The Java EE 6 Tutorial

Advanced Topics

Fourth Edition
Contents

Preface ..................................................................................................................................xxi

Part I Introduction ...........................................................................................................................1

1 Overview .................................................................................................................................3
  Java EE 6 Platform Highlights................................................................................................4
  Java EE Application Model ....................................................................................................5
  Distributed Multitiered Applications ...................................................................................6
    Security ..............................................................................................................................7
  Java EE Components ...........................................................................................................8
  Java EE Clients ..................................................................................................................8
  Web Components .............................................................................................................10
  Business Components ......................................................................................................11
  Enterprise Information System Tier ................................................................................12
  Java EE Containers ............................................................................................................13
    Container Services ..........................................................................................................13
    Container Types ............................................................................................................14
  Web Services Support .......................................................................................................15
    XML ...............................................................................................................................15
    SOAP Transport Protocol ............................................................................................16
    WSDL Standard Format ..............................................................................................16
  Java EE Application Assembly and Deployment .............................................................17
  Packaging Applications .....................................................................................................17
  Development Roles ...........................................................................................................19
    Java EE Product Provider ............................................................................................19
    Tool Provider ................................................................................................................19
    Application Component Provider .................................................................................20
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Assembler</td>
<td>20</td>
</tr>
<tr>
<td>Application Deployer and Administrator</td>
<td>21</td>
</tr>
<tr>
<td>Java EE 6 APIs</td>
<td>21</td>
</tr>
<tr>
<td>Enterprise JavaBeans Technology</td>
<td>25</td>
</tr>
<tr>
<td>Java Servlet Technology</td>
<td>26</td>
</tr>
<tr>
<td>JavaServer Faces Technology</td>
<td>26</td>
</tr>
<tr>
<td>JavaServer Pages Technology</td>
<td>27</td>
</tr>
<tr>
<td>JavaServer Pages Standard Tag Library</td>
<td>28</td>
</tr>
<tr>
<td>Java Persistence API</td>
<td>28</td>
</tr>
<tr>
<td>Java Transaction API</td>
<td>28</td>
</tr>
<tr>
<td>Java API for RESTful Web Services</td>
<td>29</td>
</tr>
<tr>
<td>Managed Beans</td>
<td>29</td>
</tr>
<tr>
<td>Contexts and Dependency Injection for the Java EE Platform (JSR 299)</td>
<td>29</td>
</tr>
<tr>
<td>Dependency Injection for Java (JSR 330)</td>
<td>29</td>
</tr>
<tr>
<td>Bean Validation</td>
<td>30</td>
</tr>
<tr>
<td>Java Message Service API</td>
<td>30</td>
</tr>
<tr>
<td>Java EE Connector Architecture</td>
<td>30</td>
</tr>
<tr>
<td>JavaMail API</td>
<td>31</td>
</tr>
<tr>
<td>Java Authorization Contract for Containers</td>
<td>31</td>
</tr>
<tr>
<td>Java Authentication Service Provider Interface for Containers</td>
<td>31</td>
</tr>
<tr>
<td>Java EE 6 APIs in the Java Platform, Standard Edition 6 and 7</td>
<td>32</td>
</tr>
<tr>
<td>Java Database Connectivity API</td>
<td>32</td>
</tr>
<tr>
<td>Java Naming and Directory Interface API</td>
<td>32</td>
</tr>
<tr>
<td>JavaBeans Activation Framework</td>
<td>33</td>
</tr>
<tr>
<td>Java API for XML Processing</td>
<td>33</td>
</tr>
<tr>
<td>Java Architecture for XML Binding</td>
<td>33</td>
</tr>
<tr>
<td>SOAP with Attachments API for Java</td>
<td>34</td>
</tr>
<tr>
<td>Java API for XML Web Services</td>
<td>34</td>
</tr>
<tr>
<td>Java Authentication and Authorization Service</td>
<td>34</td>
</tr>
<tr>
<td>GlassFish Server Tools</td>
<td>35</td>
</tr>
<tr>
<td>2 Using the Tutorial Examples</td>
<td>37</td>
</tr>
<tr>
<td>Required Software</td>
<td>37</td>
</tr>
<tr>
<td>Java Platform, Standard Edition</td>
<td>37</td>
</tr>
<tr>
<td>Java EE 6 Software Development Kit</td>
<td>38</td>
</tr>
</tbody>
</table>
Java EE 6 Tutorial Component .................................................................38
NetBeans IDE .........................................................................................39
Apache Ant ............................................................................................40
Starting and Stopping the GlassFish Server ...........................................41
▼ To Start the GlassFish Server Using NetBeans IDE ..........................42
Starting the Administration Console .....................................................42
▼ To Start the Administration Console Using NetBeans IDE ..........42
Starting and Stopping the Java DB Server ..............................................43
▼ To Start the Database Server Using NetBeans IDE .........................43
Building the Examples ..........................................................................43
Tutorial Example Directory Structure ..................................................44
Getting the Latest Updates to the Tutorial .........................................45
▼ To Update the Tutorial through the Update Center .......................45
Debugging Java EE Applications ..........................................................45
Using the Server Log .............................................................................45
Using a Debugger ................................................................................46

