

# *GWT in Action*

**SECOND EDITION**

ADAM TACY  
ROBERT HANSON  
JASON ESSINGTON  
ANNA TÖKKE



**MANNING**  
**SHELTER ISLAND**

# contents

---

*preface* xxi  
*acknowledgments* xxiv  
*about this book* xxvi  
*about the cover illustration* xxxiv

## PART 1 BASICS ..... 1

### **1** GWT 3

1.1 Unplanned consequences (or the road to GWT) 4

1.2 Exploring the toolkit 7

*Compiling and optimizing* 7 ▪ *Powerful widgets and a  
template binding engine* 8 ▪ *Event handling beyond  
JavaScript* 10 ▪ *Client/server communication* 11  
*Simplified development and debugging* 12 ▪ *Integration  
with JavaScript* 13 ▪ *History support* 14  
*Internationalization—Sprechen sie Deutsch?* 15

1.3 Setting up your development environment 16

*Installing the JDK* 18 ▪ *Installing Eclipse* 18 ▪ *Installing  
the Google Plugin for Eclipse* 19 ▪ *Installing the Development  
Mode Browser Plugin* 22

1.4 Summary 22

<b>2</b>	<b><i>Building a GWT application: saying “Hello World!”</i></b>	<b>24</b>
2.1	What’s a GWT application?	26
	<i>Seeing the user’s view</i>	26
	▪ <i>Examining the developer’s view</i>	27
	▪ <i>Understanding development vs. web mode</i>	28
2.2	Examining the options for building an application	31
2.3	Creating the HelloWorld application with the GPE	32
	<i>Creating a web application</i>	33
	▪ <i>Defining a GWT module</i>	35
	<i>Adding an entry point</i>	36
	▪ <i>Providing the web page</i>	38
	<i>Recapping the magic</i>	39
2.4	Running HelloWorld in development mode	40
	<i>Starting development mode in Eclipse</i>	41
	▪ <i>Passing parameters to development mode</i>	42
2.5	Finding out where it went wrong	45
	<i>Checking the code in the IDE for errors</i>	45
	▪ <i>Looking at development mode output</i>	46
	▪ <i>Reading the console output</i>	46
	▪ <i>Debugging in Eclipse</i>	47
	▪ <i>Inspecting using browser development/inspection tools</i>	48
2.6	Compiling HelloWorld for web mode	49
	<i>Running the GWT compiler from Eclipse</i>	49
	<i>Welcoming the user</i>	52
	▪ <i>Passing parameters to the GWT compiler</i>	52
2.7	Understanding modules vs. packages	54
	<i>What’s in a GWT module?</i>	55
	▪ <i>What are the benefits of modules?</i>	57
	▪ <i>How many modules should you have?</i>	58
2.8	Digging deeper into the uncompiled application	60
	<i>Folder structure convention</i>	60
	▪ <i>Package structure convention</i>	60
	▪ <i>What parts of Java can you use in GWT?</i>	61
	▪ <i>The server side</i>	62
2.9	Reviewing the deployable application part of a GWT application	62
	<i>Harnessing different linkers</i>	63
2.10	Building on your understanding	64
2.11	Summary	65
<b>3</b>	<b><i>Building a GWT application: enhancing HelloWorld</i></b>	<b>67</b>
3.1	Reexamining the example application	68
	<i>Enhancements</i>	69

- 3.2 Updating the HTML 70
- 3.3 Enhancing the code 72
- 3.4 Creating your user interface 73
  - Presenting widgets* 74 ▪ *Organizing layout with Panels* 77
- 3.5 Manipulating the page 82
  - Using the RootPanel/RootLayoutPanel* 82 ▪ *Manipulating the DOM directly* 83
- 3.6 Handling events 85
  - What are events?* 85 ▪ *Handling events* 85 ▪ *Preventing the browser from handling events for you* 87
- 3.7 Managing history 89
  - Handling history in GWT* 89 ▪ *Implementing history management in your application* 90
- 3.8 Styling components 92
  - Programmatic styling* 93 ▪ *Low-level styling* 93
  - Cascading Style Sheets* 94 ▪ *GWT themes* 95
- 3.9 Securing your application 96
- 3.10 Building on your understanding 98
- 3.11 Summary 99

