, Abraham, 144–145, 148, 225, 233–234

LinkedHashMap(. . .) method, 93–94

Linux

- client-only GWT deployment and, 297
- SOP restriction and, 121

ListBox widgets, 93, 94–99

Live suggestions, 108–112

Loading.texts

- bundling data and, 277
- thread simulation and, 268–269

Localization (l10n)

- overview, 211
- process of, 223–227
- summary, 227

-localWorkers, compilation option, 288, 289

Logging in, security and, 185–190

Login button, 67–69

Login procedure, 34–35

Login service, 242–247

Login view, 247–251

LoginFormPresenter class, 60, 62, 64, 66

LoginFormView class

- MVP implementation and, 60, 64–66
- UiBinder and, 70, 72, 74

LoginFormView.ui.xml files, 70, 72

loginServiceMock(. . .) function, 243–244, 246

LoginView class, 60–61, 63

- logLevel, compilation option, 288

long variables, 13

<longitude> method, 134–135

LRU (least recently used) logic, 262

M

Magic naming, 78

Management Information Systems (MIS) applications, 162

Maps

- fixed, 173–175
- interactive, 168–173
- overview, 168

MD5 (Message-Digest algorithm 5), 142–143, 180

Measurement tools, speed

- JavaScript debuggers, 285–286
- overview, 277–278
- Page Speed, 283–285
- Speed Tracer, 278–280
- YSlow, 280–282

Memory leaks, 139

Menus, 41–43

Message-Digest algorithm 5 (MD5)

- hashing and, 180
- hashing with JavaScript and, 142–143
- logging in and, 186–190

Messages, dynamic, 217

method, 140

Method parameters, 140–141

Microsoft Bing Maps, 168

MIS (Management Information Systems) applications, 162

Mock objects testing, 239–240, 242–243, 245–246

Model

- caching and, 260–261
- MVP implementation and, 61
- role in MVC, 56–57
- role in MVP, 57–58
- RPC usage and, 100–101

Model-View-Controller (MVC) design pattern, 56–57

Model-View-Presenter (MVP)

- code testing, 238–239
- Composite widgets and, 95
- database-related widgets and, 94–100
- design pattern overview, 57–58

Model-View-Presenter (MVP) implementation

- callbacks and, 59–60
- details, 60–66
- overview, 59

modelMock call, 243–244, 246

-module, compilation option, 288

Modules

- application deployment and, 289–291
- project structure and, 24–25

Montevideo, 157–159

mouseover events, 167

moveMarker(...) method, 172–173

Moving files

- downloading, 204–208
- overview, 195
- summary, 209
- uploading. *See* Uploading files

Multithreading, 14

MultiWordSuggestOracle widgets, 108–112

MVC (Model-View-Controller) design pattern, 56–57

MVP. *See* Model-View-Presenter (MVP)

MyMessages interface, 217–219

N

nameBlurCallback attribute, 68

Net tab, 285–286

Network mode, 278–279

new(...) syntax, 151

newCityList function, 131–132

NewsFeed object

- JSON and, 151–152
- JSONP and, 155

NewsReaderDisplay interface, 149–150

NewsReaderPresenter function, 148–150

NewsReaderView files, 147–149

Nixie display widgets, 143–144

NixieDisplay class, 144–146

Non-repudiation, 178

Nonce

- changing passwords and, 192–193
- encryption and, 183
- logging in and, 185–190

