The Java EE 7 Tutorial

Volume 1

Fifth Edition

Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans,
Kim Haase, William Markito

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

The Java EE 7 Tutorial

Volume 1

Fifth Edition

This page intentionally left blank

The Java EE 7 Tutorial

Volume 1

Fifth Edition

Eric Jendrock
Ricardo Cervera-Navarro
Ian Evans
Kim Haase
William Markito

Addison-Wesley

Upper Saddle River, NJ ● Boston ● Indianapolis ● San Francisco
New York ● Toronto ● Montreal ● London ● Munich ● Paris ● Madrid
Capetown ● Sydney ● Tokyo ● Singapore ● Mexico City

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.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

The authors and publishers 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.

This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or at (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the United States, please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress control Number: 2014933972

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.
500 Oracle Parkway, Redwood Shores, CA 94065

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. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-321-99492-9
ISBN-10: 0-321-99492-2
Text printed in the United States on recycled paper at Edwards Brothers Malloy in Ann Arbor, Michigan.
First printing, May 2014

Contents

Preface xxix

Audience xxx
Before You Read This Book xxx
Related Documentation xxx
The Oracle Accessibility Program xxxi
Conventions xxxi
Default Paths and File Names xxxi
Acknowledgments xxxii

Part I Introduction 1

1 Overview 3

1.1 Java EE 7 Platform Highlights 4
1.2 Java EE Application Model 5
1.3 Distributed Multitiered Applications 6
1.3.1 Security 7
1.3.2 Java EE Components 8
1.3.3 Java EE Clients 8
1.3.3.1 Web Clients 8
1.3.3.2 Application Clients 8
1.3.3.3 Applets 9
1.3.3.4 The JavaBeans Component Architecture 9
1.3.3.5 Java EE Server Communications 10
1.3.4 Web Components 10
1.3.5 Business Components 11

1.3.6 Enterprise Information System Tier 12
1.4 Java EE Containers 13
1.4.1 Container Services 13
1.4.2 Container Types 14
1.5 Web Services Support 15
1.5.1 XML 16
1.5.2 SOAP Transport Protocol 16
1.5.3 WSDL Standard Format 16
1.6 Java EE Application Assembly and Deployment 17
1.7 Development Roles 17
1.7.1 Java EE Product Provider 18
1.7.2 Tool Provider 18
1.7.3 Application Component Provider 18
1.7.3.1 Enterprise Bean Developer 18
1.7.3.2 Web Component Developer 18
1.7.3.3 Application Client Developer 19
1.7.4 Application Assembler 19
1.7.5 Application Deployer and Administrator 19
1.8 Java EE 7 APIs 20
1.8.1 Enterprise JavaBeans Technology 23
1.8.2 Java Servlet Technology 24
1.8.3 JavaServer Faces Technology 24
1.8.4 JavaServer Pages Technology 25
1.8.5 JavaServer Pages Standard Tag Library 25
1.8.6 Java Persistence API 26
1.8.7 Java Transaction API 26
1.8.8 Java API for RESTful Web Services 26
1.8.9 Managed Beans 26
1.8.10 Contexts and Dependency Injection for Java EE 27
1.8.11 Dependency Injection for Java 27
1.8.12 Bean Validation 27
1.8.13 Java Message Service API 27
1.8.14 Java EE Connector Architecture 28
1.8.15 JavaMail API 28
1.8.16 Java Authorization Contract for Containers 29
1.8.17 Java Authentication Service Provider Interface for Containers 29
1.8.18 Java API for WebSocket 29

1.8.19 Java API for JSON Processing 29
1.8.20 Concurrency Utilities for Java EE 30
1.8.21 Batch Applications for the Java Platform 30
1.9 Java EE 7 APIs in the Java Platform, Standard Edition 7 30
1.9.1 Java Database Connectivity API 30
1.9.2 Java Naming and Directory Interface API 31
1.9.3 JavaBeans Activation Framework 31
1.9.4 Java API for XML Processing 32
1.9.5 Java Architecture for XML Binding 32
1.9.6 Java API for XML Web Services 32
1.9.7 SOAP with Attachments API for Java 33
1.9.8 Java Authentication and Authorization Service 33
1.9.9 Common Annotations for the Java Platform 33
1.10 GlassFish Server Tools 33

2 Using the Tutorial Examples 35

2.1 Required Software 35
2.1.1 Java EE 7 Software Development Kit 36
2.1.1.1 SDK Installation Tips 36
2.1.2 Java Platform, Standard Edition 37
2.1.3 Java EE 7 Tutorial Component 37
2.1.3.1 To Obtain the Tutorial Component Using the Update Tool 37
2.1.4 NetBeans IDE 37
2.1.4.1 To Install NetBeans IDE without GlassFish Server 38
2.1.4.2 To Add GlassFish Server as a Server Using NetBeans IDE 38
2.1.5 Apache Maven 38
2.2 Starting and Stopping GlassFish Server 39
2.2.1 To Start GlassFish Server Using NetBeans IDE 39
2.2.2 To Stop GlassFish Server Using NetBeans IDE 39
2.2.3 To Start GlassFish Server Using the Command Line 39
2.2.4 To Stop GlassFish Server Using the Command Line 40
2.3 Starting the Administration Console 40
2.3.1 To Start the Administration Console Using NetBeans IDE 40
2.4 Starting and Stopping the Java DB Server 40
2.4.1 To Start the Database Server Using NetBeans IDE 41
2.5 Building the Examples 41
2.6 Tutorial Example Directory Structure 41

2.7 Java EE 7 Maven Archetypes in the Tutorial 42
2.7.1 Installing the Tutorial Archetypes 42
2.7.1.1 Installing the Tutorial Archetypes Using NetBeans IDE 43
2.7.1.2 Installing the Tutorial Archetypes Using Maven 43
2.8 Getting the Latest Updates to the Tutorial 43
2.8.1 To Update the Tutorial Using NetBeans IDE 43
2.8.2 To Update the Tutorial Using the Command Line 43
2.9 Debugging Java EE Applications 44
2.9.1 Using the Server Log 44
2.9.1.1 To Use the Administration Console Log Viewer 44
2.9.2 Using a Debugger 45
2.9.2.1 To Debug an Application Using a Debugger 45

Part II Platform Basics 47

3 Resource Creation 49

3.1 Resources and JNDI Naming 49
3.2 DataSource Objects and Connection Pools 50
3.3 Creating Resources Administratively 51

4 Injection 53

4.1 Resource Injection 53
4.2 Dependency Injection 54
4.3 The Main Differences between Resource Injection and Dependency Injection 55

5 Packaging 57

5.1 Packaging Applications 57
5.2 Packaging Enterprise Beans 59
5.2.1 Packaging Enterprise Beans in EJB JAR Modules 59
5.2.2 Packaging Enterprise Beans in WAR Modules 60
5.3 Packaging Web Archives 61
5.4 Packaging Resource Adapter Archives 63

Part III The Web Tier 65

6 Getting Started with Web Applications 67

6.1 Web Applications 68
6.2 Web Application Lifecycle 69
6.3 A Web Module That Uses JavaServer Faces Technology: The hello1 Example 70
6.3.1 To View the hello1 Web Module Using NetBeans IDE 70
6.3.1.1 Introduction to Scopes 74
6.3.2 Packaging and Deploying the hello1 Web Module 74
6.3.2.1 To Build and Package the hello1 Web Module Using NetBeans IDE 75
6.3.2.2 To Build and Package the hello1 Web Module Using Maven 75
6.3.3 Viewing Deployed Web Modules 75
6.3.3.1 To View Deployed Web Modules Using the Administration Console 75
6.3.3.2 To View Deployed Web Modules Using the asadmin Command 76
6.3.3.3 To View Deployed Web Modules Using NetBeans IDE 76
6.3.4 Running the Deployed hello1 Web Module 76
6.3.4.1 Dynamic Reloading of Deployed Modules 76
6.3.5 Undeploying the hello1 Web Module 77
6.3.5.1 To Undeploy the hello1 Web Module Using NetBeans IDE 77
6.3.5.2 To Undeploy the hello1 Web Module Using the asadmin Command 77
6.4 A Web Module That Uses Java Servlet Technology: The hello2 Example 77
6.4.1 Mapping URLs to Web Components 77
6.4.2 Examining the hello2 Web Module 78
6.4.2.1 To View the hello2 Web Module Using NetBeans IDE 78
6.4.3 Running the hello2 Example 80
6.4.3.1 To Run the hello2 Example Using NetBeans IDE 80
6.4.3.2 To Run the hello2 Example Using Maven 81
6.5 Configuring Web Applications 81
6.5.1 Setting Context Parameters 81
6.5.1.1 To Add a Context Parameter Using NetBeans IDE 82
6.5.1.2 To Create a web.xml File Using NetBeans IDE 82
6.5.2 Declaring Welcome Files 82
6.5.3 Mapping Errors to Error Screens 83
6.5.3.1 To Set Up Error Mapping Using NetBeans IDE 83
6.5.4 Declaring Resource References 84
6.5.4.1 Declaring a Reference to a Resource 85

6.5.4.2 Declaring a Reference to a Web Service 86
6.6 Further Information about Web Applications 86

7 JavaServer Faces Technology 87

7.1 What Is a JavaServer Faces Application? 88
7.2 JavaServer Faces Technology Benefits 89
7.3 A Simple JavaServer Faces Application 91
7.4 User Interface Component Model 92
7.4.1 User Interface Component Classes 93
7.4.2 Component Rendering Model 95
7.4.3 Conversion Model 96
7.4.4 Event and Listener Model 97
7.4.5 Validation Model 99
7.5 Navigation Model 100
7.6 The Lifecycle of a JavaServer Faces Application 103
7.6.1 Overview of the JavaServer Faces Lifecycle 104
7.6.2 Restore View Phase 107
7.6.3 Apply Request Values Phase 107
7.6.4 Process Validations Phase 108
7.6.5 Update Model Values Phase 108
7.6.6 Invoke Application Phase 109
7.6.7 Render Response Phase 109
7.7 Partial Processing and Partial Rendering 110
7.8 Further Information about JavaServer Faces Technology 110

8 Introduction to Facelets 111

8.1 What Is Facelets? 112
8.2 The Lifecycle of a Facelets Application 114
8.3 Developing a Simple Facelets Application: The guessnumber-jsf Example Application 115
8.3.1 Creating a Facelets Application 115
8.3.1.1 Developing a Managed Bean 115
8.3.1.2 Creating Facelets Views 117
8.3.2 Configuring the Application 119
8.3.3 Running the guessnumber-jsf Facelets Example 120
8.3.3.1 To Build, Package, and Deploy the guessnumber-jsf Example Using NetBeans IDE 120

8.3.3.2 To Build, Package, and Deploy the guessnumber-jsf Example Using Maven 121
8.3.3.3 To Run the guessnumber-jsf Example 121
8.4 Using Facelets Templates 121
8.5 Composite Components 124
8.6 Web Resources 127
8.7 Relocatable Resources 128
8.8 Resource Library Contracts 128
8.8.1 The hello1-rlc Example Application 130
8.8.1.1 Configuring the hello1-rlc Example 130
8.8.1.2 The Facelets Pages for the hello1-rlc Example 131
8.8.1.3 To Build, Package, and Deploy the hello1-rlc Example Using NetBeans IDE 131
8.8.1.4 To Build, Package, and Deploy the hello1-rlc Example Using Maven 132
8.8.1.5 To Run the hello1-rlc Example 132
8.9 HTML5-Friendly Markup 132
8.9.1 Using Pass-Through Elements 133
8.9.2 Using Pass-Through Attributes 135
8.9.3 The reservation Example Application 136
8.9.3.1 The Facelets Pages for the reservation Application 137
8.9.3.2 The Managed Bean for the reservation Application 138
8.9.3.3 To Build, Package, and Deploy the reservation Example Using NetBeans IDE 138
8.9.3.4 To Build, Package, and Deploy the reservation Example Using Maven 139
8.9.3.5 To Run the reservation Example 139

9 Expression Language 141

9.1 Overview of the EL 141
9.2 Immediate and Deferred Evaluation Syntax 142
9.2.1 Immediate Evaluation 143
9.2.2 Deferred Evaluation 143
9.3 Value and Method Expressions 144
9.3.1 Value Expressions 144
9.3.1.1 Referencing Objects 145
9.3.1.2 Referencing Object Properties or Collection Elements 145
9.3.1.3 Referencing Literals 146

9.3.1.4 Parameterized Method Calls 147
9.3.1.5 Where Value Expressions Can Be Used 148
9.3.2 Method Expressions 149
9.3.3 Lambda Expressions 150
9.4 Operations on Collection Objects 151
9.5 Operators 153
9.6 Reserved Words 154
9.7 Examples of EL Expressions 154
9.8 Further Information about the Expression Language 155

10 Using JavaServer Faces Technology in Web Pages 157

10.1 Setting Up a Page 158
10.2 Adding Components to a Page Using HTML Tag Library Tags 159
10.2.1 Common Component Tag Attributes 161
10.2.1.1 The id Attribute 162
10.2.1.2 The immediate Attribute 162
10.2.1.3 The rendered Attribute 163
10.2.1.4 The style and styleClass Attributes 164
10.2.1.5 The value and binding Attributes 164
10.2.2 Adding HTML Head and Body Tags 165
10.2.3 Adding a Form Component 165
10.2.4 Using Text Components 166
10.2.4.1 Rendering a Field with the h:inputText Tag 169
10.2.4.2 Rendering a Password Field with the h:inputSecret Tag 170
10.2.4.3 Rendering a Label with the h:outputLabel Tag 170
10.2.4.4 Rendering a Link with the h:outputLink Tag 171
10.2.4.5 Displaying a Formatted Message with the h:outputFormat Tag 171
10.2.5 Using Command Component Tags for Performing Actions and Navigation 172
10.2.5.1 Rendering a Button with the h:commandButton Tag 173
10.2.5.2 Rendering a Link with the h:commandLink Tag 173
10.2.6 Adding Graphics and Images with the h:graphicImage Tag 174
10.2.7 Laying Out Components with the h:panelGrid and h:panelGroup Tags 174
10.2.8 Displaying Components for Selecting One Value 177
10.2.8.1 Displaying a Check Box Using the h:selectBooleanCheckbox Tag 177
10.2.8.2 Displaying a Menu Using the h:selectOneMenu Tag 178
10.2.9 Displaying Components for Selecting Multiple Values 179