## PART 2 NEXT STEPS ..... 101

### 4 *Creating your own widgets* 103

- 4.1 What is a widget, again? 104
- 4.2 Creating a new widget from the DOM 106
  - Introducing the GWTiACanvas widget* 107 ▪ *Indicating functionality* 109 ▪ *Hooking up events* 110
  - Getting secure by using SafeHTML, SafeUri, and SafeStyles* 112
- 4.3 Extending an existing widget 114
  - Introducing the ReportSizeLabel widget* 114 ▪ *Indicating functionality* 115
- 4.4 Extending a panel 117
- 4.5 Creating a composite 118
  - Introducing the DataField question/answer widget* 119
  - Indicating functionality* 121

- 4.6 Using layout panels 124
  - Types of layout panels* 125 ▪ *Creating layout panels* 126
  - Animating layout panels* 129
- 4.7 Applying animation to widgets 130
  - Widgets that animate* 130 ▪ *Building your own animation* 131
- 4.8 Exploring the lifecycle of a widget 132
  - Creating a widget* 132 ▪ *Adding a widget* 133
  - Removing a widget* 135 ▪ *Destroying a widget* 136
- 4.9 Getting Elemental, my dear Watson! 137
  - Examining Elemental* 137 ▪ *Understanding the challenge* 138 ▪ *Noting the benefit* 138
- 4.10 Summary 139

## 5 Using client bundles 140

- 5.1 Client bundle basics using DataResources 141
  - DataResource* 142 ▪ *A simple ClientBundle* 143
  - Creating ClientBundles using the Google Plugin for Eclipse* 146 ▪ *Using ClientBundles in an application* 147
- 5.2 Text resource types 149
  - TextResource* 149 ▪ *ExternalTextResource* 150
- 5.3 ImageResource 152
  - Internationalizing image resources* 153 ▪ *Using ImageResource in an application* 153 ▪ *Controlling ImageResource optimizations* 155
- 5.4 CssResource 157
  - Optimizations* 158 ▪ *Constants* 160 ▪ *Runtime evaluation* 162 ▪ *Nonstandard CSS values* 163
  - Conditional sections* 164 ▪ *Using other resources in CSS* 165
- 5.5 Summary 166

## 6 Interface design with UiBinder 168

- 6.1 Binding the designer's HTML to Java code 169
  - Creating the UiBinder XML template from HTML* 171
  - Working with panels* 175 ▪ *Binding the UiBinder XML template to the Java code* 177 ▪ *Binding XML template elements to Java variables* 178 ▪ *Making sense of it all* 181

- 6.2 Handling events with UiBinder 183
- 6.3 Introducing the UiBinder expression language 184
- 6.4 Applying style with UiBinder 188
  - Using <ui:style> to generate a CssResource 188* ■ *Accessing a generated CssResource in your widget 190*
- 6.5 Using the Eclipse plug-in with UiBinder 194
- 6.6 Summary 195

## **7 Communicating with GWT-RPC 196**

- 7.1 Surveying GWT-RPC 197
  - Understanding asynchronous behavior 198* ■ *Defining the GWT-RPC classes, interfaces, and annotations 198*
  - Understanding GWT-RPC package structure 201*
- 7.2 Learning GWT-RPC with Twitter 202
- 7.3 Fetching data from Twitter the non-GWT way 203
- 7.4 Defining a GWT-RPC-compatible model 204
  - Using the Serializable and IsSerializable interfaces 205*
  - Special considerations when using JPA/JDO model objects as DTOs 206* ■ *Developing custom serializers 207*
- 7.5 Building and deploying the server side 210
  - Handling exceptions 211* ■ *Defining the service interface 211* ■ *Writing the servlet 213* ■ *Deploying the servlet 215*
- 7.6 Writing the client 217
  - Defining the asynchronous interface 217* ■ *Making the call to the server 218*
- 7.7 Debugging GWT-RPC 222
- 7.8 Securing GWT-RPC against XSRF attacks 223
  - Understanding XSRF attacks 223* ■ *Adding XSRF protection to your RPC calls 224*
- 7.9 Summary 229