Part II The Web Tier ..................................................................................47

3 JavaServer Faces Technology: Advanced Concepts ..........................49
The Lifecycle of a JavaServer Faces Application .....................................50
  Overview of the JavaServer Faces Lifecycle .......................................50
  Restore View Phase ............................................................................53
  Apply Request Values Phase ............................................................53
  Process Validations Phase .................................................................54
  Update Model Values Phase ..............................................................55
  Invoke Application Phase .................................................................55
  Render Response Phase ....................................................................55
Partial Processing and Partial Rendering .............................................56
The Lifecycle of a Facelets Application ..................................................56
User Interface Component Model ..........................................................57
  User Interface Component Classes ..................................................58
  Component Rendering Model ..........................................................60
  Conversion Model .............................................................................61
  Event and Listener Model .................................................................61
## Contents

- **Validation Model** ................................................................. 63
- **Navigation Model** ............................................................... 64

### 4 Using Ajax with JavaServer Faces Technology .......................... 69
- **Overview of Ajax** ................................................................. 70
- **Using Ajax Functionality with JavaServer Faces Technology** ........ 70
- **Using Ajax with Facelets** ..................................................... 71
  - Using the `f:ajax` Tag......................................................... 71
- **Sending an Ajax Request** .................................................... 73
  - Using the `event` Attribute.................................................. 74
  - Using the `execute` Attribute............................................. 74
  - Using the `immediate` Attribute......................................... 75
  - Using the `listener` Attribute............................................. 75
- **Monitoring Events on the Client** .......................................... 75
- **Handling Errors** ................................................................ 76
- **Receiving an Ajax Response** ............................................... 77
- **Ajax Request Lifecycle** ........................................................ 78
- **Grouping of Components** .................................................. 78
- **Loading JavaScript as a Resource** ....................................... 79
  - Using JavaScript API in a Facelets Application...................... 80
  - Using the `@ResourceDependency` Annotation in a Bean Class........ 81
  - The `ajaxguessnumber` Example Application....................... 81
    - The `ajaxguessnumber` Source Files ............................... 81
    - Running the `ajaxguessnumber` Example .......................... 83
- **Further Information about Ajax in JavaServer Faces Technology** .... 85

### 5 Composite Components: Advanced Topics and Example ............... 87
- **Attributes of a Composite Component** ................................ 87
- **Invoking a Managed Bean** ................................................ 88
- **Validating Composite Component Values** ............................ 89
- **The compositecomponentlogin Example Application** .............. 89
  - The Composite Component File ....................................... 89
  - The Using Page ............................................................... 90
  - The Managed Bean ......................................................... 91
  - Running the `compositecomponentlogin` Example .................. 92
6 Creating Custom UI Components and Other Custom Objects ........................................... 95

Determining Whether You Need a Custom Component or Renderer ............................... 97
When to Use a Custom Component ................................................................................ 97
When to Use a Custom Renderer .................................................................................... 98
Component, Renderer, and Tag Combinations ................................................................ 99

Understanding the Image Map Example......................................................................... 100
Why Use JavaServer Faces Technology to Implement an Image Map? ..................... 100
Understanding the Rendered HTML ............................................................................... 101
Understanding the Facelets Page .................................................................................. 102
Configuring Model Data ............................................................................................... 103
Summary of the Image Map Application Classes ....................................................... 104

Steps for Creating a Custom Component ......................................................................... 105

Creating Custom Component Classes ............................................................................ 106
Specifying the Component Family .................................................................................. 109
Performing Encoding .................................................................................................... 109
Performing Decoding .................................................................................................... 111
Enabling Component Properties to Accept Expressions .............................................. 112
Saving and Restoring State ......................................................................................... 113

Delegating Rendering to a Renderer .............................................................................. 114
Creating the Renderer Class ......................................................................................... 115
Identifying the Renderer Type ....................................................................................... 117