10.2.10 Using the f:selectItem and f:selectItems Tags 180
10.2.10.1 Using the f:selectItems Tag.... 181
10.2.10.2 Using the f:selectItem Tag 181
10.2.11 Displaying the Results from Selection Components 182
10.2.12 Using Data-Bound Table Components.... 182
10.2.13 Displaying Error Messages with the h:message and h:messages Tags.... 186
10.2.14 Creating Bookmarkable URLs with the h:button and h:link Tags 187
10.2.15 Using View Parameters to Configure Bookmarkable URLs.... 188
10.2.16 The bookmarks Example Application 189
10.2.16.1 To Build, Package, and Deploy the bookmarks Example Using NetBeans IDE 189
10.2.16.2 To Build, Package, and Deploy the bookmarks Example Using Maven.... 190
10.2.16.3 To Run the bookmarks Example 190
10.2.17 Resource Relocation Using h:outputScript and h:outputStylesheet Tags 190
10.3 Using Core Tags 192

11 Using Converters, Listeners, and Validators.... 197

11.1 Using the Standard Converters.... 198
11.1.1 Converting a Component's Value 199
11.1.2 Using DateTimeConverter.... 200
11.1.3 Using NumberConverter 202
11.2 Registering Listeners on Components 203
11.2.1 Registering a Value-Change Listener on a Component.... 204
11.2.2 Registering an Action Listener on a Component 205
11.3 Using the Standard Validators.... 207
11.3.1 Validating a Component's Value.... 208
11.3.2 Using Validator Tags.... 209
11.4 Referencing a Managed Bean Method 209
11.4.1 Referencing a Method That Performs Navigation 210
11.4.2 Referencing a Method That Handles an Action Event.... 211
11.4.3 Referencing a Method That Performs Validation 211
11.4.4 Referencing a Method That Handles a Value-Change Event.... 212

12 Developing with JavaServer Faces Technology.... 213

12.1 Managed Beans in JavaServer Faces Technology.... 213
12.1.1 Creating a Managed Bean.... 214

12.1.2 Using the EL to Reference Managed Beans 215
12.2 Writing Bean Properties 216
12.2.1 Writing Properties Bound to Component Values........ 217
12.2.1.1 UIInput and UIOutput Properties 218
12.2.1.2 UIData Properties........ 219
12.2.1.3 UISelectBoolean Properties........ 220
12.2.1.4 UISelectMany Properties........ 221
12.2.1.5 UISelectOne Properties 222
12.2.1.6 UISelectItem Properties........ 223
12.2.1.7 UISelectItems Properties 223
12.2.2 Writing Properties Bound to Component Instances 224
12.2.3 Writing Properties Bound to Converters, Listeners, or Validators........ 225
12.3 Writing Managed Bean Methods 226
12.3.1 Writing a Method to Handle Navigation........ 226
12.3.2 Writing a Method to Handle an Action Event 228
12.3.3 Writing a Method to Perform Validation 229
12.3.4 Writing a Method to Handle a Value-Change Event........ 230

13 Using Ajax with JavaServer Faces Technology 231

13.1 Overview of Ajax........ 232
13.2 Using Ajax Functionality with JavaServer Faces Technology 233
13.3 Using Ajax with Facelets 233
13.3.1 Using the f:ajax Tag........ 234
13.4 Sending an Ajax Request........ 236
13.4.1 Using the event Attribute........ 236
13.4.2 Using the execute Attribute 236
13.4.3 Using the immediate Attribute........ 237
13.4.4 Using the listener Attribute 237
13.5 Monitoring Events on the Client 238
13.6 Handling Errors 238
13.7 Receiving an Ajax Response 239
13.8 Ajax Request Lifecycle 240
13.9 Grouping of Components........ 240
13.10 Loading JavaScript as a Resource........ 241
13.10.1 Using JavaScript API in a Facelets Application 242
13.10.2 Using the @ResourceDependency Annotation in a Bean Class........ 243
13.11 The ajaxguessnumber Example Application 243

13.11.1 The ajaxguessnumber Source Files 244
13.11.1.1 The ajaxgreeting.xhtml Facelets Page 244
13.11.1.2 The UserNumberBean Backing Bean 245
13.11.1.3 The DukesNumberBean CDI Managed Bean 245
13.11.2 Running the ajaxguessnumber Example 246
13.11.2.1 To Build, Package, and Deploy the ajaxguessnumber Example Using NetBeans IDE 246
13.11.2.2 To Build, Package, and Deploy the ajaxguessnumber Example Using Maven 246
13.11.2.3 To Run the ajaxguessnumber Example 246
13.12 Further Information about Ajax in JavaServer Faces Technology 247

14 Composite Components: Advanced Topics and an Example 249

14.1 Attributes of a Composite Component 249
14.2 Invoking a Managed Bean 251
14.3 Validating Composite Component Values 251
14.4 The compositecomponentexample Example Application 251
14.4.1 The Composite Component File 252
14.4.2 The Using Page 253
14.4.3 The Managed Bean 253
14.4.4 Running the compositecomponentexample Example 254
14.4.4.1 To Build, Package, and Deploy the compositecomponentexample Example Using NetBeans IDE 254
14.4.4.2 To Build, Package, and Deploy the compositecomponentexample Example Using Maven 255
14.4.4.3 To Run the compositecomponentexample Example 255

15 Creating Custom UI Components and Other Custom Objects 257

15.1 Determining Whether You Need a Custom Component or Renderer 259
15.1.1 When to Use a Custom Component 259
15.1.2 When to Use a Custom Renderer 261
15.1.3 Component, Renderer, and Tag Combinations 261
15.2 Understanding the Image Map Example 262
15.2.1 Why Use JavaServer Faces Technology to Implement an Image Map? 263
15.2.2 Understanding the Rendered HTML 263
15.2.3 Understanding the Facelets Page 264
15.2.4 Configuring Model Data 266

15.2.5 Summary of the Image Map Application Classes 267
15.3 Steps for Creating a Custom Component 268
15.4 Creating Custom Component Classes 268
15.4.1 Specifying the Component Family 272
15.4.2 Performing Encoding 272
15.4.3 Performing Decoding 274
15.4.4 Enabling Component Properties to Accept Expressions 275
15.4.5 Saving and Restoring State 277
15.5 Delegating Rendering to a Renderer 278
15.5.1 Creating the Renderer Class 278
15.5.2 Identifying the Renderer Type 280
15.6 Implementing an Event Listener 281
15.6.1 Implementing Value-Change Listeners 281
15.6.2 Implementing Action Listeners 282
15.7 Handling Events for Custom Components 283
15.8 Defining the Custom Component Tag in a Tag Library Descriptor 284
15.9 Using a Custom Component 285
15.10 Creating and Using a Custom Converter 287
15.10.1 Creating a Custom Converter 288
15.10.2 Using a Custom Converter 291
15.11 Creating and Using a Custom Validator 292
15.11.1 Implementing the Validator Interface 294
15.11.2 Specifying a Custom Tag 296
15.11.3 Using a Custom Validator 297
15.12 Binding Component Values and Instances to Managed Bean Properties 298
15.12.1 Binding a Component Value to a Property 300
15.12.2 Binding a Component Value to an Implicit Object 301
15.12.3 Binding a Component Instance to a Bean Property 302
15.13 Binding Converters, Listeners, and Validators to Managed Bean Properties 303

16 Configuring JavaServer Faces Applications 305

16.1 Using Annotations to Configure Managed Beans 306
16.1.1 Using Managed Bean Scopes 307
16.2 Application Configuration Resource File 307
16.2.1 Configuring Eager Application-Scoped Managed Beans 309
16.2.2 Ordering of Application Configuration Resource Files 309
16.3 Using Faces Flows 311

16.3.1 Packaging Flows in an Application 313
16.3.2 The Simplest Possible Flow: The simple-flow Example Application 313
16.3.2.1 To Build, Package, and Deploy the simple-flow Example Using NetBeans IDE 315
16.3.2.2 To Build, Package, and Deploy the simple-flow Example Using Maven 316
16.3.2.3 To Run the simple-flow Example 316
16.3.3 The checkout-module Example Application 316
16.3.3.1 The Facelets Pages for the checkout-module Example 317
16.3.3.2 Using a Configuration File to Configure a Flow 319
16.3.3.3 Using a Java Class to Configure a Flow 320
16.3.3.4 The Flow-Scoped Managed Beans 322
16.3.3.5 To Build, Package, and Deploy the checkout-module Example Using NetBeans IDE 322
16.3.3.6 To Build, Package, and Deploy the checkout-module Example Using Maven 322
16.3.3.7 To Run the checkout-module Example 323
16.4 Configuring Managed Beans 323
16.4.1 Using the managed-bean Element 324
16.4.2 Initializing Properties Using the managed-property Element 326
16.4.2.1 Referencing a Java Enum Type 327
16.4.2.2 Referencing a Context Initialization Parameter 327
16.4.2.3 Initializing Map Properties 328
16.4.2.4 Initializing Array and List Properties 329
16.4.2.5 Initializing Managed Bean Properties 330
16.4.3 Initializing Maps and Lists 332
16.5 Registering Application Messages 332
16.5.1 Using FacesMessage to Create a Message 333
16.5.2 Referencing Error Messages 334
16.6 Using Default Validators 335
16.7 Registering a Custom Validator 336
16.8 Registering a Custom Converter 337
16.9 Configuring Navigation Rules 337
16.10 Registering a Custom Renderer with a Render Kit 340
16.11 Registering a Custom Component 342
16.12 Basic Requirements of a JavaServer Faces Application 343
16.12.1 Configuring an Application with a Web Deployment Descriptor 344
16.12.1.1 Identifying the Servlet for Lifecycle Processing 345

16.12.1.2 To Specify a Path to an Application Configuration Resource File.......... 346
16.12.1.3 To Specify Where State Is Saved.......... 347
16.12.2 Configuring Project Stage.......... 348
16.12.3 Including the Classes, Pages, and Other Resources.......... 348

17 Java Servlet Technology.......... 351

17.1 What Is a Servlet?.......... 352
17.2 Servlet Lifecycle.......... 352
17.2.1 Handling Servlet Lifecycle Events.......... 353
17.2.1.1 Defining the Listener Class.......... 353
17.2.2 Handling Servlet Errors.......... 354
17.3 Sharing Information.......... 354
17.3.1 Using Scope Objects.......... 354
17.3.2 Controlling Concurrent Access to Shared Resources.......... 355
17.4 Creating and Initializing a Servlet.......... 356
17.5 Writing Service Methods.......... 356
17.5.1 Getting Information from Requests.......... 357
17.5.2 Constructing Responses.......... 358
17.6 Filtering Requests and Responses.......... 359
17.6.1 Programming Filters.......... 359
17.6.2 Programming Customized Requests and Responses.......... 361
17.6.3 Specifying Filter Mappings.......... 361
17.6.3.1 To Specify Filter Mappings Using NetBeans IDE.......... 362
17.7 Invoking Other Web Resources.......... 363
17.7.1 Including Other Resources in the Response.......... 364
17.7.2 Transferring Control to Another Web Component.......... 364
17.8 Accessing the Web Context.......... 365
17.9 Maintaining Client State.......... 365
17.9.1 Accessing a Session.......... 365
17.9.2 Associating Objects with a Session.......... 366
17.9.3 Session Management.......... 366
17.9.3.1 To Set the Timeout Period Using NetBeans IDE.......... 366
17.9.4 Session Tracking.......... 367
17.10 Finalizing a Servlet.......... 367
17.10.1 Tracking Service Requests.......... 368
17.10.2 Notifying Methods to Shut Down.......... 368
17.10.3 Creating Polite Long-Running Methods.......... 369

17.11 Uploading Files with Java Servlet Technology 370
17.11.1 The @MultipartConfig Annotation 370
17.11.2 The getParts and getPart Methods 371
17.12 Asynchronous Processing 372
17.12.1 Asynchronous Processing in Servlets 372
17.12.2 Waiting for a Resource 374
17.13 Nonblocking I/O 375
17.13.1 Reading a Large HTTP POST Request Using Nonblocking I/O 377
17.14 Protocol Upgrade Processing 378
17.15 The mood Example Application 381
17.15.1 Components of the mood Example Application 381
17.15.2 Running the mood Example 382
17.15.2.1 To Run the mood Example Using NetBeans IDE 382
17.15.2.2 To Run the mood Example Using Maven 382
17.16 The fileupload Example Application 383
17.16.1 Architecture of the fileupload Example Application 383
17.16.2 Running the fileupload Example 386
17.16.2.1 To Build, Package, and Deploy the fileupload Example Using NetBeans IDE 386
17.16.2.2 To Build, Package, and Deploy the fileupload Example Using Maven 387
17.16.2.3 To Run the fileupload Example 387
17.17 The dukeetf Example Application 388
17.17.1 Architecture of the dukeetf Example Application 388
17.17.1.1 The Servlet 388
17.17.1.2 The Enterprise Bean 391
17.17.1.3 The HTML Page 392
17.17.2 Running the dukeetf Example Application 393
17.17.2.1 To Run the dukeetf Example Application Using NetBeans IDE 393
17.17.2.2 To Run the dukeetf Example Application Using Maven 393
17.18 Further Information about Java Servlet Technology 394

18 Java API for WebSocket 395

18.1 Introduction to WebSocket 396
18.2 Creating WebSocket Applications in the Java EE Platform 397
18.3 Programmatic Endpoints 398
18.4 Annotated Endpoints 399