## **8 Using RequestFactory 231**

- 8.1 Enabling annotation processing 233
  - Enabling RequestFactory annotation processing with javac 233* ■ *Enabling RequestFactory annotation processing in Eclipse 234* ■ *Enabling RequestFactory annotation processing in Maven 235*

- 8.2 Understanding RequestFactory architecture 236
    - Investigating the client-side architecture* 236
    - *Investigating the server-side architecture* 237
  - 8.3 Understanding the example project in this chapter 239
    - Enabling RequestFactory the simple way* 240
    - *Creating proxy interfaces for the domain classes* 240
    - *Developing the factory interface* 243
    - *Using the domain class as the service* 245
    - Adding the RequestFactory servlet to the web.xml* 248
  - 8.4 Making calls to the server 249
    - Initializing RequestFactory and making a simple call to the server* 249
    - *Creating and persisting using instance methods* 251
    - *Fetching persisted objects from the server* 253
    - *Editing domain objects and updating them on the server* 255
    - *Error handling and validation* 257
  - 8.5 Using custom Locators and ServiceLocators (the “long way”) 262
    - Creating a custom Locator* 262
    - *Creating a custom ServiceLocator* 264
  - 8.6 Summary 268
- ## 9 The Editor framework 269
- 9.1 Framework and editor overview 270
    - Local domain object* 272
    - *Remote domain objects* 272
  - 9.2 Examining the chapter’s examples 273
  - 9.3 Editor types 275
  - 9.4 Constructing your first editor 276
    - Defining the local domain object* 277
    - *Defining the editor* 278
  - 9.5 Binding an editor with drivers 283
    - EmployeeEditor with SimpleBeanEditorDriver* 285
    - EmployeeEditor with RequestFactoryEditorDriver* 287
  - 9.6 Editor subinterfaces 290
    - Accessing the backing framework services* 291
    - *Editors with error handling* 292
    - *Editing immutable objects or read-only editors* 293
    - *Building customized editor behavior* 295
    - *Handling subeditors of the same type* 296
  - 9.7 Accessing the RequestContext 299

- 9.8 Alternate way to construct an editor 300
- 9.9 Adapters 302
  - Editing a range of domain objects* 302
  - Adapting a list of objects with associated editors* 304
  - Adapters for single-domain objects* 306
- 9.10 Summary 308

## 10 *Data-presentation (cell) widgets* 309

- 10.1 Understanding cells 310
  - Looking at display cells* 312
  - Updating edit cells* 314
  - Reacting with action cells* 315
- 10.2 Creating custom cells 317
  - Composite* 317
  - From first principles* 319
- 10.3 Reviewing GWT's cell widgets 327
- 10.4 Looking at a CellList 329
  - Creating a CellList* 329
  - Populating data* 330
  - Paging* 332
  - Handling user updates* 333
  - Managing data selection with SelectionModels* 334
  - Managing the keyboard* 335
- 10.5 Walking through a CellTree 336
  - Opening a new node (with an asynchronous data provider)* 337
  - Determining if you're in the leaves* 340
- 10.6 Browsing a CellBrowser 340
- 10.7 Constructing a CellTable 341
  - Creating a table* 342
  - Applying headers and footers* 343
  - Sorting the view* 344
- 10.8 Building a DataGrid 346
  - Custom CellTable building* 347
- 10.9 Summary 350

## 11 *Using JSNI—JavaScript Native Interface* 352

- 11.1 What is JSNI? 354
- 11.2 Should you use JavaScript Native Interface? 355
  - No, JSNI can quickly limit the benefits of using GWT* 355
  - Yes, in these circumstances* 356
- 11.3 Benefiting from the Google Plugin for Eclipse 357