<none>...</none> construct, 45

<noscript> tag, 53

-noserver parameter, 120

NoSuchAlgorithmException(...) function, 180

NumberFormat function, 226

O

onAttach(...) method, 171

onFailure(...) method

- callbacks and, 59
- code splitting and, 291–293

`onModuleLoad(...)` method, 290

`onSuccess(...)` method

 bundling data and, 277

 callbacks and, 59

 code splitting and, 291–293

 interactive maps and, 175

`onVisualizationLoadCallback(...)`
method, 164

OOPHM (Out Of Process Hosted Mode), 27

Open Laszlo, 9

openSUSE, 297

Opera, DragonFly debugger, 285–286

Optimizations, code, 12–13

Optimizing, for application speed

 design patterns for. *See* Design patterns,
 for speed

 measurement tools for. *See* Speed
 measurement tools

 overview, 259

 summary, 286

Options class, 165–166

OPTIONS request, 135

Out Of Process Hosted Mode (OOPHM), 27

OutputStream request, 207

outputStyle, compilation and, 288

Overlays

 getting at feed data with, 161

 JSON usage and, 151–152

P

Page Speed, 283–285

PanelPopup object, 37–38

Pando, 113

Panels

 bundling data and, 275

 displaying forms in pop-up, 37–38

 UI library and, 17

`panToLatLon(...)` method, 172–173

Parameters, 38–41

`parse(...)` method, 227

`parseStrict(...)` method, 226

passwordBlurCallback attribute, 68

Passwords

 changing, security and, 190–193

 logging in and, 185–190

Pattern codes, 225–227

Performance tab, 283–284

Permutation report, 295

Perpetual beta, 7–8

PieChart objects

 dashboard visualizations and, 167–168

 Google Visualization API and, 164

**PieChart.Options specifications,
165–166**

Placeholders, 222

@PluralCount annotation, 218

Pop-up panels, 37–38

Port(s)

 changes, SOP restriction and, 119–121

 processing XML using Ajax and, 128

POST methods

 file producing servlet and, 208

 processing XML using Ajax and,
 127–128, 130

 SOP restriction and, 121

`postToUrl(...)` method, 129

Prefetching, 263–266

Presenter

 changing passwords and, 191–193

 city update application and, 123

 Country/State cities browser and, 102,
 105, 107–108

 data prevalidation and, 114–115

 database-related widgets and, 97–98

- EasyMock testing and, 240–247
- enabling/disabling Login button and, 67–69
- file download form and, 206
- interactive maps and, 169
- live suggestions and, 108, 110–111
- MVP implementation and, 60–66
- receiving/processing XML and, 126
- role in MVP, 57–58
- thread simulation and, 267, 269
- uploading files and, 199–200

PresenterDisplay interface, 62–63

Pretty code, 153

Prevalidation, data, 112–116

Primitive types, 79

Printable View, 282

Processing XML. See Receiving/processing XML

processWeather(...) method, 162

Progressive enhancement, 10–11

Project creation

- with Google Plugin for Eclipse, 21–22
- with GWT shell script, 22–23
- overview, 21

Project structure, 23–27

Projects and development, understanding

- Development mode, running application, 27–30
- overview, 21
- project creation, 21–23
- project structure, 23–27
- summary, 30

.properties files, 212–213, 215

Properties, presetting widget, 73

Protocol changes, SOP restriction and, 119–120

Prototype JavaScript Framework, 5, 11

Proxy

- getting weather feed with, 159
- `RemoteServlet` as, 129–131
- sending XML via, 136–137

pStart+pCount position, 264–265

<public> element, 25–26

public static void deserialize (...), 81

public static void serialize (...), 81

Pyjamas project, 2, 9

Q–R

RC4 encryption, 180–183

readString(...) method, 82

Receiving/processing XML

- overview, 125–127
- through proxy, 129–131
- using Ajax, 127–128

Remote Procedure Calls (RPC), introduction

- Direct Evaluation RPC, 83–84
- implementation, 78–79
- overview, 77
- serialization, 79–83

Remote Procedure Calls (RPC), usage

- code sharing, 86–88
- coding server side services, 88–94
- Country/State cities browser, 101–108
- data prevalidation, 112–116
- database-related widgets, MVP and, 94–100
- deployment and, 300
- Enterprise Java Beans, 116–118
- `GWTTestCase` testing and, 252–253
- live suggestions, 108–112
- looking at Model class, 100–101
- overview, 84

Remote Procedure Calls (RPC), usage*(continued)*

summary, 118

world cities service, 84–85

@RemoteServiceRelativePath(...)
annotation, 78**RemoteServlet function**

Enterprise Java Beans and, 116–118

as proxy, 129–131

RPC implementation and, 79

removeWhitespace(...) method, 126**rename-to** attribute, 25**replay(...)** method, 246**Report link**, 295–296**Representational State Transfer (REST) API**,
173–175**RequestBuilder class**, 203–204**requestSuggestions(...)** method,
110–112**Resource bundles**

annotations and, 215

internationalization and, 212–213

localization and, 224–227

UiBinder-based internationalization
and, 219–223

using constants and, 213–214

ResourceCallback<TextResource>
object, 277**Resources tab**, 284–285**ResultSet** object, 147**RIAs**. See **Rich Internet Applications (RIAs)****Rich Internet Applications (RIAs)**