18.5 Sending and Receiving Messages 400
18.5.1 Sending Messages 400
18.5.1.1 Sending Messages to All Peers Connected to an Endpoint 401
18.5.2 Receiving Messages 402
18.6 Maintaining Client State 402
18.7 Using Encoders and Decoders 403
18.7.1 Implementing Encoders to Convert Java Objects into WebSocket Messages 403
18.7.2 Implementing Decoders to Convert WebSocket Messages into Java Objects 404
18.8 Path Parameters 406
18.9 Handling Errors 407
18.10 Specifying an Endpoint Configurator Class 407
18.11 The dukeetf2 Example Application 408
18.11.1 Architecture of the dukeetf2 Sample Application 409
18.11.1.1 The Endpoint 409
18.11.1.2 The Enterprise Bean 410
18.11.1.3 The HTML Page 411
18.11.2 Running the dukeetf2 Example Application 412
18.11.2.1 To Run the dukeetf2 Example Application Using NetBeans IDE 412
18.11.2.2 To Run the dukeetf2 Example Application Using Maven 412
18.12 The websocketbot Example Application 413
18.12.1 Architecture of the websocketbot Example Application 413
18.12.1.1 The CDI Bean 413
18.12.1.2 The WebSocket Endpoint 414
18.12.1.3 The Application Messages 416
18.12.1.4 The Encoder Classes 416
18.12.1.5 The Message Decoder 417
18.12.1.6 The HTML Page 418
18.12.2 Running the websocketbot Example Application 418
18.12.2.1 To Run the websocketbot Example Application Using NetBeans IDE 418
18.12.2.2 To Run the websocketbot Example Application Using Maven 419
18.12.2.3 To Test the websocketbot Example Application 419
18.13 Further Information about WebSocket 420

19 **JSON Processing** 421

19.1 Introduction to JSON 421
19.1.1 JSON Syntax 422
19.1.2 Uses of JSON 422
19.1.3 Generating and Parsing JSON Data 423
19.2 JSON Processing in the Java EE Platform 423
19.3 Using the Object Model API 425
19.3.1 Creating an Object Model from JSON Data 425
19.3.2 Creating an Object Model from Application Code 425
19.3.3 Navigating an Object Model 426
19.3.4 Writing an Object Model to a Stream 428
19.4 Using the Streaming API 429
19.4.1 Reading JSON Data Using a Parser 429
19.4.2 Writing JSON Data Using a Generator 431
19.5 JSON in Java EE RESTful Web Services 431
19.6 The jsonpmodel Example Application 432
19.6.1 Components of the jsonpmodel Example Application 432
19.6.2 Running the jsonpmodel Example Application 433
19.6.2.1 To Run the jsonpmodel Example Application Using NetBeans IDE 433
19.6.2.2 To Run the jsonpmodel Example Application Using Maven 433
19.7 The jsonpstreaming Example Application 434
19.7.1 Components of the jsonpstreaming Example Application 434
19.7.2 Running the jsonpstreaming Example Application 434
19.7.2.1 To Run the jsonpstreaming Example Application Using NetBeans IDE 435
19.7.2.2 To Run the jsonpstreaming Example Application Using Maven 435
19.8 Further Information about the Java API for JSON Processing 436

20 **Internationalizing and Localizing Web Applications** 437

20.1 Java Platform Localization Classes 437
20.2 Providing Localized Messages and Labels 438
20.2.1 Establishing the Locale 439
20.2.2 Setting the Resource Bundle 439
20.2.3 Retrieving Localized Messages 440
20.3 Date and Number Formatting 441
20.4 Character Sets and Encodings 442

20.4.1 Character Sets ... 442
20.4.2 Character Encoding ... 442

Part IV Bean Validation ... 445

21 Introduction to Bean Validation ... 447

21.1 Using Bean Validation Constraints ... 447
21.2 Validating Null and Empty Strings ... 450
21.3 Validating Constructors and Methods ... 451
21.3.1 Cross-Parameter Constraints ... 452
21.3.2 Identifying Parameter Constraint Violations ... 453
21.3.3 Adding Constraints to Method Return Values ... 453
21.4 Further Information about Bean Validation ... 454

22 Bean Validation: Advanced Topics ... 455

22.1 Creating Custom Constraints ... 455
22.1.1 Using the Built-In Constraints to Make a New Constraint ... 455
22.1.2 Removing Ambiguity in Constraint Targets ... 456
22.2 Customizing Validator Messages ... 457
22.2.1 The ValidationMessages Resource Bundle ... 457
22.2.1.1 Localizing Validation Messages ... 457
22.3 Grouping Constraints ... 457
22.3.1 Customizing Group Validation Order ... 458
22.4 Using Method Constraints in Type Hierarchies ... 458
22.4.1 Rules for Using Method Constraints in Type Hierarchies ... 460

Part V Contexts and Dependency Injection for Java EE ... 461

23 Introduction to Contexts and Dependency Injection for Java EE ... 463

23.1 Getting Started ... 464
23.2 Overview of CDI ... 466
23.3 About Beans ... 467
23.4 About CDI Managed Beans ... 468
23.5 Beans as Injectable Objects ... 468
23.6 Using Qualifiers ... 469
23.7 Injecting Beans ... 470

23.8 Using Scopes ... 471
23.9 Giving Beans EL Names ... 473
23.10 Adding Setter and Getter Methods ... 473
23.11 Using a Managed Bean in a Facelets Page ... 474
23.12 Injecting Objects by Using Producer Methods ... 474
23.13 Configuring a CDI Application ... 475
23.14 Using the @PostConstruct and @PreDestroy Annotations with CDI Managed Bean Classes ... 476
23.14.1 To Initialize a Managed Bean Using the @PostConstruct Annotation ... 476
23.14.2 To Prepare for the Destruction of a Managed Bean Using the @PreDestroy Annotation ... 477
23.15 Further Information about CDI ... 477

24 Running the Basic Contexts and Dependency Injection Examples ... 479

24.1 The simplegreeting CDI Example ... 479
24.1.1 The simplegreeting Source Files ... 480
24.1.2 The Facelets Template and Page ... 480
24.1.3 Running the simplegreeting Example ... 482
24.1.3.1 To Build, Package, and Deploy the simplegreeting Example Using NetBeans IDE ... 482
24.1.3.2 To Build, Package, and Deploy the simplegreeting Example Using Maven ... 483
24.1.3.3 To Run the simplegreeting Example ... 483
24.2 The guessnumber-cdi CDI Example ... 483
24.2.1 The guessnumber-cdi Source Files ... 484
24.2.1.1 The @MaxNumber and @Random Qualifier Interfaces ... 484
24.2.1.2 The Generator Managed Bean ... 485
24.2.1.3 The UserNumberBean Managed Bean ... 486
24.2.2 The Facelets Page ... 488
24.2.3 Running the guessnumber-cdi Example ... 490
24.2.3.1 To Build, Package, and Deploy the guessnumber-cdi Example Using NetBeans IDE ... 490
24.2.3.2 To Build, Package, and Deploy the guessnumber-cdi Example Using Maven ... 491
24.2.3.3 To Run the guessnumber Example ... 491

25 Contexts and Dependency Injection for Java EE: Advanced Topics 493

25.1 Packaging CDI Applications 493
25.2 Using Alternatives in CDI Applications 494
25.2.1 Using Specialization 496
25.3 Using Producer Methods, Producer Fields, and Disposer Methods in CDI Applications 497
25.3.1 Using Producer Methods 497
25.3.2 Using Producer Fields to Generate Resources 498
25.3.3 Using a Disposer Method 499
25.4 Using Predefined Beans in CDI Applications 499
25.5 Using Events in CDI Applications 501
25.5.1 Defining Events 501
25.5.2 Using Observer Methods to Handle Events 501
25.5.3 Firing Events 503
25.6 Using Interceptors in CDI Applications 504
25.7 Using Decorators in CDI Applications 506
25.8 Using Stereotypes in CDI Applications 507

26 Running the Advanced Contexts and Dependency Injection Examples 511

26.1 The encoder Example: Using Alternatives 511
26.1.1 The Coder Interface and Implementations 512
26.1.2 The encoder Facelets Page and Managed Bean 513
26.1.3 Running the encoder Example 514
26.1.3.1 To Build, Package, and Deploy the encoder Example Using NetBeans IDE 514
26.1.3.2 To Run the encoder Example Using NetBeans IDE 515
26.1.3.3 To Build, Package, and Deploy the encoder Example Using Maven 516
26.1.3.4 To Run the encoder Example Using Maven 516
26.2 The producermethods Example: Using a Producer Method to Choose a Bean Implementation 517
26.2.1 Components of the producermethods Example 517
26.2.2 Running the producermethods Example 519
26.2.2.1 To Build, Package, and Deploy the producermethods Example Using NetBeans IDE 519
26.2.2.2 To Build, Package, and Deploy the producermethods Example Using Maven 519

26.2.2.3 To Run the producermethods Example 520
26.3 The producerfields Example: Using Producer Fields to Generate Resources..... 520
26.3.1 The Producer Field for the producerfields Example 520
26.3.2 The producerfields Entity and Session Bean 522
26.3.3 The producerfields Facelets Pages and Managed Bean 524
26.3.4 Running the producerfields Example..... 526
26.3.4.1 To Build, Package, and Deploy the producerfields Example Using NetBeans IDE 526
26.3.4.2 To Build, Package, and Deploy the producerfields Example Using Maven..... 526
26.3.4.3 To Run the producerfields Example 527
26.4 The billpayment Example: Using Events and Interceptors 527
26.4.1 The PaymentEvent Event Class 527
26.4.2 The PaymentHandler Event Listener..... 528
26.4.3 The billpayment Facelets Pages and Managed Bean..... 529
26.4.4 The LoggedInterceptor Interceptor Class..... 532
26.4.5 Running the billpayment Example..... 533
26.4.5.1 To Build, Package, and Deploy the billpayment Example Using NetBeans IDE 533
26.4.5.2 To Build, Package, and Deploy the billpayment Example Using Maven..... 533
26.4.5.3 To Run the billpayment Example..... 534
26.5 The decorators Example: Decorating a Bean..... 535
26.5.1 Components of the decorators Example..... 535
26.5.2 Running the decorators Example 536
26.5.2.1 To Build, Package, and Deploy the decorators Example Using NetBeans IDE 536
26.5.2.2 To Build, Package, and Deploy the decorators Example Using Maven..... 536
26.5.2.3 To Run the decorators Example 537

Part VI Web Services 539

27 Introduction to Web Services..... 541

27.1 What Are Web Services? 541
27.2 Types of Web Services..... 541
27.2.1 "Big" Web Services 542

27.2.2 RESTful Web Services 542
27.3 Deciding Which Type of Web Service to Use 544

28 **Building Web Services with JAX-WS** 545

28.1 Creating a Simple Web Service and Clients with JAX-WS 546
28.1.1 Requirements of a JAX-WS Endpoint 547
28.1.2 Coding the Service Endpoint Implementation Class 548
28.1.3 Building, Packaging, and Deploying the Service 549
28.1.3.1 To Build, Package, and Deploy the Service Using NetBeans IDE 549
28.1.3.2 To Build, Package, and Deploy the Service Using Maven 549
28.1.4 Testing the Methods of a Web Service Endpoint 550
28.1.4.1 To Test the Service without a Client 550
28.1.5 A Simple JAX-WS Application Client 550
28.1.5.1 Coding the Application Client 550
28.1.5.2 Running the Application Client 551
28.1.6 A Simple JAX-WS Web Client 552
28.1.6.1 Coding the Servlet 553
28.1.6.2 Running the Web Client 554
28.2 Types Supported by JAX-WS 555
28.2.1 Schema-to-Java Mapping 556
28.2.2 Java-to-Schema Mapping 557
28.3 Web Services Interoperability and JAX-WS 557
28.4 Further Information about JAX-WS 558

29 **Building RESTful Web Services with JAX-RS** 559

29.1 What Are RESTful Web Services? 559
29.2 Creating a RESTful Root Resource Class 561
29.2.1 Developing RESTful Web Services with JAX-RS 561
29.2.2 Overview of a JAX-RS Application 563
29.2.3 The @Path Annotation and URI Path Templates 564
29.2.4 Responding to HTTP Methods and Requests 567
29.2.4.1 The Request Method Designator Annotations 567
29.2.4.2 Using Entity Providers to Map HTTP Response and Request Entity Bodies 568
29.2.5 Using @Consumes and @Produces to Customize Requests and Responses 570
29.2.5.1 The @Produces Annotation 570

29.2.5.2 The @Consumes Annotation ... 572
29.2.6 Extracting Request Parameters ... 573
29.2.7 Configuring JAX-RS Applications ... 577
29.2.7.1 Configuring a JAX-RS Application Using a Subclass of Application ... 577
29.2.7.2 Configuring the Base URI in web.xml ... 577
29.3 Example Applications for JAX-RS ... 578
29.3.1 Creating a Simple RESTful Web Service ... 578
29.3.1.1 To Create a RESTful Web Service Using NetBeans IDE ... 578
29.3.2 The rsvp Example Application ... 579
29.3.2.1 Components of the rsvp Example Application ... 580
29.3.2.2 Running the rsvp Example Application ... 581
29.3.3 Real-World Examples ... 583
29.4 Further Information about JAX-RS ... 583

30 Accessing REST Resources with the JAX-RS Client API ... 585

30.1 Overview of the Client API ... 585
30.1.1 Creating a Basic Client Request Using the Client API ... 585
30.1.1.1 Obtaining the Client Instance ... 586
30.1.1.2 Setting the Client Target ... 586
30.1.1.3 Setting Path Parameters in Targets ... 587
30.1.1.4 Invoking the Request ... 587
30.2 Using the Client API in the JAX-RS Example Applications ... 589
30.2.1 The Client API in the rsvp Example Application ... 589
30.2.2 The Client API in the customer Example Application ... 590
30.3 Advanced Features of the Client API ... 592
30.3.1 Configuring the Client Request ... 592
30.3.1.1 Setting Message Headers in the Client Request ... 592
30.3.1.2 Setting Cookies in the Client Request ... 593
30.3.1.3 Adding Filters to the Client ... 593
30.3.2 Asynchronous Invocations in the Client API ... 595
30.3.2.1 Using Custom Callbacks in Asynchronous Invocations ... 595

31 JAX-RS: Advanced Topics and an Example ... 597

31.1 Annotations for Field and Bean Properties of Resource Classes ... 598
31.1.1 Extracting Path Parameters ... 598
31.1.2 Extracting Query Parameters ... 599

31.1.3 Extracting Form Data 599
31.1.4 Extracting the Java Type of a Request or Response 600
31.2 Validating Resource Data with Bean Validation 601
31.2.1 Using Constraint Annotations on Resource Methods 601
31.2.2 Validating Entity Data 603
31.2.3 Validation Exception Handling and Response Codes 604
31.3 Subresources and Runtime Resource Resolution 604
31.3.1 Subresource Methods 605
31.3.2 Subresource Locators 605
31.4 Integrating JAX-RS with EJB Technology and CDI 606
31.5 Conditional HTTP Requests 608
31.6 Runtime Content Negotiation 609
31.7 Using JAX-RS with JAXB 611
31.7.1 Using Java Objects to Model Your Data 613
31.7.2 Starting from an Existing XML Schema Definition 614
31.7.3 Using JSON with JAX-RS and JAXB 616
31.8 The customer Example Application 618
31.8.1 Overview of the customer Example Application 618
31.8.2 The Customer and Address Entity Classes 618
31.8.3 The CustomerService Class 621
31.8.4 Using the JAX-RS Client in the CustomerBean Classes 623
31.8.5 Running the customer Example 624
31.8.5.1 To Build, Package, and Deploy the customer Example Using NetBeans IDE 625
31.8.5.2 To Build, Package, and Deploy the customer Example Using Maven 625

Index 627

Preface

This tutorial is a guide to developing enterprise applications for the Java Platform, Enterprise Edition 7 (Java EE 7), using GlassFish Server Open Source Edition.