Implementing an Event Listener .................................................................................... 117
Implementing Value-Change Listeners .......................................................................... 117
Implementing Action Listeners ..................................................................................... 118

Handling Events for Custom Components .................................................................... 119

Defining the Custom Component Tag in a Tag Library Descriptor ............................. 120

Using a Custom Component .......................................................................................... 121
Creating and Using a Custom Converter ....................................................................... 123
Creating a Custom Converter ....................................................................................... 123
Using a Custom Converter ......................................................................................... 126

Creating and Using a Custom Validator ......................................................................... 127
Implementing the Validator Interface ............................................................................ 127
Specifying a Custom Tag ............................................................................................. 131
Using a Custom Validator ............................................................................................ 132

Binding Component Values and Instances to Managed Bean Properties ................... 133
Binding a Component Value to a Property .................................................................... 134
### 7 Configuring JavaServer Faces Applications

- Binding a Component Value to an Implicit Object ................................................136
- Binding a Component Instance to a Bean Property ...............................................137
- Binding Converters, Listeners, and Validators to Managed Bean Properties ..........138

#### Using Annotations to Configure Managed Beans

- Using Managed Bean Scopes .................................................................................143
- Application Configuration Resource File ..............................................................144
- Ordering of Application Configuration Resource Files ........................................145

#### Configuring Managed Beans

- Using the `managed-bean` Element .....................................................................147
- Initializing Properties Using the `managed-property` Element .........................150
- Initializing Maps and Lists ..................................................................................155

#### Registering Application Messages

- Using `FacesMessage` to Create a Message .......................................................157
- Referencing Error Messages ...............................................................................157

#### Registering Default Validators

- Registering a Custom Validator ...........................................................................159

#### Registering a Custom Converter

- Registering a Custom Converter .........................................................................160

#### Configuring Navigation Rules

- ▼ To Configure a Navigation Rule .........................................................................163
- Implicit Navigation Rules .....................................................................................164

#### Registering a Custom Renderer with a Render Kit

- Registering a Custom Renderer with a Render Kit ...............................................165

#### Registering a Custom Component

- Registering a Custom Component .......................................................................167

#### Basic Requirements of a JavaServer Faces Application

- Configuring an Application with a Web Deployment Descriptor ............................169
- Configuring Project Stage ....................................................................................172
- Including the Classes, Pages, and Other Resources ............................................173

### 8 Uploading Files with Java Servlet Technology

- The `@MultipartConfig` Annotation .....................................................................175
- The `getParts` and `getPart` Methods ................................................................176
- The `fileupload` Example Application ................................................................177
- Architecture of the `fileupload` Example Application .......................................177
- Running the `fileupload` Example .....................................................................180
9 Internationalizing and Localizing Web Applications ............................................... 183
   Java Platform Localization Classes ................................................................. 183
   Providing Localized Messages and Labels ...................................................... 184
      Establishing the Locale .............................................................................. 185
      Setting the Resource Bundle .................................................................. 185
      Retrieving Localized Messages ............................................................... 186
   Date and Number Formatting ......................................................................... 187
   Character Sets and Encodings ........................................................................ 188
      Character Sets ......................................................................................... 188
      Character Encoding .................................................................................. 188

Part III Web Services ................................................. ................................. 191

10 JAX-RS: Advanced Topics and Example ............................................................ 193
   Annotations for Field and Bean Properties of Resource Classes .................. 193
      Extracting Path Parameters ...................................................................... 194
      Extracting Query Parameters .................................................................. 195
      Extracting Form Data .............................................................................. 195
      Extracting the Java Type of a Request or Response .................................. 196
   Subresources and Runtime Resource Resolution ........................................ 197
      Subresource Methods ............................................................................... 197
      Subresource Locators ............................................................................... 197
   Integrating JAX-RS with EJB Technology and CDI .................................... 198
   Conditional HTTP Requests ......................................................................... 199
   Runtime Content Negotiation ....................................................................... 200
   Using JAX-RS With JAXB ............................................................................. 202
      Using Java Objects to Model Your Data ................................................... 204
      Starting from an Existing XML Schema Definition .................................. 206
      Using JSON with JAX-RS and JAXB ......................................................... 208
   The customer Example Application ............................................................ 209
      Overview of the customer Example Application .................................... 209
      The Customer and Address Entity Classes .......................................... 209
      The CustomerService Class .................................................................... 212
      The CustomerClientXML and CustomerClientJSON Classes ............... 214
      Modifying the Example to Generate Entity Classes from an Existing Schema ... 216
## Contents

Running the customer Example ................................................................. 219