Cloud Computing, 3

desktop death, 4

overview, 1–2

Web 2.0, 2–3

RPC. See **Remote Procedure Calls (RPC)**,
introduction; **Remote Procedure Calls**
(RPC), usage**RpcResponse** objects, 81**RSS weather feeds**, 157–159**run(...)** method, 267–270**RunAsyncCallback(...)** interface,
291–293**Running applications**

Development mode for, 27–30

getting started, 19–20

Ryan, Ray, 32

S

SAAS (Software as a Service), 3**Safari debugger**, 285**Same Origin Policy (SOP) restriction**

JSONP and, 153–155

server communication and, 119–121

Sampler application, **ClientBundle**, 276**SayAge(...)** string, 218**schedule(...)** method, 266–269**<script>** element

hashing with JavaScript and, 142

project structure and, 26

Scrum, 229**Searching**

live suggestions and, 108–112

with simple news reader, 147–149

weather vane, 157–159

Yahoo's services for, 146–147

Secure Sockets Layer (SSL) communica-
tions, 178**Security**

GWT 2 and, 11

hashing for, 142–143

SOP restriction for, 119–121

Security, servers and

AAA for, 178

Ajax problems, 179

- common operations and. *See* Common operations, security
 - cryptography, 179–183
 - full SSL security and, 177–178
 - overview, 177
 - stateless vs. stateful coding, 183–184
 - summary, 193
- select events, 167**
- Selenium, acceptance testing and**
 - example of, 255–256
 - overview, 253–255
 - potential problems, 257
- Sending XML**
 - overview, 131–132, 135–136
 - through Ajax, 136
 - through proxy, 136–137
- Serialization, RPC, 79–83**
- serialize(...) method, 82**
- Server, communication with**
 - introduction to RPC. *See* Remote Procedure Calls (RPC), introduction
 - RPC patterns of usage. *See* Remote Procedure Calls (RPC), usage
- Server side services, 88–94**
- ServerCityData classes, 86–88, 90**
- Servers, communication with other**
 - city update application, 121–125
 - overview, 119
 - producing XML, 131–135
 - receiving/processing XML, 125–131
 - sending XML, 135–137
 - SOP restriction and, 119–121
 - summary, 137
- Servers, working with**
 - challenges in, 177–183
 - common operations and. *See* Common operations, security
 - cryptography, 179–183
 - overview, 177
 - stateless vs. stateful coding, 183–184
 - summary, 193
- Service-Oriented Architectures (SOA), 3**
- <servlet> element**
 - client-plus-server GWT 2 and, 299–300
 - file processing and, 200
 - RPC implementation and, 78
- Servlet mapping, 78**
- <servlet-mapping> element, 78**
- Servlet(s)**
 - calling remote, 79
 - deployment and, 297–300
 - file download form, 204–207
 - file processing, 200–202
 - file producing, 207–208
 - GWTTestCase testing, 252–253
- Session keys**
 - changing passwords and, 192–193
 - logging in and, 185–190
- setAttribute(...) method, 134–135**
- set.Callback(...) method, 123**
- setCitiesOracle(...) method, 109, 111**
- setCityData(...) method**
 - city update application and, 123
 - Country/State cities browser and, 102, 104, 106–107
 - receiving/processing XML and, 127
- setCoordinates(...) method, 171–173**
- setCountryList(...) function, 97–98**
- setNameBlurCallback(...) method, 244–245**
- setStateList(...) function, 97–98**