GlassFish Server Open Source Edition is the leading open-source and open-community platform for building and deploying next-generation applications and services. GlassFish Server Open Source Edition, developed by the GlassFish project open-source community at `https://glassfish.java.net/`, is the first compatible implementation of the Java EE 7 platform specification. This lightweight, flexible, and open-source application server enables organizations not only to leverage the new capabilities introduced within the Java EE 7 specification, but also to add to their existing capabilities through a faster and more streamlined development and deployment cycle. GlassFish Server Open Source Edition is hereafter referred to as GlassFish Server.

The following topics are addressed here:

- Audience
- Before You Read This Book
- Related Documentation
- The Oracle Accessibility Program
- Conventions
- Default Paths and File Names
- Acknowledgments

Audience

This tutorial is intended for programmers interested in developing and deploying Java EE 7 applications. It covers the technologies comprising the Java EE platform and describes how to develop Java EE components and deploy them on the Java EE Software Development Kit (SDK).

Before You Read This Book

Before proceeding with this book, you should have a good knowledge of the Java programming language. A good way to get to that point is to work through the Java Tutorials (`http://docs.oracle.com/javase/tutorial/`).

Related Documentation

The Java EE 7 Tutorial, Volume 2 covers Java EE 7 technologies and topics not included in this volume, including Enterprise JavaBeans, Java Persistence, the Java Message Service, security, and others. The volume concludes with three case studies that incorporate multiple Java EE 7 technologies.

The GlassFish Server documentation set describes deployment planning and system installation. To obtain documentation for GlassFish Server Open Source Edition, go to `https://glassfish.java.net/docs/`.

The Java EE 7 API specification can be viewed at `http://docs.oracle.com/javaee/7/api/` and is also provided in the Java EE 7 SDK.

Additionally, the Java EE Specifications at `http://www.oracle.com/technetwork/java/javaee/tech/` might be useful.

For information about creating enterprise applications in the NetBeans Integrated Development Environment (IDE), see `https://netbeans.org/kb/`.

For information about the Java DB database for use with GlassFish Server, see `http://www.oracle.com/technetwork/java/javadb/overview/`.

The GlassFish Samples project is a collection of sample applications that demonstrate a broad range of Java EE technologies. The GlassFish Samples are bundled with the Java EE Software Development Kit (SDK) and are also available from the GlassFish Samples project page at `https://glassfish-samples.java.net/`.

The Oracle Accessibility Program

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Conventions

The following table describes the typographic conventions that are used in this book.

Convention	Meaning	Example
Boldface	Boldface type indicates graphical user interface elements associated with an action or terms defined in text.	From the **File** menu, choose **Open Project**. A **cache** is a copy that is stored locally.
`Monospace`	Monospace type indicates the names of files and directories, commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
Italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.	Read Chapter 6 in the *User's Guide*. Do *not* save the file. The command to remove a file is `rm` *filename*.

Default Paths and File Names

The following table describes the default paths and file names that are used in this book.

Placeholder	Description	Default Value
as-install	Represents the base installation directory for GlassFish Server or the SDK of which GlassFish Server is a part.	Installations on the Solaris operating system, Linux operating system, and Mac operating system: *user's-home-directory*`/glassfish4/glassfish` Windows, all installations: *SystemDrive*`:\glassfish4\glassfish`

Placeholder	Description	Default Value
as-install-parent	Represents the parent of the base installation directory for GlassFish Server.	Installations on the Solaris operating system, Linux operating system, and Mac operating system: *user's-home-directory*`/glassfish4` Windows, all installations: *SystemDrive*`:\glassfish4`
tut-install	Represents the base installation directory for the *Java EE Tutorial* after you install GlassFish Server or the SDK and run the Update Tool.	*as-install-parent*`/docs/javaee-tutorial`
domain-dir	Represents the directory in which a domain's configuration is stored.	*as-install*`/domains/domain1`

Acknowledgments

The Java EE tutorial team would like to thank the Java EE specification leads: Linda DeMichiel, Bill Shannon, Emmanuel Bernard, Ed Burns, Shing Wai Chan, Kin-Man Chung, Danny Coward, Nigel Deakin, Rod Johnson, Roger Kitain, Jitendra Kotamraju, Anthony Lai, Bob Lee, Ron Monzillo, Rajiv Mordani, Pete Muir, Paul Parkinson, Santiago Pericas-Geertsen, Marek Potociar, Sivakumar Thyagarajan, Marina Vatkina, and Chris Vignola.

We would also like to thank the Java EE 7 SDK team, especially Snjezana Sevo-Zenzerovic, Adam Leftik, Michael Chen, and John Clingan.

The JavaServer Faces technology chapters benefited greatly from suggestions by Manfred Riem as well as by the spec leads.

We would like to thank our manager, Alan Sommerer, for his support and steadying influence.

We also thank Jordan Douglas and Dawn Tyler for developing and updating the illustrations. Edna Elle provided invaluable help with tools. Sheila Cepero helped smooth our path in many ways.

Finally, we would like to express our profound appreciation to Greg Doench, Elizabeth Ryan, Caroline Senay, and the production team at Addison-Wesley for graciously seeing our manuscript to publication.

This page intentionally left blank

6

Getting Started with Web Applications

This chapter introduces web applications, which typically use JavaServer Faces technology and/or Java Servlet technology. A **web application** is a dynamic extension of a web or application server. Web applications are of the following types:

- **Presentation-oriented**: A **presentation-oriented web application** generates interactive web pages containing various types of markup language (HTML, XHTML, XML, and so on) and dynamic content in response to requests. Development of presentation-oriented web applications is covered in Chapter 7, "JavaServer Faces Technology," through Chapter 20, "Internationalizing and Localizing Web Applications."
- **Service-oriented**: A **service-oriented web application** implements the endpoint of a web service. Presentation-oriented applications are often clients of service-oriented web applications. Development of service-oriented web applications is covered in Chapter 28, "Building Web Services with JAX-WS," through Chapter 31, "JAX-RS: Advanced Topics and an Example," in Part VI, "Web Services."

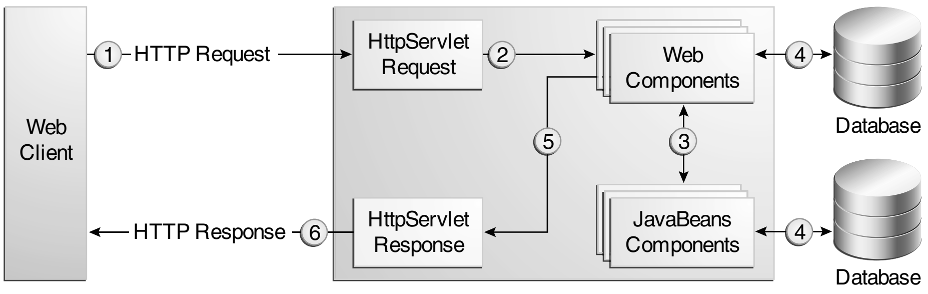
The following topics are addressed here:

- Web Applications
- Web Application Lifecycle
- A Web Module That Uses JavaServer Faces Technology: The hello1 Example
- A Web Module That Uses Java Servlet Technology: The hello2 Example
- Configuring Web Applications
- Further Information about Web Applications

6.1 Web Applications

In the Java EE platform, **web components** provide the dynamic extension capabilities for a web server. Web components can be Java servlets, web pages implemented with JavaServer Faces technology, web service endpoints, or JSP pages. Figure 6–1 illustrates the interaction between a web client and a web application that uses a servlet. The client sends an HTTP request to the web server. A web server that implements Java Servlet and JavaServer Pages technology converts the request into an `HTTPServletRequest` object. This object is delivered to a web component, which can interact with JavaBeans components or a database to generate dynamic content. The web component can then generate an `HTTPServletResponse` or can pass the request to another web component. A web component eventually generates a `HTTPServletResponse` object. The web server converts this object to an HTTP response and returns it to the client.

Figure 6–1 Java Web Application Request Handling



Servlets are Java programming language classes that dynamically process requests and construct responses. Java technologies, such as JavaServer Faces and Facelets, are used for building interactive web applications. (Frameworks can also be used for this purpose.) Although servlets and JavaServer Faces and Facelets pages can be used to accomplish similar things, each has its own strengths. Servlets are best suited for service-oriented applications (web service endpoints can be implemented as servlets) and the control functions of a presentation-oriented application, such as dispatching requests and handling nontextual data. JavaServer Faces and Facelets pages are more appropriate for generating text-based markup, such as XHTML, and are generally used for presentation-oriented applications.

Web components are supported by the services of a runtime platform called a **web container**. A web container provides such services as request dispatching, security, concurrency, and lifecycle management. A web container also gives web components access to such APIs as naming, transactions, and email.

Certain aspects of web application behavior can be configured when the application is installed, or *deployed*, to the web container. The configuration information can be specified using Java EE annotations or can be maintained in a text file in XML format called a web application deployment descriptor (DD). A web application DD must conform to the schema described in the Java Servlet specification.

This chapter gives a brief overview of the activities involved in developing web applications. First, it summarizes the web application lifecycle and explains how to package and deploy very simple web applications on GlassFish Server. The chapter then moves on to configuring web applications and discusses how to specify the most commonly used configuration parameters.

6.2 Web Application Lifecycle

A web application consists of web components; static resource files, such as images and cascading style sheets (CSS); and helper classes and libraries. The web container provides many supporting services that enhance the capabilities of web components and make them easier to develop. However, because a web application must take these services into account, the process for creating and running a web application is different from that of traditional stand-alone Java classes.

The process for creating, deploying, and executing a web application can be summarized as follows:

1. Develop the web component code.
2. Develop the web application deployment descriptor, if necessary.
3. Compile the web application components and helper classes referenced by the components.
4. Optionally, package the application into a deployable unit.
5. Deploy the application into a web container.
6. Access a URL that references the web application.

Developing web component code is covered in the later chapters. Steps 2 through 4 are expanded on in the following sections and illustrated with a Hello, World–style, presentation-oriented application. This application allows a user to enter a name into an HTML form and then displays a greeting after the name is submitted.

The Hello application contains two web components that generate the greeting and the response. This chapter discusses the following simple applications:

- `hello1`, a JavaServer Faces technology–based application that uses two XHTML pages and a managed bean
- `hello2`, a servlet-based web application in which the components are implemented by two servlet classes

The applications are used to illustrate tasks involved in packaging, deploying, configuring, and running an application that contains web components.

6.3 A Web Module That Uses JavaServer Faces Technology: The hello1 Example

The `hello1` application is a web module that uses JavaServer Faces technology to display a greeting and response. You can use a text editor to view the application files, or you can use NetBeans IDE.

The source code for this application is in the *tut-install*`/examples/web/jsf/hello1/` directory.

6.3.1 To View the hello1 Web Module Using NetBeans IDE

1. From the **File** menu, choose **Open Project**.
2. In the Open Project dialog box, navigate to:

   tut-install`/examples/web/jsf`

3. Select the `hello1` folder and click **Open Project**.
4. Expand the **Web Pages** node and double-click the `index.xhtml` file to view it in the editor.

   The `index.xhtml` file is the default landing page for a Facelets application. In a typical Facelets application, web pages are created in XHTML. For this application, the page uses simple tag markup to display a form with a graphic image, a header, a field, and two command buttons:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html lang="en"
      xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Facelets Hello Greeting</title>
    </h:head>
    <h:body>
        <h:form>
            <h:graphicImage url="#{resource['images:duke.waving.gif']}"
                            alt="Duke waving his hand"/>
            <h2>Hello, my name is Duke. What's yours?</h2>
            <h:inputText id="username"
                         title="My name is: "
                         value="#{hello.name}"
                         required="true"
                         requiredMessage="Error: A name is required."
                         maxlength="25" />
            <p></p>
            <h:commandButton id="submit" value="Submit"
                             action="response" />
            <h:commandButton id="reset" value="Reset" type="reset" />
        </h:form>
        ...
    </h:body>
</html>
```

The most complex element on the page is the `inputText` field. The `maxlength` attribute specifies the maximum length of the field. The `required` attribute specifies that the field must be filled out; the `requiredMessage` attribute provides the error message to be displayed if the field is left empty. The `title` attribute provides the text to be used by screen readers for the visually disabled. Finally, the `value` attribute contains an expression that will be provided by the `Hello` managed bean.

The web page connects to the `Hello` managed bean through the Expression Language (EL) value expression `#{hello.name}`, which retrieves the value of the `name` property from the managed bean. Note the use of `hello` to reference the managed bean `Hello`. If no name is specified in the `@Named` annotation of the managed bean, the managed bean is always accessed with the first letter of the class name in lowercase.

The Submit `commandButton` element specifies the action as `response`, meaning that when the button is clicked, the `response.xhtml` page is displayed.

5. Double-click the `response.xhtml` file to view it.

The response page appears. Even simpler than the greeting page, the response page contains a graphic image, a header that displays the expression provided by the managed bean, and a single button whose `action` element transfers you back to the `index.xhtml` page:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html lang="en"
      xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Facelets Hello Response</title>
    </h:head>
    <h:body>
        <h:form>
            <h:graphicImage url="#{resource['images:duke.waving.gif']}"
                            alt="Duke waving his hand"/>
            <h2>Hello, #{hello.name}!</h2>
            <p></p>
            <h:commandButton id="back" value="Back" action="index" />
        </h:form>
    </h:body>
</html>
```

6. Expand the **Source Packages** node, then the `javaeetutorial.hello1` node.
7. Double-click the `Hello.java` file to view it.