- 11.4 Interacting with the browser 359
  - Example: getting a browser element's computed style* 359
  - *Passing data in to a JSNI method* 362
  - Passing data out of a JSNI method* 363
- 11.5 Handling objects from JavaScript 365
  - Example: using a JavaScriptObject* 366
  - *Example: extending a JavaScriptObject (an overlay)* 367
  - *Example: overlaying JSONP data* 370
- 11.6 Wrapping a third-party library 372
  - Ensuring the library is loaded* 373
  - *Accessing Java fields from JSNI* 377
  - *Calling Java methods from JSNI* 380
  - Creating Java objects within JSNI* 383
  - *Handling exceptions* 384
- 11.7 Exposing an API to JavaScript 384
- 11.8 Summary 386

## 12 *Classic Ajax and HTML forms* 387

- 12.1 Understanding the underlying technology 388
  - Understanding how HTTP works* 388
  - *Understanding Ajax and the XMLHttpRequest object* 391
  - *Understanding JSON* 392
  - *Solving same-site-origin policy issues with JSONP* 393
- 12.2 Using RequestBuilder 394
- 12.3 Posting data with RequestBuilder 396
- 12.4 Using the JSON API and JsonRequestBuilder 399
- 12.5 Using JSON with JS overlay 402
- 12.6 Using the XML API and RequestBuilder 404
  - Developing a server-side proxy* 404
  - *Calling the proxy from GWT* 406
  - *Parsing XML content* 409
- 12.7 Using FormPanel 411
  - Designing a FormPanel registration form with UiBinder* 412
  - Adding behavior to the FormPanel* 413
- 12.8 Summary 416

## 13 *Internationalization, localization, and accessibility* 417

- 13.1 Making a user feel comfortable 418
  - What is a locale?* 419
  - *Setting up to use internationalization* 420
  - *The three types of GWT internationalization* 421

- 13.2 Using static string internationalization 422
  - The basics* 424 ▪ *The Localizable interface* 426
  - Localizable annotations* 427 ▪ *Internationalizing constants* 428 ▪ *Constants with lookup* 430
  - Messaging the user* 430 ▪ *Dealing with plurals* 433
  - Selecting an alternate message based on a user-defined value* 434 ▪ *Securing against hack attacks* 435
- 13.3 Using static-string i18n with UiBinder 436
  - Constants with UiBinder* 437 ▪ *Parameterized messages with UiBinder* 439
- 13.4 Determining the locale for static-string internationalization 440
  - Where to find the locale* 441 ▪ *Searching the URL* 442
  - Digesting a cookie* 443 ▪ *Finding a HTML meta tag* 444 ▪ *Letting the browser decide* 444
- 13.5 Internationalizing client bundles through static internationalization 445
- 13.6 Dynamic string internationalization 445
  - The basics* 446 ▪ *Enhancing the standard approach* 447
  - Using with UiBinder* 448
- 13.7 Localization of dates, times, and currencies 448
  - Displaying numbers and currency values* 449 ▪ *Displaying times and dates* 449
- 13.8 Displaying the right direction 450
- 13.9 Accessibility 452
  - Using alternative text for images* 452 ▪ *Setting up a tab index* 452 ▪ *Establishing keyboard shortcuts* 453
  - Providing alternative styling* 453 ▪ *Using ARIA* 453
- 13.10 Summary 455

## PART3    **ADVANCED** ..... 457

### **14** *Advanced event handling and event busses* 459

- 14.1 Understanding events 460
  - Native events* 461 ▪ *Logical events* 462
- 14.2 How GWT manages events 463
  - Dealing with browser differences* 463 ▪ *Preventing event propagation* 466 ▪ *Sinking events* 466 ▪ *Event-handling efficiency* 469

- 14.3 Previewing and canceling events 470
- 14.4 Preventing default actions 472
- 14.5 Programmatically firing events 473
- 14.6 Creating your own events 474
  - Defining your own event* 475 ▪ *Providing the related interfaces* 476
- 14.7 Event busses 477
  - What is an event bus?* 478 ▪ *Types of event busses* 479
  - Using SimpleEventBus* 480
- 14.8 Summary 482