setText(...) method, 145–146

Setting up GWT

overview, 17

running and deploying, 19–20

version control management/testing,
19, 20

writing code, 17–18

setUp(...) methods, 233

setYGeoPoint(...) method, 172–173

Shell script, webAppCreator, 22–23

show(...) method, 37–38

signature parameters, 140

Simple city browser application, 101

Selenium testing and, 255

thread simulation and, 266–270

Simple city input form, 112–113

Simple news reader, 147–148

Sluggishness report, 278–279

Smush.it, 282

SOA (Service-Oriented Architectures), 3

Software as a Service (SAAS), 3

Software methodologies

Agile Software Methodologies, 7

classic development problems, 5–7

perpetual beta, 7–8

somefile.txt text file, 208

someModules, compilation and, 288

someMore variable, 268

someNumber, compilation and, 288, 289

SOP. See Same Origin Policy (SOP) restriction

Soriano, 124, 252, 255

@Source(...) annotation, 274–275

<source> element, 23, 25–26

-soyc parameter, 221

Speed, design patterns for. See Design patterns, for speed

Speed measurement tools

JavaScript debuggers, 285–286

overview, 277–278

Page Speed, 283–285

Speed Tracer, 278–280

YSlow, 280–282

Speed Tracer, 278–280

src directory

JUnit test directory and, 231

modules directory and, 290

project structure and, 23, 24

SSL (Secure Sockets Layer) communications, 178

Starting, GWT application, 34–37

Stateless server coding vs. stateful, 183–184

Statement Coverage, 236

statesCache function, 261

static object

bundling data and, 274

caching and, 260–262

Statistics tabs, 282

Steampunk display widgets, 143–145

Stooges. See Three Stooges, The

stop(...) method, 143

Streams, reading/writing to, 82

strictfp keyword, 13

String Interning, 13

Strings

creating XML with, 132–133

DOM structure and, 133–135

dynamic messages and, 217–219

localization and, 224–227

resource bundles as, 212–215

sending XML, 135–136

serialization of, 80

weather feed, 159

Stubs, 240

- style, compilation option, 288
- <stylesheet> element, 26
- Submit event code, 203–204
- Subversion, for version control management, 19
- SuggestBox widgets, 108–109, 112
- SuggestionItem class, 110–111
- Super-validationProblems(...)
 - function, 86–88
- SupportsCDATASection(...) method, 126

T

- TDD (Test-driven development), 229
- tearDown(...) methods, 233
- Templates
 - creating several, 75
 - creating UiBinder, 70–72
- test directory, 23
- Test-driven development (TDD), 229
- @Test methods, 233–236
- Testing
 - applications. *See* Application testing
 - getting started, 19, 20
 - test/gwttest directories for, 23
- TextBox function, 147–149, 171
- TextResource elements, 274–277
- Thread simulation
 - deferred command-based solution, 270–273
 - overview, 266
 - Timer-based solution, 266–270
- Three Stooges, The, 237–238
- Time formats, 224–226
- TimedCitiesDisplay class, 269–270
- Timer function
 - live suggestions and, 112
 - thread simulation and, 266–270
- Tokens, 33, 34–37

- Tools tab, 282
- toString(...) methods, 133, 135, 234
- totalResultsAvailable attribute, 147
- transient attributes, 80
- Translating error codes, 215–217
- treeLogger, compilation option, 288

U

- <u:attribute> element, 221
- u:field attribute, 70, 72–73
- UI. *See* User Interface (UI), programming
- UI library, 17
- UI patterns
 - MVC classic pattern, 56–57
 - MVP pattern, 57–58
 - overview, 55–56
- UiBinder
 - changing password and, 190
 - Country/State cities browser and, 102–104
 - data prevalidation and, 113–114
 - dealing with constructors, 74–75
 - internationalization, 219–223
 - Java defined in, 72–73
 - overview, 69
 - presetting widget properties, 73
 - template defined in, 70–72
 - using your own widgets, 73–74
 - working with complex layouts, 75
- @UiField annotation
 - JSON and, 149
 - UiBinder and, 71–73
 - uploading files and, 198
 - widgets and, 74, 97, 103
- @UiHandler annotation
 - Country/State cities browser and, 104–105
 - data prevalidation and, 114

@UiTemplate annotation

- JSON and, 149
- UiBinder and, 71–72, 75
- uploading files and, 198
- widgets and, 96, 103

ui.xml files

- internationalization and, 219–223
- UiBinder and, 70, 72

<u:msg> element, 220–222**Unicode Transformation Format (UTF-8), 213****Unified Modeling Language (UML), 61****Uniform Resource Locator (URL)**