   The `Hello` class, called a managed bean class, provides getter and setter methods for the `name` property used in the Facelets page expressions. By default, the expression language refers to the class name, with the first letter in lowercase (`hello.name`).

```
package javaeetutorial.hello1;

import javax.enterprise.context.RequestScoped;
import javax.inject.Named;

@Named
@RequestScoped
public class Hello {
    private String name;

    public Hello() {
    }

    public String getName() {
        return name;
    }
    public void setName(String user_name) {
        this.name = user_name;
    }
}
```

If you use the default name for the bean class, you can specify `@Model` as the annotation instead of having to specify both `@Named` and `@RequestScoped`. The `@Model` annotation is called a **stereotype**, a term for an annotation that encapsulates other annotations. It is described later in Section 25.8, "Using Stereotypes in CDI Applications." Some examples will use `@Model` where it is appropriate.

8. Under the **Web Pages** node, expand the **WEB-INF** node and double-click the `web.xml` file to view it.

   The `web.xml` file contains several elements that are required for a Facelets application. All of the following are created automatically when you use NetBeans IDE to create an application.

   - A context parameter specifying the project stage:

     ```
     <context-param>
         <param-name>javax.faces.PROJECT_STAGE</param-name>
         <param-value>Development</param-value>
     </context-param>
     ```

     A context parameter provides configuration information needed by a web application. An application can define its own context parameters. In addition, JavaServer Faces technology and Java Servlet technology define context parameters that an application can use.

   - A `servlet` element and its `servlet-mapping` element specifying the `FacesServlet`. All files with the `.xhtml` suffix will be matched:

     ```
     <servlet>
         <servlet-name>Faces Servlet</servlet-name>
         <servlet-class>
             javax.faces.webapp.FacesServlet
         </servlet-class>
         <load-on-startup>1</load-on-startup>
     </servlet>
     <servlet-mapping>
         <servlet-name>Faces Servlet</servlet-name>
         <url-pattern>*.xhtml</url-pattern>
     </servlet-mapping>
     ```

- A `welcome-file-list` element specifying the location of the landing page:

  ```
  <welcome-file-list>
      <welcome-file>index.xhtml</welcome-file>
  </welcome-file-list>
  ```

6.3.1.1 Introduction to Scopes

In the `Hello.java` class, the annotations `javax.inject.Named` and `javax.enterprise.context.RequestScoped` identify the class as a managed bean using request scope. Scope defines how application data persists and is shared.

The most commonly used scopes in JavaServer Faces applications are the following:

- **Request (**`@RequestScoped`**)**: Request scope persists during a single HTTP request in a web application. In an application like `hello1`, in which the application consists of a single request and response, the bean uses request scope.
- **Session (**`@SessionScoped`**)**: Session scope persists across multiple HTTP requests in a web application. When an application consists of multiple requests and responses where data needs to be maintained, beans use session scope.
- **Application (**`@ApplicationScoped`**)**: Application scope persists across all users' interactions with a web application.

For more information on scopes in JavaServer Faces technology, see Section 16.1.1, "Using Managed Bean Scopes."

6.3.2 Packaging and Deploying the hello1 Web Module

A web module must be packaged into a WAR in certain deployment scenarios and whenever you want to distribute the web module. You can package a web module into a WAR file by using Maven or by using the IDE tool of your choice. This tutorial shows you how to use NetBeans IDE or Maven to build, package, and deploy the `hello1` sample application.

You can deploy a WAR file to GlassFish Server by:

- Using NetBeans IDE
- Using the `asadmin` command
- Using the Administration Console
- Copying the WAR file into the *domain-dir*`/autodeploy/` directory

Throughout the tutorial, you will use NetBeans IDE or Maven for packaging and deploying.

6.3.2.1 To Build and Package the hello1 Web Module Using NetBeans IDE

1. Start GlassFish Server as described in Section 2.2.1, "To Start GlassFish Server Using NetBeans IDE," if you have not already done so.
2. From the **File** menu, choose **Open Project**.
3. In the Open Project dialog box, navigate to:

   ```
   tut-install/examples/web/jsf
   ```

4. Select the `hello1` folder.
5. Click **Open Project**.
6. In the **Projects** tab, right-click the `hello1` project and select **Build**. This command deploys the project to the server.

6.3.2.2 To Build and Package the hello1 Web Module Using Maven

1. Start GlassFish Server as described in Section 2.2.3, "To Start GlassFish Server Using the Command Line," if you have not already done so.
2. In a terminal window, go to:

   ```
   tut-install/examples/web/jsf/hello1/
   ```

3. Enter the following command:

   ```
   mvn install
   ```

   This command spawns any necessary compilations and creates the WAR file in *tut-install*`/examples/web/jsf/hello1/target/`. It then deploys the project to the server.

6.3.3 Viewing Deployed Web Modules

GlassFish Server provides two ways to view the deployed web modules: the Administration Console and the `asadmin` command. You can also use NetBeans IDE to view deployed modules.

6.3.3.1 To View Deployed Web Modules Using the Administration Console

1. Open the URL `http://localhost:4848/` in a browser.
2. Select the **Applications** node.

The deployed web modules appear in the **Deployed Applications** table.

6.3.3.2 To View Deployed Web Modules Using the asadmin Command

Enter the following command:

```
asadmin list-applications
```

6.3.3.3 To View Deployed Web Modules Using NetBeans IDE

1. In the **Services** tab, expand the **Servers** node, then expand the GlassFish Server node.
2. Expand the **Applications** node to view the deployed modules.

6.3.4 Running the Deployed hello1 Web Module

Now that the web module is deployed, you can view it by opening the application in a web browser. By default, the application is deployed to host `localhost` on port 8080. The context root of the web application is `hello1`.

1. Open a web browser.
2. Enter the following URL:

   ```
   http://localhost:8080/hello1/
   ```

3. In the field, enter your name and click **Submit**.

   The response page displays the name you submitted. Click **Back** to try again.

6.3.4.1 Dynamic Reloading of Deployed Modules

If dynamic reloading is enabled, you do not have to redeploy an application or module when you change its code or deployment descriptors. All you have to do is copy the changed pages or class files into the deployment directory for the application or module. The deployment directory for a web module named *context-root* is *domain-dir*`/applications/`*context-root*. The server checks for changes periodically and redeploys the application, automatically and dynamically, with the changes.

This capability is useful in a development environment because it allows code changes to be tested quickly. Dynamic reloading is not recommended for a production environment, however, because it may degrade performance. In addition, whenever a reload takes place, the sessions at that time become invalid, and the client must restart the session.

In GlassFish Server, dynamic reloading is enabled by default.

6.3.5 Undeploying the hello1 Web Module

You can undeploy web modules and other types of enterprise applications by using either NetBeans IDE or the `asadmin` command.

6.3.5.1 To Undeploy the hello1 Web Module Using NetBeans IDE

1. In the **Services** tab, expand the **Servers** node, then expand the GlassFish Server node.
2. Expand the **Applications** node.
3. Right-click the `hello1` module and select **Undeploy**.
4. To delete the class files and other build artifacts, go back to the **Projects** tab, right-click the project, and select **Clean**.

6.3.5.2 To Undeploy the hello1 Web Module Using the asadmin Command

1. In a terminal window, go to:

   `tut-install``/examples/web/jsf/hello1/`

2. Enter the following command:

   ```
   mvn cargo:undeploy
   ```

3. To delete the class files and other build artifacts, enter the following command:

   ```
   mvn clean
   ```

6.4 A Web Module That Uses Java Servlet Technology: The hello2 Example

The `hello2` application is a web module that uses Java Servlet technology to display a greeting and response. You can use a text editor to view the application files, or you can use NetBeans IDE.

The source code for this application is in the *tut-install*`/examples/web/servlet/hello2/` directory.

6.4.1 Mapping URLs to Web Components

When it receives a request, the web container must determine which web component should handle the request. The web container does so by mapping the

URL path contained in the request to a web application and a web component. A URL path contains the context root and, optionally, a URL pattern:

```
http://host:port/context-root[/url-pattern]
```

You set the URL pattern for a servlet by using the `@WebServlet` annotation in the servlet source file. For example, the `GreetingServlet.java` file in the `hello2` application contains the following annotation, specifying the URL pattern as `/greeting`:

```
@WebServlet("/greeting")
public class GreetingServlet extends HttpServlet {
    ...
```

This annotation indicates that the URL pattern `/greeting` follows the context root. Therefore, when the servlet is deployed locally, it is accessed with the following URL:

```
http://localhost:8080/hello2/greeting
```

To access the servlet by using only the context root, specify `"/"` as the URL pattern.

6.4.2 Examining the hello2 Web Module

The `hello2` application behaves almost identically to the `hello1` application, but it is implemented using Java Servlet technology instead of JavaServer Faces technology. You can use a text editor to view the application files, or you can use NetBeans IDE.

6.4.2.1 To View the hello2 Web Module Using NetBeans IDE

1. From the **File** menu, choose **Open Project**.
2. In the Open Project dialog box, navigate to:

   ```
   tut-install/examples/web/servlet
   ```

3. Select the `hello2` folder and click **Open Project**.
4. Expand the **Source Packages** node, then expand the `javaeetutorial.hello2` node.
5. Double-click the `GreetingServlet.java` file to view it.

   This servlet overrides the `doGet` method, implementing the `GET` method of HTTP. The servlet displays a simple HTML greeting form whose **Submit** button, like that of `hello1`, specifies a response page for its action. The following excerpt begins with the `@WebServlet` annotation, which specifies the URL pattern relative to the context root:

```
@WebServlet("/greeting")
public class GreetingServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        response.setBufferSize(8192);
        try (PrintWriter out = response.getWriter()) {
            out.println("<html lang=\"en\">"
                    + "<head><title>Servlet Hello</title></head>");

            // then write the data of the response
            out.println("<body  bgcolor=\"#ffffff\">"
                + "<img src=\"duke.waving.gif\" "
                + "alt=\"Duke waving his hand\">"
                + "<form method=\"get\">"
                + "<h2>Hello, my name is Duke. What's yours?</h2>"
                + "<input title=\"My name is: \"type=\"text\" "
                + "name=\"username\" size=\"25\">"
                + "<p></p>"
                + "<input type=\"submit\" value=\"Submit\">"
                + "<input type=\"reset\" value=\"Reset\">"
                + "</form>");

            String username = request.getParameter("username");
            if (username != null && username.length()> 0) {
                RequestDispatcher dispatcher =
                    getServletContext().getRequestDispatcher("/response");

                if (dispatcher != null) {
                    dispatcher.include(request, response);
                }
            }
            out.println("</body></html>");
        }
    }
    ...
```

6. Double-click the `ResponseServlet.java` file to view it.

This servlet also overrides the `doGet` method, displaying only the response. The following excerpt begins with the `@WebServlet` annotation, which specifies the URL pattern relative to the context root:

```
@WebServlet("/response")
public class ResponseServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException {
        try (PrintWriter out = response.getWriter()) {

            // then write the data of the response
            String username = request.getParameter("username");
            if (username != null && username.length()> 0) {
                out.println("<h2>Hello, " + username + "!</h2>");
            }
        }
    }
    ...
```

6.4.3 Running the hello2 Example

You can use either NetBeans IDE or Maven to build, package, deploy, and run the `hello2` example.

6.4.3.1 To Run the hello2 Example Using NetBeans IDE

1. Start GlassFish Server as described in Section 2.2.1, "To Start GlassFish Server Using NetBeans IDE," if you have not already done so.
2. From the **File** menu, choose **Open Project**.
3. In the Open Project dialog box, navigate to:

   ```
   tut-install/examples/web/servlet
   ```

4. Select the `hello2` folder.
5. Click **Open Project**.
6. In the **Projects** tab, right-click the `hello2` project and select **Build** to package and deploy the project.
7. In a web browser, open the following URL:

   ```
   http://localhost:8080/hello2/greeting
   ```

The URL specifies the context root, followed by the URL pattern.

The application looks much like the `hello1` application. The major difference is that after you click **Submit** the response appears below the greeting, not on a separate page.

6.4.3.2 To Run the hello2 Example Using Maven

1. Start GlassFish Server as described in Section 2.2.3, "To Start GlassFish Server Using the Command Line," if you have not already done so.
2. In a terminal window, go to:

   ```
   tut-install/examples/web/servlet/hello2/
   ```

3. Enter the following command:

   ```
   mvn install
   ```

   This target builds the WAR file, copies it to the *tut-install*`/examples/web/hello2/target/` directory, and deploys it.
4. In a web browser, open the following URL:

   ```
   http://localhost:8080/hello2/greeting
   ```

   The URL specifies the context root, followed by the URL pattern.

   The application looks much like the `hello1` application. The major difference is that after you click **Submit** the response appears below the greeting, not on a separate page.

6.5 Configuring Web Applications

This section describes the following tasks involved with configuring web applications:

- Setting context parameters
- Declaring welcome files
- Mapping errors to error screens
- Declaring resource references

6.5.1 Setting Context Parameters

The web components in a web module share an object that represents their application context. You can pass context parameters to the context, or you can pass initialization parameters to a servlet. Context parameters are available to the entire application. For information on initialization parameters, see Section 17.4, "Creating and Initializing a Servlet."

6.5.1.1 To Add a Context Parameter Using NetBeans IDE

These steps apply generally to web applications but do not apply specifically to the examples in this chapter.

1. Open the project.
2. Expand the project's node in the **Projects** tree.
3. Expand the **Web Pages** node and then the **WEB-INF** node.
4. Double-click `web.xml`.