## 15 *Building MVP-based applications* 483

- 15.1 What is MVP? 484
  - The two-way presenter/view relationship* 485 ▪ *Benefits of MVP* 487
- 15.2 Looking at the PhotoApp's MVP foundations 487
  - From the user's perspective* 487 ▪ *From the MVP perspective* 488
- 15.3 Building MVP yourself 490
  - Creating views* 491 ▪ *Presenters* 493 ▪ *Controlling the application* 495
- 15.4 Altering an MVP application 497
  - Swapping out layers* 497 ▪ *Optimizing with code splitting* 499
- 15.5 Activity and Place (GWT's reference MVP approach) 500
  - How objects plug together* 501 ▪ *Activity* 503
  - Places* 508 ▪ *Place tokenizers* 509
  - PlaceHistoryMapper* 510 ▪ *ActivityMapper* 511
  - Managing the activities* 512 ▪ *Controlling the place* 513 ▪ *Views* 513 ▪ *Code splitting with activities and places* 514
- 15.6 Fitting editors/data-presentation widgets into MVP 514
- 15.7 Summary 515

## 16 *Dependency injection* 516

- 16.1 Dependency injection—the fundamentals 518
  - At the beginning of time* 519 ▪ *Straight from the factory* 520
  - Automatically injecting dependencies* 521

- 16.2 Guice—a Java dependency injection framework 522
  - Defining the dependencies* 523
  - *Types of injection* 524
- 16.3 GIN—how DI differs in a GWT application 527
  - Setting up for GIN* 527
  - *Defining the dependencies* 528
  - Bootstrapping the injection* 530
  - *Types of injection* 531
  - Swapping components* 534
- 16.4 When to avoid DI 535
- 16.5 Summary 536

## 17 *Deferred binding* 538

- 17.1 What is deferred binding? 540
  - Storing implementation differences in a Java class hierarchy* 541
  - *Identifying differences via deferred-binding properties* 544
  - *Informing the GWT compiler which class to pick* 544
  - *Telling the GWT compiler to make a choice* 547
  - *Selecting the right difference (permutation) at runtime* 547
- 17.2 Pulling it all together 548
- 17.3 Using GWT properties to drive deferred binding 549
  - *Defining properties* 550
  - *Extending properties* 551
  - Setting properties* 551
  - *Conditionally setting a property* 553
- 17.4 Managing explosive permutation numbers 554
  - Using conditional properties* 555
  - *Using soft permutations* 555
- 17.5 Determining a property value 557
  - Directly setting a property value in a module file* 557
  - Understanding property providers* 558
  - *Generating a property provider* 559
  - *Defining your own property provider* 560
  - *Handling failure to get a property value* 562
- 17.6 Coping when deferred binding isn't enough 563
- 17.7 Summary 565

## 18 *Generators* 566

- 18.1 What does a generator do? 567
- 18.2 What can a generator do? 569
  - Accessing code* 570
  - *Reading annotations* 570
  - Accessing properties* 571
  - *Using resources* 571
  - Manipulating resources* 572

- 18.3 Indicating what generator to use and when 573
- 18.4 Configuration properties 574
  - Defining a configuration property* 574
  - Setting the value of a configuration property* 575
  - Extending the value of a configuration property* 575
- 18.5 Pulling it all together 576
- 18.6 Preparing to write a generator 576
- 18.7 Creating your own generator 578
  - The generator skeleton* 579
  - Creating a new type* 580
  - Writing the new content* 582
  - Accessing types through the TypeOracle* 584
  - Accessing properties through the PropertyOracle* 586
  - Accessing resources through the ResourceOracle* 587
  - Logging in the generator* 589
- 18.8 Using your new generator 589
- 18.9 Summary 590

## 19 *Metrics and code splitting* 591

- 19.1 Using the lightweight metrics tool 592
  - Defining lightweight metrics* 593
  - Writing the global collector* 593
  - Sending events to the global collector* 599
- 19.2 Using the Compile Report 602
  - Turning on the Compile Report* 602
  - Understanding the permutation list* 604
  - Digging into the Split Point report* 605
- 19.3 Making use of code splitting 611
  - Understanding code-splitting basics* 611
  - Using the Async Package pattern* 614
  - Reducing leftover code by specifying load order* 619
- 19.4 Summary 620

*index* 623