- fixed maps and, 173–175
- JSONP and, 155
- news search service and, 150–153
- receiving/processing XML and, 127–130
- sending XML via Ajax and, 136–137
- for weather search, 158

Upload form, 195–200**Uploading files**

- file processing servlet, 200–202
- overview, 195
- providing feedback, 202–204
- upload form, 195–200

URL.encode(...) method, 136–137**User Interface (UI), programming**

- declarative UI, 69–75
- extensions, 67–69
- MVP implementation. *See* Model-View-Presenter (MVP) implementation
- overview, 55
- summary, 76
- UI patterns, 55–58

UTF-8 (Unicode Transformation Format), 213**Utility methods, 15****<u:Uibinder> element**

- city browser and, 102
- defining templates and, 69, 71, 73
- internationalization and, 220
- JSON and, 148
- widgets and, 96

V
-validateOnly, compilation option, 288**Validation, 86–88****ValueChangeHandler method**

- data prevalidation and, 114
- database-related widgets and, 95

Version control management, 19**VerticalPanel function, 170****View**

- changing password and, 190–191
- Country/State cities browser and, 102–104, 106–108
- data prevalidation and, 114
- database-related widgets and, 98, 100
- EasyMock testing and, 242–246
- FileUpload, 195–198
- GWTTestCase and, 247–251
- interactive maps and, 170
- live suggestions and, 108–109
- MVP implementation and, 60–61
- role in MVC, 56–57
- role in MVP, 57–58
- simple news reader, 147–148

Visualization options, 165, 168**Visualizations, dashboard**

- Google Visualization API, 164–167
- handling events, 167–168
- overview, 162–164

VisualizationUtils package, 164

W

W3C "Access Control Specification," 121

`waitFor`. **commands,** 257

`-war`, **compilation option,** 289

`war` **directory,** 297

`war` **folder,** 24

`wasCalled` **variable,** 242–246

Waterfall Model, for development process, 5–7

The Weather Channel, 158–159

Weather vane

- getting at feed data with overlays, 161
- getting everything together, 160–161
- getting feed, 159–160
- getting feed with JSNI, 162
- getting weather data, 157–159
- overview, 157

WeatherFeed data, 161

Web 2.0, 2–3

"Web as Platform" concept, 2

`webAppCreator` **shell script,** 22–23

`webAppGenerator`, 53

`web.xml` **file,** 298

Where On Earth ID (WOEID) code, 158

Widgets

- `Composite`, 95, 101–108, 171
- `FileUpload`, 196–198
- `HTML`, 46–47
- `Hyperlink`, 43
- interactive maps and, 170–171
- `ListBox`, 93, 94–99
- `MultiWordSuggestOracle`, 111
- MVP and database-related, 94–100
- presetting properties of, 73
- Steampunk display, 143–144
- `SuggestBox`, 108–109, 112

- UI library and, 17
- using your own, 73–74
- weather vane, 157–162

`Window.alert(...)` **message,** 13, 239

`$wnd`

- getting feed data and, 162
- hashing with JavaScript and, 142–143
- interactive maps and, 172–173

`-workDir`, **compilation option,** 289

World cities service, 84–85

`WorldService` **remote service,** 84–85

`WorldService` **remote servlet,** 78

`WorldService.java` **interface,** 88–89

`writeString(...)` **method,** 82

X

- `-XdisableAggressiveOptimization`, **compilation option,** 289
- `-XdisableCastChecking`, **compilation option,** 289
- `-XdisableClassMetadata`, **compilation option,** 289
- `-XdisableRunAsync`, **compilation option,** 289
- `XhrProxy` **servlet,** 129–131, 175
- XML.** See Extensible Markup Language (XML)
- `XMLHttpRequest` **method,** 136
- `XMLParser.parse(...)` **method,** 125–127
- XP (Extreme Programming),** 229

Y–Z

Yahoo!

- Maps, 168–175
- news search using, 146–147
- Weather RSS Feed, 157–159
- Yahoo Pipes, 159

322 yahooMap attribute

yahooMap attribute, 172

YGeoPoint object, 172-173

YMap function, 172

yourownfunction(...) method, 154

YSlow, 280-282

YUI Library, 5