   If the project does not have a `web.xml` file, create one by following the steps in Section 6.5.1.2, "To Create a web.xml File Using NetBeans IDE."
5. Click **General** at the top of the editor window.
6. Expand the **Context Parameters** node.
7. Click **Add**.
8. In the Add Context Parameter dialog box, in the **Parameter Name** field, enter the name that specifies the context object.
9. In the **Parameter Value** field, enter the parameter to pass to the context object.
10. Click **OK**.

6.5.1.2 To Create a web.xml File Using NetBeans IDE

1. From the **File** menu, choose **New File**.
2. In the New File wizard, select the **Web** category, then select **Standard Deployment Descriptor** under **File Types**.
3. Click **Next**.
4. Click **Finish**.

   A basic `web.xml` file appears in `web/WEB-INF/`.

6.5.2 Declaring Welcome Files

The **welcome files** mechanism allows you to specify a list of files that the web container can append to a request for a URL (called a valid partial request) that is not mapped to a web component. For example, suppose that you define a welcome file `welcome.html`. When a client requests a URL such as *host*:*port*/*webapp*/*directory*, where *directory* is not mapped to a servlet or XHTML page, the file *host*:*port*/*webapp*/*directory*/`welcome.html` is returned to the client.

If a web container receives a valid partial request, the web container examines the welcome file list, appends to the partial request each welcome file in the order specified, and checks whether a static resource or servlet in the WAR is mapped to that request URL. The web container then sends the request to the first resource that matches in the WAR.

If no welcome file is specified, GlassFish Server will use a file named `index.html` as the default welcome file. If there is no welcome file and no file named `index.html`, GlassFish Server returns a directory listing.

You specify welcome files in the `web.xml` file. The welcome file specification for the `hello1` example looks like this:

```
<welcome-file-list>
    <welcome-file>index.xhtml</welcome-file>
</welcome-file-list>
```

A specified welcome file must not have a leading or trailing slash (`/`).

The `hello2` example does not specify a welcome file, because the URL request is mapped to the `GreetingServlet` web component through the URL pattern `/greeting`.

6.5.3 Mapping Errors to Error Screens

When an error occurs during execution of a web application, you can have the application display a specific error screen according to the type of error. In particular, you can specify a mapping between the status code returned in an HTTP response or a Java programming language exception returned by any web component and any type of error screen.

You can have multiple `error-page` elements in your deployment descriptor. Each element identifies a different error that causes an error page to open. This error page can be the same for any number of `error-page` elements.

6.5.3.1 To Set Up Error Mapping Using NetBeans IDE

These steps apply generally to web applications but do not apply specifically to the examples in this chapter.

1. Open the project.
2. Expand the project's node in the **Projects** tab.

3. Expand the **Web Pages** node and then the **WEB-INF** node.
4. Double-click `web.xml`.

   If the project does not have a `web.xml` file, create one by following the steps in Section 6.5.1.2, "To Create a web.xml File Using NetBeans IDE."
5. Click **Pages** at the top of the editor window.
6. Expand the **Error Pages** node.
7. Click **Add**.
8. In the Add Error Page dialog box, click **Browse** to locate the page that you want to act as the error page.
9. Specify either an error code or an exception type.
   - To specify an error code, in the **Error Code** field enter the HTTP status code that will cause the error page to be opened, or leave the field blank to include all error codes.
   - To specify an exception type, in the **Exception Type** field enter the exception that will cause the error page to load. To specify all throwable errors and exceptions, enter `java.lang.Throwable`.
10. Click **OK**.

6.5.4 Declaring Resource References

If your web component uses such objects as enterprise beans, data sources, or web services, you use Java EE annotations to inject these resources into your application. Annotations eliminate a lot of the boilerplate lookup code and configuration elements that previous versions of Java EE required.

Although resource injection using annotations can be more convenient for the developer, there are some restrictions on using it in web applications. First, you can inject resources only into container-managed objects, because a container must have control over the creation of a component so that it can perform the injection into a component. As a result, you cannot inject resources into such objects as simple JavaBeans components. However, managed beans are managed by the container; therefore, they can accept resource injections.

Components that can accept resource injections are listed in Table 6–1.

This section explains how to use a couple of the annotations supported by a web container to inject resources. Chapter 9, "Running the Persistence Examples," in *The Java EE 7 Tutorial, Volume 2*, explains how web applications use annotations supported by the Java Persistence API. Chapter 19, "Getting Started Securing Web

Applications," in *The Java EE 7 Tutorial, Volume 2*, explains how to use annotations to specify information about securing web applications. See Chapter 23, "Resource Adapters and Contracts," in *The Java EE 7 Tutorial, Volume 2*, for more information on resources.

Table 6–1 Web Components That Accept Resource Injections

Component	Interface/Class
Servlets	`javax.servlet.Servlet`
Servlet filters	`javax.servlet.ServletFilter`

Event listeners	`javax.servlet.ServletContextListener`
	`javax.servlet.ServletContextAttributeListener`
	`javax.servlet.ServletRequestListener`
	`javax.servlet.ServletRequestAttributeListener`
	`javax.servlet.http.HttpSessionListener`
	`javax.servlet.http.HttpSessionAttributeListener`
	`javax.servlet.http.HttpSessionBindingListener`

Managed beans	Plain Old Java Objects

6.5.4.1 Declaring a Reference to a Resource

The `@Resource` annotation is used to declare a reference to a resource, such as a data source, an enterprise bean, or an environment entry.

The `@Resource` annotation is specified on a class, a method, or a field. The container is responsible for injecting references to resources declared by the `@Resource` annotation and mapping it to the proper JNDI resources.

In the following example, the `@Resource` annotation is used to inject a data source into a component that needs to make a connection to the data source, as is done when using JDBC technology to access a relational database:

```
@Resource javax.sql.DataSource catalogDS;
public getProductsByCategory() {
    // get a connection and execute the query
    Connection conn = catalogDS.getConnection();
    ...
}
```

The container injects this data source prior to the component's being made available to the application. The data source JNDI mapping is inferred from the field name, `catalogDS`, and the type, `javax.sql.DataSource`.

If you have multiple resources that you need to inject into one component, you need to use the `@Resources` annotation to contain them, as shown by the following example:

```
@Resources ({
    @Resource(name="myDB" type=javax.sql.DataSource.class),
    @Resource(name="myMQ" type=javax.jms.ConnectionFactory.class)
})
```

The web application examples in this tutorial use the Java Persistence API to access relational databases. This API does not require you to explicitly create a connection to a data source. Therefore, the examples do not use the `@Resource` annotation to inject a data source. However, this API supports the `@PersistenceUnit` and `@PersistenceContext` annotations for injecting `EntityManagerFactory` and `EntityManager` instances, respectively. Chapter 9, "Running the Persistence Examples," in *The Java EE 7 Tutorial, Volume 2*, describes these annotations and the use of the Java Persistence API in web applications.

6.5.4.2 Declaring a Reference to a Web Service

The `@WebServiceRef` annotation provides a reference to a web service. The following example shows uses the `@WebServiceRef` annotation to declare a reference to a web service. `WebServiceRef` uses the `wsdlLocation` element to specify the URI of the deployed service's WSDL file:

```
...
import javax.xml.ws.WebServiceRef;
...
public class ResponseServlet extends HTTPServlet {
@WebServiceRef(wsdlLocation="http://localhost:8080/helloservice/hello?wsdl")
static HelloService service;
```

6.6 Further Information about Web Applications

For more information on web applications, see

- JavaServer Faces 2.2 specification:

  `http://jcp.org/en/jsr/detail?id=344`

- Java Servlet 3.1 specification:

  `http://jcp.org/en/jsr/detail?id=340`

This page intentionally left blank

Index

Symbols

`@Alternative` annotation, 494
`@ApplicationScoped` annotation, 74, 307, 471
`@Consumes` annotation, 562, 572
`@Context` annotation, 598
`@ConversationScoped` annotation, 471
`@CookieParam` annotation, 598
`@Decorator` annotation, 506
`@Delegate` annotation, 506
`@DELETE` annotation, 562, 567
`@Dependent` annotation, 307, 471
`@Disposes` annotation, 499
`@FlowScoped` annotation, 307
`@FormParam` annotation, 598, 599
`@GET` annotation, 561, 567
`@GroupSequence` annotation, 458
`@HEAD` annotation, 562
`@HeaderParam` annotation, 598
`@Inject` annotation, 470, 500
`@ManagedBean` annotation, 115, 306
`@MatrixParam` annotation, 598
`@MultipartConfig` annotation, 370
`@Named` annotation, 473
`@Observes` annotation, 501
`@OnClose` annotation, 400
`@OnError` annotation, 400
`@OnMessage` annotation, 400
`@OnOpen` annotation, 400
`@Path` annotation, 561, 564
`@PathParam` annotation, 562, 573, 598
`@POST` annotation, 562, 567
`@Produces` annotation, 474, 497, 562, 570
`@Provider` annotation, 563
`@PUT` annotation, 562, 567
`@Qualifier` annotation, 469
`@QueryParam` annotation, 562, 573, 598, 599
`@RequestScoped` annotation, 74, 307, 471
`@Resource` annotation, 50, 54, 85, 500
`@ResourceDependency` annotation, 192, 243
`@SessionScoped` annotation, 74, 307, 471
`@WebFilter` annotation, 360
`@WebInitParam` annotation, 356, 360
`@WebListener` annotation, 353
`@WebService` annotation, 546
`@WebServiceRef` annotation, 86
`@WebServlet` annotation, 78, 356

A

action events, 97, 102, 173, 281
  `ActionEvent` class, 281, 282
  `actionListener` attribute, 172, 210, 211, 265
  `ActionListener` implementation, 281, 282
  `ActionListener` interface, 205
  `actionListener` tag, 193, 205, 260
  `processAction(ActionEvent)` method, 282
  referencing methods that handle action events, 211, 228
  writing a managed bean method to handle action events, 228
action method, 101
Administration Console, 34
  starting, 40
Ajax
  error handling, 238
  `event` attribute of `f:ajax` tag, 236
  example, 243
  `execute` attribute of `f:ajax` tag, 236
  grouping components, 240
  `immediate` attribute of `f:ajax` tag, 237
  `listener` attribute of `f:ajax` tag, 237

Ajax (cont.)
  loading JavaScript resource library, 241
  monitoring events, 238
  onerror attribute of f:ajax tag, 238
  onevent attribute of f:ajax tag, 238
  overview, 232
  receiving responses, 239
  render attribute of f:ajax tag, 239
  request lifecycle, 240
  sending requests, 236
  using JavaScript API directly, 242
  using with Facelets, 233
  using with JavaServer Faces technology, 231
alternatives
  CDI, 494
  example, 511
annotations, 3
  JAX-RS, 561, 598
appclient tool, 34
applet container, 15
applets, 9, 10
application client container, 15
application clients, 8
applications
  dynamic reloading, 76
  JavaServer Faces, 88
  packaging, 57
  undeploying, 77
asadmin tool, 34
attributes referencing managed bean methods, 209
  action attribute, 210
  actionListener attribute, 210, 211
  validator attribute, 210, 211
  valueChangeListener attribute, 210, 212
auto commit, 26

B

Batch Applications for the Java Platform, 30
Bean Validation, 27, 447
  advanced, 455
  constraint violations, 453
  constructors, 451
  custom constraints, 455
  empty strings, 450
  exceptions, 453
  inheritance, 458
  JavaServer Faces applications, 447
  localization, 457
  messages, 457
  methods, 451, 453, 458
  null strings, 450
  ordering, 458
  parameters, 451, 452, 453
  resource bundles, 457
  using f:validateBean tag, 207
beans, in CDI, 467
beans.xml file, 475
bookmarkable URLs
  component tags, 187
  example, 189
  view parameters, 188
BufferedReader class, 357
bundles. *See* resource bundles

C

capture-schema tool, 34
CDI. *See* Contexts and Dependency Injection for Java EE (CDI)
character encodings, 442
character sets, 442
component binding, 216, 217, 298, 302
  binding attribute, 216, 298, 302
component properties. *See* managed bean properties
component rendering model, 93, 95
  decode method, 107, 274, 283, 289
  decoding, 261, 269
  delegated implementation, 261
  direct implementation, 261
  encode method, 290
  encodeBegin method, 272
  encodeChildren method, 272
  encodeEnd method, 272, 279
  encoding, 261, 269
  HTML render kit, 284, 340
  render kit, 96
  Renderer class, 96
  Renderer implementation, 340
  RenderKit class, 96
  RenderKit implementation, 340

component tag attributes
  action attribute, 172, 226, 265
  actionListener attribute, 172, 210, 228, 265
  alt attribute, 265
  binding attribute, 161, 164, 216, 298, 302
  columns attribute, 176
  converter attribute, 167, 199, 291
  for attribute, 170, 186
  id attribute, 161, 162
  immediate attribute, 161, 162, 265
  redisplay attribute, 170
  rendered attribute, 161, 163, 303
  style attribute, 161, 164, 186
  styleClass attribute, 161, 164
  validator attribute, 168, 229
  value attribute, 161, 164, 216, 266, 298, 300
  valueChangeListener attribute, 168, 212, 230
  var attribute, 441
component tags, 96, 98, 216
  attributes. *See* component tag attributes
  body tag, 165
  bookmarkable URLs, 187
  button tag, 187
  column tag, 159
  commandButton tag, 159, 173
  commandLink tag, 159, 173
  dataTable tag, 159, 182, 219
  form tag, 159, 165
  graphicImage tag, 159, 174, 265
  head tag, 165
  inputHidden tag, 159, 167
  inputSecret tag, 159, 167, 170
  inputText tag, 159, 167, 169
  inputTextarea tag, 160, 167
  link tag, 187
  message tag, 160, 186
  messages tag, 160, 186
  outputFormat tag, 160, 168, 171
  outputLabel tag, 160, 168, 170
  outputLink tag, 160, 168, 171
  outputScript tag, 190
  outputStylesheet tag, 190
  outputText tag, 160, 168, 169, 173, 220
  panelGrid tag, 160, 174
  panelGroup tag, 160, 174
  resource relocation, 190
  selectBooleanCheckbox tag, 160, 177, 220
  selectItems tag, 223
  selectManyCheckbox tag, 160, 179, 221
  selectManyListbox tag, 160, 179
  selectManyMenu tag, 160, 179
  selectOneListbox tag, 160, 178
  selectOneMenu tag, 161, 178, 222, 223
  selectOneRadio tag, 161, 178
components
  boxes, 160
  buttons, 159, 173
  check boxes, 160, 177
  data grids, 159
  fields, 159
  hidden fields, 159
  hyperlinks, 173
  images, 174
  Java EE, 8
  labels, 160
  links, 159
  menus, 160, 161, 178, 179
  options, 161
  password fields, 159
  radio buttons, 161
  table columns, 159
  tables, 160, 174, 182
  text areas, 160
composite components
  advanced features, 249
  attributes, 249
  default attribute, 250
  example, 251
  f:validateBean tag, 251
  f:validateRegex tag, 251
  f:validateRequired tag, 251
  Facelets, 124
  invoking managed beans, 251
  method-signature attribute, 250
  name attribute, 250
  required attribute, 250
  type attribute, 250
  validating values, 251
Concurrency Utilities for Java EE, 30
conditional HTTP requests in JAX-RS, 608
configuring JavaServer Faces applications
  Application class, 309
  application configuration resource files, 307
  error message registration, 293

configuring JavaServer Faces applications (cont.)
  faces-config.xml files, 339
  including the classes, pages, and other resources, 348
  javax.faces.application.CONFIG_FILES context parameter, 308
  registering custom converters, 337
  registering custom renderers, 340
  registering custom UI components, 342
  registering custom validators, 336
  registering messages, 332
  specifying a path to an application configuration resource file, 346
  specifying where UI component state is saved, 278, 347
  value binding, 300
  See also configuring managed beans, configuring navigation rules
configuring managed beans, 266, 306, 323
configuring navigation rules, 100, 337
  from-action element, 339
  from-view-id element, 339
  navigation-case element, 339
  navigation-rule element, 339
  to-view-id element, 339
connection pooling, 50
connectors. *See* Java EE Connector architecture
constructors, static, 451
containers, 13
  application client, 15
  configurable services, 13
  nonconfigurable services, 14
  services, 13
  See also EJB container, web container
context parameters, 73
  specifying, 82
Contexts and Dependency Injection for Java EE (CDI), 27
  advanced topics, 493
  alternatives, 494
  basic concepts, 463
  beans, 467
  configuring applications, 475
  converting managed beans to JAX-RS root resource classes, 607
  decorators, 506
  disposer methods, 499
  EL, 473
  events, 501
  examples, 479, 511
  Facelets pages, 474
  injectable objects, 468
  injecting beans, 470
  integrating with JAX-RS, 606
  interceptors, 504
  managed beans, 468
  observer methods, 501
  overview, 466
  producer fields, 497
  producer methods, 474, 497
  qualifiers, 469
  scopes, 471
  setter and getter methods, 473
  specialization, 496
  stereotypes, 507
conversion model, 93, 96
  converter attribute, 167, 199, 291
  Converter implementations, 97, 198, 291
  Converter interface, 288
  converterId attribute, 199
  converting data between model and presentation, 96
  javax.faces.convert package, 198
  model view, 288, 289
  presentation view, 288, 289
  See also converters, converter tags
Converter implementation classes
  BigDecimalConverter class, 198
  BigIntegerConverter class, 198
  BooleanConverter class, 198
  ByteConverter class, 198
  CharacterConverter class, 198
  DateTimeConverter class, 198, 199, 200
  DoubleConverter class, 198
  EnumConverter class, 198
  FloatConverter class, 198
  IntegerConverter class, 198
  LongConverter class, 198
  NumberConverter class, 198, 199, 202
  ShortConverter class, 198
converter tags
  convertDateTime tag, 200
  convertDateTime tag attributes, 201
  converter tag, 200, 291

convertNumber tag, 199, 202
convertNumber tag attributes, 203
converters, 93, 107
custom converters, 97, 291
See also standard converters
converting data. *See* conversion model
cookie parameters, JAX-RS, 575
custom converters
binding to managed bean properties, 303
creating, 287
getAsObject method, 289
getAsObject(FacesContext, UIComponent, String) method, 288
getAsString method, 290
getAsString(FacesContext, UIComponent, Object) method, 289
registering. *See* registering custom converters
using, 291
custom objects
custom converters, 291
using, 285
using custom components, renderers and tags together, 261
See also custom renderers, custom tags, custom UI components, custom validators
custom renderers
creating the Renderer class, 278
determining necessity of, 261
performing decoding, 274
performing encoding, 272
registering with a render kit, 340
custom tags, 99, 261
getRendererType method, 280
identifying the renderer type, 278
specifying, 296
tag library descriptor, 268
custom UI components
creating, 257
creating component classes, 268
custom objects, 285
delegating rendering, 278
determining necessity of, 259
handling events emitted by, 283
queueEvent method, 274
registering. *See* registering custom UI components
restoreState(FacesContext, Object) method, 277
saveState(FacesContext) method, 277
saving state, 277
specifying where state is saved, 347
steps for creating, 268
custom validators, 292
binding to managed bean properties, 303
custom validator tags, 296
implementing the Validator interface, 294
registering, 336
using, 297
validate method, 229, 294
Validator implementation, 226, 294, 296
Validator interface, 293
validator tag, 293, 296

D

data sources, 50
databases, EIS tier, 6
DataSource interface, 50
debugging Java EE applications, 44
decorators
CDI, 506
example, 535
dependency injection, 53, 54
Dependency Injection for Java (JSR 330), 27, 463
deployer roles, 19
deployment descriptors, 57
enterprise bean, 60
Java EE, 58
runtime, 58, 62
web application, 62, 69, 343
destroy method, 367
development roles, 17
application assemblers, 19
application client developers, 19
application component providers, 18
application deployers and administrators, 19
enterprise bean developers, 18
Java EE product providers, 18
tool providers, 18
web component developers, 18
disposer methods, CDI, 499
document roots, 61
doFilter method, 359, 361, 362

doGet method, 356
domains, 39
doPost method, 356
downloading GlassFish Server, 36

E

eager attribute, managed beans, 309
EAR files, 57
EIS tier, 12
EJB container, 15
ejb-jar.xml file, 60
EL, 71, 141
  CDI managed beans, 473
  composite expressions, 148
  deferred evaluation expressions, 142
  expression examples, 154
  immediate evaluation expressions, 142
  literals, 146
  lvalue expressions, 144
  managed beans, 215
  method expressions, 149
  method-binding expressions, 101
  operators, 153
  overview, 141
  parameterized method calls, 147
  reserved words, 154
  rvalue expressions, 144
  type conversion during expression evaluation, 149
  value expressions, 144
  See also method binding
endpoints, JAX-WS, 547
enterprise applications, 3
enterprise beans, 11, 23
  converting to JAX-RS root resource classes, 606
  implementor of business logic, 11
  integrating with JAX-RS, 606
  JAX-RS resources, 579
  packaging, 59
  singletons, 580
Enterprise Information Systems. *See* EIS tier
entity providers, JAX-RS, 568
event and listener model, 93, 97
  binding listeners to managed bean properties, 303
  Event class, 97
  event handlers, 107, 268
  event listeners, 107, 108, 109
  handling events of custom UI components, 283
  implementing event listeners, 281
  Listener class, 97
  listener class, 226
  queueEvent method, 274
  ValueChangeEvent class, 211
  See also action events, value-change events
events
  CDI, 501
  example, 527
examples, 35
  Ajax, 243
  bookmarkable URLs, 189
  building, 41
  CDI, 479, 511
  composite components, 251
  directory structure, 41
  file upload using servlets, 383
  HTML5-friendly markup, 136
  Java API for JSON Processing, 432, 434
  Java API for WebSocket, 408, 413
  JAX-RS, 578, 618
  JAX-WS, 546
  required software, 35
  resource library contracts, 130
  servlets, 77, 381
exceptions, mapping to error screens, 83
Expression Language. *See* EL

F

Facelets, 111
  composite components, 124
  configuring applications, 119
  f:ajax tag, 234
  features, 112
  resources, 127
  templating, 121
  using Ajax with, 233
  XHTML pages, 117
  See also EL
Facelets applications
  developing, 115
  lifecycle, 114
  using JavaScript in, 242

Faces Flows, using, 311
faces-config.xml file, 307
FacesContext class, 105, 287
  Apply Request Values phase, 107
  custom converters, 289
  performing encoding, 273
  Process Validations phase, 108
  Update Model Values phase, 109
  validation methods, 229
  Validator interface, 294
FacesServlet class, 345
field-based injection, 54
filter chains, 359, 362
Filter interface, 359
filters, 359
  defining, 359
  mapping to web components, 361
  mapping to web resources, 361
  overriding request methods, 361
  overriding response methods, 361
  response wrappers, 361
form parameters, JAX-RS, 576, 599
forward method, 364

G

GenericServlet interface, 352
getConnection method, 50
getPart method, 371
getParts method, 371
getRequestDispatcher method, 363
getServletContext method, 365
getSession method, 365
GlassFish Server
  downloading, 36
  enabling debugging, 45
  installation tips, 36
  server log, 44
  starting, 39
  stopping, 40
  tools, 33
glassfish-resources.xml file, 51

H

handling events. *See* event and listener model
header parameters, JAX-RS, 575
HTML5-friendly markup, 132
  example, 136
  pass-through attributes, 135
  pass-through elements, 133
HTTP, 545
HTTP cookies, 593
HTTP methods, 567
HTTP request and response entity bodies, 568
  supported types, 569
HTTP request URLs, 357
  query strings, 358
  request paths, 357
HTTP requests, 357, 567
  See also requests
HTTP responses, 359
  status codes, 83
  See also responses
HttpServlet interface, 352
HttpServletRequest interface, 357
HttpServletResponse interface, 359
HttpSession interface, 365

I

implicit navigation, 90, 100
implicit objects, 302
  binding component values to, 301
include method, 364
init method, 356
InitialContext interface, 31
initialization parameters, 356
initializing properties with the managed-property element
  initializing Array and List properties, 329
  initializing managed-bean properties, 330
  initializing Map properties, 328
  initializing maps and lists, 332
  referencing a context initialization parameter, 327
initParams attribute, 356
injectable objects, CDI, 468
interceptor binding types, 504
interceptors
  CDI, 504
  example (CDI), 527
internationalization, 437

internationalizing JavaServer Faces applications
  FacesContext.getLocale method, 201
  loadBundle tag, 441
  using the FacesMessage class to create a message, 333
invalidate method, 366
ISO 8859 character encoding, 442

J

JAAS, 33
JACC, 29
JAF, 31
JAR files, 57
JASPIC, 29
Java API for JavaBean Validation
  See Bean Validation
Java API for JSON Processing, 29, 421
  examples, 432, 434
  introduction, 421
  Java EE platform, 423
  object model API, 425
  streaming API, 429
Java API for RESTful Web Services. *See* JAX-RS
Java API for WebSocket, 29, 395
  annotated endpoints, 399
  configuring endpoints, 407
  creating applications, 397
  endpoints, 396
  error handling, 407
  examples, 408, 413
  introduction, 396
  maintaining client state, 402
  path parameters, 406
  programmatic endpoints, 398
  receiving messages, 402
  sending messages, 400
  using decoders, 404
  using encoders, 403
Java API for XML Binding (JAXB), 32
  using with JAX-RS, 611
Java API for XML Processing (JAXP), 32
Java API for XML Web Services. *See* JAX-WS
Java Authentication and Authorization Service. *See* JAAS
Java Authentication Service Provider Interface for Containers (JASPIC), 29
Java Authorization Contract for Containers. *See* JACC
Java BluePrints, 41
Java Database Connectivity API. *See* JDBC API
Java DB, 34
  starting, 40
  stopping, 41
Java EE applications, 6
  debugging, 44
  tiers, 6
Java EE clients, 8
  See also application clients, web clients
Java EE components, 8
Java EE Connector architecture, 28
Java EE modules, 57, 58
  application client modules, 59
  EJB modules, 59
  resource adapter modules, 59
  See also web modules
Java EE platform, 6
  APIs, 20
  overview, 3
Java EE security model, 13
Java EE servers, 15
Java EE transaction model, 13
Java Message Service (JMS) API, 27
Java Naming and Directory Interface API. *See* JNDI
Java Persistence API, 26
Java Servlet technology, 24, 351
  See also servlets
Java Transaction API, 26
JavaBeans Activation Framework (JAF), 31
JavaBeans components, 9
JavaMail API, 28
JavaServer Faces application development
  bean property, 219
  Bean Validation, 447
  managed beans, 213
  web pages, 157
JavaServer Faces applications
  configuring, 305
  HTML tags, 159
  queueing messages, 229
JavaServer Faces core tag library, 158, 192
  actionListener tag, 193, 205, 260
  ajax tag, 234
  attribute tag, 194

convertDateTime tag, 193, 200
convertDateTime tag attributes, 201
converter tag, 193, 200, 291
converterId attribute, 199
convertNumber tag, 193, 199, 202
convertNumber tag attributes, 203
facet tag, 184, 193
loadBundle tag, 194
metadata tag, 188, 193
param tag, 171, 194
selectItem tag, 178, 180, 181, 193
selectItems tag, 178, 180, 181, 193
type attribute, 204
validateBean tag, 207
validateDoubleRange tag, 194, 207
validateLength tag, 194, 207
validateLongRange tag, 194, 207, 209
validateRegEx tag, 207
validateRegex tag, 209
validateRequired tag, 207
validator tag, 99, 194, 293
custom validator tags, 296
valueChangeListener tag, 193, 204
viewparam tag, 188
JavaServer Faces standard HTML render kit library, 96, 340
html_basic TLD, 284
See also component tags
JavaServer Faces standard UI components, 93, 257
UIComponent component, 290
JavaServer Faces tag libraries, 112
JavaServer Faces core tag library, 158, 192
JavaServer Faces HTML render kit tag library, 158
namespace directives, 158
JavaServer Faces Technology
Faces Flows, 311
resource library contracts, 128
JavaServer Faces technology, 10, 24, 87
advantages, 89
bookmarkable URLs, 188
component tags. *See* component tags
composite components, 249
FacesContext class. *See* FacesContext class
FacesServlet class, 345
features, 88
HTML5-friendly markup, 132
partial processing, 110
partial rendering, 110
relocatable resources, 128
using Ajax with, 231
Validator interface, 229
See also component rendering model
See also component tags
See also conversion model
See also event and listener model
See also Facelets
See also JavaServer Faces standard UI components
See also lifecycle of a JavaServer Faces application
See also UI component behavioral interfaces
See also UI component classes
See also validation model
JavaServer Pages Standard Tag Library (JSTL), 25
JavaServer Pages technology, 25
javax.servlet package, 352
javax.servlet.http package, 352
JAXB, 32
using with JAX-RS, 611
JAXP, 32
JAX-RS, 26, 559
accessing XML documents, 611
advanced features, 597
annotations, 598
application overview, 563
asynchronous invocations, 595
clients, 585
conditional HTTP requests, 608
configuring, 577
converting CDI managed beans to root resource classes, 607
converting enterprise beans to root resource classes, 606
cookies, 593
entity providers, 568
examples, 578, 618
extracting Java type of request or response, 600
filters, 593
form parameters, 599
HTTP headers, 592
integrating with CDI, 606
integrating with EJB technology, 606
introduction, 542

JAX-RS (cont.)
  invocations, 587
  JSON, 616
  path parameters, 587, 598
  path templates, 564
  query parameters, 599
  reference implementation, 559
  request headers, 598
  request method designators, 561, 567
  resource class methods, 604
  resource classes, 561
  resource methods, 561
  runtime content negotiation, 609
  runtime resource resolution, 604
  static content negotiation, 609
  subresource locators, 605
  subresource methods, 605
  subresources, 604
  URI, 598
  using with JAXB, 611
JAX-RS Client API, 581, 585
JAX-RS clients, 581
JAX-WS, 32
  defined, 545
  endpoints, 547
  examples, 546
  introduction, 542
  service endpoint interfaces, 546
  specification, 558
JDBC API, 30, 50
JMS, 27
JNDI, 31, 49
  data source naming subcontexts, 31
  enterprise bean naming subcontexts, 31
  environment naming contexts, 31
  naming contexts, 31
  naming environments, 31
  naming subcontexts, 31
`jsf.js` file, 241
JSON
  generating and parsing data, 423
  JAX-RS, 431, 616
  syntax, 422
  uses, 422
  See also Java API for JSON Processing
JSR 339. *See* JAX-RS
JSR 346. *See* Contexts and Dependency Injection for Java EE (CDI)
JSTL, 25
JTA, 26

L

lifecycle of a JavaServer Faces application, 91, 103
  action and value-change event processing, 98
  Apply Request Values phase, 107, 274
  custom converters, 289, 290
  `getRendererType` method (Render Response phase), 280
  `immediate` attribute, 265
  Invoke Application phase, 109
  performing encoding (Render Response phase), 272
  Process Validations phase, 108
  Render Response phase, 109
  `renderResponse` method, 105, 107, 108, 109
  `responseComplete` method, 106, 108, 109
  Restore View phase, 107
  saving state, 277
  Update Model Values phase, 108
  `updateModels` method, 109
  `Validator` interface, 295
  views, 107
listener classes, 353
  defining, 353
listener interfaces, 353
localization, 437
  Bean Validation, 457
log, server, 44

M

managed bean creation facility, 323
managed bean declarations, 266
  `key-class` element, 329
  `list-entries` element, 326
  `managed-bean` element, 324, 331
  `managed-bean-name` element, 325
  `managed-property` element, 326
  `map-entries` element, 326, 328
  `map-entry` element, 328
  `null-value` elements, 326
  `value` element, 326

managed bean methods
  attributes. *See* attributes referencing managed bean methods
  referencing. *See* referencing managed bean methods
  writing. *See* writing managed bean methods
managed bean properties, 199, 214, 216, 298
  bound to component instances, 224
  `UIData` properties, 219
  `UIInput` and `UIOutput` properties, 218
  `UISelectBoolean` properties, 220
  `UISelectItem` properties, 223
  `UISelectItems` properties, 223
  `UISelectMany` properties, 221
  `UISelectOne` properties, 222
  writing, 216
managed beans, 88
  composite components, 251
  configuring in JavaServer Faces technology, 306
  conversion model, 96
  custom component alternative, 260
  defined for CDI, 468
  developing, 115
  event and listener model, 98
  JavaServer Faces technology, 213
  loading JavaScript, 243
  method binding, 168
  properties. *See* managed bean properties
  See also value binding
Managed Beans specification, 26, 463
mapping URLs, 577
matrix parameters, JAX-RS, 576
Maven tool, 38
`MessageBodyReader` interface, 568
`MessageBodyWriter` interface, 568
message-driven beans, 23
messages
  `MessageFormat` pattern, 171, 194
  `outputFormat` tag, 171
  `param` tag, 171, 194
  parameter substitution tags, 194
  queueing messages, 229, 332
  using the `FacesMessage` class to create a message, 333
method binding
  method expressions, 275
  method-binding expressions, 101, 339
method expressions, 98, 210
method-based injection, 54
methods, static, 451

N

naming contexts, 31
naming environments, 31
navigation
  configuring, 100, 337
  implicit, 100
navigation model, 100
  `action` attribute, 172, 210, 265
  action methods, 226, 338
  `ActionEvent` class, 211
  configuring navigation rules, 337
  logical outcome, 226, 338
  `NavigationHandler` class, 102
  referencing methods that perform navigation, 210, 226
  writing a managed bean method to perform navigation processing, 226
NetBeans IDE, 37

O

observer methods, CDI, 501

P

`package-appclient` tool, 34
packaging applications, 57
path parameters, JAX-RS, 575, 598
path templates, JAX-RS, 564
POJOs, 4
`PrintWriter` class, 358
producer fields
  CDI, 497
  example, 520
producer methods
  CDI, 474, 497
  example, 517
proxies, 545

Q

qualifiers, using in CDI, 469
query parameters, JAX-RS, 573, 599

R

RAR files, packaging, 63
referencing managed bean methods, 209
  for handling action events, 211, 228
  for handling value-change events, 212
  for performing navigation, 210, 226
  for performing validation, 211, 229
registering custom converters, 337
  `converter` element, 337
registering custom renderers, 340
  `renderer` element, 340
  `render-kit` element, 340
registering custom UI components, 268, 342
  `component` element, 342
registering custom validators, 336
  `validator` element, 336
registering messages, 332
  `resource-bundle` element, 333
relocatable resources, 128
request headers, JAX-RS, 598
request method designators, JAX-RS, 561, 567
`Request` objects, JAX-RS, 610
request parameters, JAX-RS, 573
`RequestDispatcher` interface, 363
requests, 357
  customizing, 361
  getting information from, 357
  retrieving a locale, 439
  See also HTTP requests
resource adapters, 28
  packaging, 63
resource bundles, 438
  Bean Validation, 457
resource classes, JAX-RS, 561
resource injection, 53
resource library contracts, 128
  example, 130
resource methods, JAX-RS, 561
resources
  creating, 49
  definition, 49
  injecting, 53
  relocatable, 128
  See also data sources
`ResponseBuilder` class, 568
responses, 358
  buffering output, 358
  customizing, 361
  setting headers, 356
  See also HTTP responses
RESTful web services, 26, 559
  defined, 559
root resource classes, 561

S

SAAJ, 33
`schemagen` tool, 34
scopes
  CDI, 471
  introduction, 74
  JavaServer Faces technology, 307
  servlets, 354
server log, 44
service methods, servlets, 356
`Servlet` interface, 352
`ServletContext` interface, 365
`ServletInputStream` class, 357
`ServletOutputStream` class, 358
`ServletRequest` interface, 357
`ServletResponse` interface, 358
servlets, 10, 352
  asynchronous processing, 372
  binary data, 357, 358
  character data, 357, 358
  creating, 356
  examples, 77, 381
  finalizing, 367
  initializing, 356
  lifecycle, 352
  lifecycle events, 353
  nonblocking I/O, 375
  scope objects, 354
  service methods, 356, 368, 369
  specifying initialization parameters, 356
  tracking service requests, 368
  uploading files with, 370, 421
session beans, 23

sessions, 365
  associating attributes, 366
  associating with user, 367
  invalidating, 366
  notifying objects associated with, 366
SingleThreadModel interface, 355
SOAP, 541, 545, 558
SOAP messages, 16, 33
SOAP with Attachments API for Java (SAAJ), 33
specialization, CDI, 496
SQL, 30
standard converters, 97
  converter tags, 200
  NumberConverter class, 199
  using, 198
standard validators, 99
  using, 207
stereotypes, CDI, 507
subresources, JAX-RS, 604
substitution parameters, defining. *See* messages, param tag

T

templating
  Facelets, 121

U

UI component behavioral interfaces, 94
  ActionSource interface, 94, 98, 269, 281
  ActionSource2 interface, 94, 269
  ClientBehaviorHolder interface, 95
  ConvertibleValueHolder interface, 95
  EditableValueHolder interface, 95, 269
  NamingContainer interface, 95, 269
  StateHolder interface, 95, 270, 277
  SystemEventListenerHolder interface, 95
  ValueHolder interface, 95, 270
UI component classes, 93, 95, 259
  javax.faces.component package, 269
  UIColumn class, 93
  UICommand class, 93, 96
  UIComponent class, 93, 96
  UIComponentBase class, 93, 269, 272
  UIData class, 93, 219
  UIForm class, 93
  UIGraphic class, 94
  UIInput and UIOutput classes, 218
  UIInput class, 94, 98
  UIMessage class, 94
  UIMessages class, 94
  UIOutcomeTarget class, 94
  UIOutput class, 94, 97
  UIPanel class, 94
  UIParameter class, 94
  UISelectBoolean class, 94, 220
  UISelectItem class, 94, 223
  UISelectItems class, 94, 223
  UISelectMany class, 94, 221
  UISelectOne class, 94, 96, 222
  UIViewRoot class, 94
  See also custom UI components
UnavailableException class, 356
undeploying modules and applications, 77
Unicode character set, 442
unified expression language. *See* EL
Uniform Resource Identifiers (URIs), 559
URI path parameters, JAX-RS, 575
URI path templates, JAX-RS, 564
URL paths, 77
URLs, mapping, 577
US-ASCII character set, 442
using pages, 126
UTF-8 character encoding, 442

V

validating input. *See* Bean Validation, validation model
validation, 447
  customizing, 455
  groups, 457
  localization, 457
  messages, 457
  ordering, 458
validation model, 93, 99
  referencing a method that performs validation, 211
  validator attribute, 168, 210, 211, 229
  Validator implementation, 99, 297
  Validator interface, 99, 226, 229
    custom validator tags, 296
    implementing, 294

validation model (cont.)
  writing a managed bean method to perform validation, 229
  See also validators
`Validator` implementation classes, 99, 207
  `BeanValidator` class, 207
  `DoubleRangeValidator` class, 194, 207
  `LengthValidator` class, 194, 207
  `LongRangeValidator` class, 194, 207, 209
  `RegexValidator` class, 207, 209
  `RequiredValidator` class, 207
validator tags
  composite components, 251
  `validateBean` tag, 207
  `validateDoubleRange` tag, 207
  `validateLength` tag, 207
  `validateLongRange` tag, 207, 209
  `validateRegex` tag, 207, 209
  `validateRequired` tag, 207
  `validator` tag, 99, 296
validators, 93, 107
  custom validators, 194, 297
  default, 335
  registering, 208
  standard, 207
value binding
  acceptable types of component values, 217
  component instances to bean properties. *See* component binding
  component values and instances to managed bean properties, 298
  component values to implicit objects, 301
  component values to managed bean properties, 300
  properties, 217
  `value` attribute, 216, 266, 298, 300
  value expressions, 219, 275, 302
  value-binding expressions, 299
value expressions, 216
  `ValueExpression` class, 216
value-change events, 98, 281
  `processValueChange(ValueChangeEvent)` method, 281
  `processValueChangeEvent` method, 230
  referencing methods that handle value-change events, 211
  `type` attribute, 204
  `ValueChangeEvent` class, 205, 281
  `valueChangeListener` attribute, 168, 210, 230
  `ValueChangeListener` class, 204, 230, 281
  `ValueChangeListener` implementation, 281
  `valueChangeListener` tag, 193, 204, 260
  writing a managed bean method to handle value-change events, 230
`Variant` class, JAX-RS, 610

W

W3C, 32, 545, 558
WAR files, 57
web applications, 61, 67
  configuring, 69
  deployment descriptors, 69
  document roots, 61
  establishing the locale, 439
  internationalizing and localizing, 437
  maintaining state across requests, 365
  parsing and formatting localized dates and numbers, 441
  presentation-oriented, 67
  providing localized messages, 438
  retrieving localized messages, 440
  service-oriented, 67
  setting the resource bundle, 439
  specifying context parameters, 82
  specifying welcome files, 82
web clients, 8, 67
web components, 10, 68
  applets bundled with, 10
  concurrent access to shared resources, 355
  forwarding to other web components, 364
  including other web resources, 364
  invoking other web resources, 363
  mapping exceptions to error screens, 83
  mapping filters to, 361
  scope objects, 354
  sharing information, 354
  types, 10
  utility classes bundled with, 10
  web context, 365
web container, 15, 68
  loading and initializing servlets, 352
  mapping URLs to web components, 77

web modules, 59, 61
  packaging and deploying, 74
  undeploying, 77
  viewing deployed, 75
web pages, XHTML, 112
web resources, 61
  Facelets, 127
  mapping filters to, 361
web services, 15
  declaring references to, 86
  examples, 546
  introduction, 541
  JAX-RS compared to JAX-WS, 541
WebSocket. *See* Java API for WebSocket
`web.xml` file, 62, 343
welcome files, 82
writing managed bean methods, 226
  for handling action events, 228
  for handling value-change events, 230
  for performing navigation, 226
  for performing validation, 229
writing managed bean properties
  converters, 225
  listeners, 225
  validators, 225
WSDL, 16, 541, 545, 558
`wsgen` tool, 34
`wsimport` tool, 34

X

`xjc` tool, 34
XML, 16, 545
XML schema
  mappings of Java classes to XML data types, 557
  mappings to Java data types, 556