**Part IV  Enterprise Beans** ........................................................................ 225

11  **A Message-Driven Bean Example** ....................................................... 227

Overview of the simplemessage Example .................................................. 227
The simplemessage Application Client ...................................................... 228
The Message-Driven Bean Class ............................................................... 229
   The onMessage Method ........................................................................ 230
Running the simplemessage Example ....................................................... 231
   Administered Objects for the simplemessage Example ............................. 231
   ▼ To Run the simplemessage Application Using NetBeans IDE .............. 232
   ▼ To Run the simplemessage Application Using Ant ............................. 232
   Removing the Administered Objects for the simplemessage Example ......... 233

12  **Using the Embedded Enterprise Bean Container** ............................... 235

Overview of the Embedded Enterprise Bean Container ............................. 235
Developing Embeddable Enterprise Bean Applications ............................ 236
   Running Embedded Applications ......................................................... 236
Creating the Enterprise Bean Container ................................................... 237
Looking Up Session Bean References ....................................................... 238
Shutting Down the Enterprise Bean Container ......................................... 238
The standalone Example Application ....................................................... 239
   ▼ To Run the standalone Example Application .................................... 240

13  **Using Asynchronous Method Invocation in Session Beans** ............... 241

Asynchronous Method Invocation ............................................................ 241
   Creating an Asynchronous Business Method .......................................... 242
Calling Asynchronous Methods from Enterprise Bean Clients ................ 243
The async Example Application ............................................................. 244
   Architecture of the async Example Application ................................... 244
   Running the async Example ............................................................... 245
## Contents

<table>
<thead>
<tr>
<th>Part VI</th>
<th>Persistence .................................................................................................................. 291</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>Creating and Using String-Based Criteria Queries ........................................................................................................... 293</td>
</tr>
<tr>
<td></td>
<td>Overview of String-Based Criteria API Queries ................................................................................................................... 293</td>
</tr>
<tr>
<td></td>
<td>Creating String-Based Queries .................................................................................................................................................. 294</td>
</tr>
<tr>
<td></td>
<td>Executing String-Based Queries .............................................................................................................................................. 295</td>
</tr>
<tr>
<td>17</td>
<td>Controlling Concurrent Access to Entity Data with Locking .................................................................................................. 297</td>
</tr>
<tr>
<td></td>
<td>Overview of Entity Locking and Concurrency .......................................................................................................................... 297</td>
</tr>
<tr>
<td></td>
<td>Using Optimistic Locking ......................................................................................................................................................... 298</td>
</tr>
<tr>
<td></td>
<td>Lock Modes .................................................................................................................................................................................. 299</td>
</tr>
<tr>
<td></td>
<td>Setting the Lock Mode ......................................................................................................................................................... 300</td>
</tr>
<tr>
<td></td>
<td>Using Pessimistic Locking .................................................................................................................................................. 301</td>
</tr>
<tr>
<td>18</td>
<td>Using a Second-Level Cache with Java Persistence API Applications ........................................................................................... 303</td>
</tr>
<tr>
<td></td>
<td>Overview of the Second-Level Cache ...................................................................................................................................... 303</td>
</tr>
<tr>
<td></td>
<td>Controlling Whether Entities May Be Cached ................................................................................................................................ 304</td>
</tr>
<tr>
<td></td>
<td>Specifying the Cache Mode Settings to Improve Performance .................................................................................................. 305</td>
</tr>
<tr>
<td></td>
<td>Setting the Cache Retrieval and Store Modes ........................................................................................................................... 306</td>
</tr>
<tr>
<td></td>
<td>Controlling the Second-Level Cache Programmatically ..................................................................................................... 307</td>
</tr>
</tbody>
</table>

| Part VII | Security .................................................................................................................................................. 309 |

<p>| 19      | Java EE Security: Advanced Topics .................................................................................................. 311 |
|         | Working with Digital Certificates ................................................................................................. 311 |</p>
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS API Architecture</td>
<td>345</td>
</tr>
<tr>
<td>Messaging Domains</td>
<td>346</td>
</tr>
<tr>
<td>Message Consumption</td>
<td>348</td>
</tr>
<tr>
<td>The JMS API Programming Model</td>
<td>348</td>
</tr>
<tr>
<td>JMS Administered Objects</td>
<td>349</td>
</tr>
<tr>
<td>JMS Connections</td>
<td>351</td>
</tr>
<tr>
<td>JMS Sessions</td>
<td>352</td>
</tr>
<tr>
<td>JMS Message Producers</td>
<td>352</td>
</tr>
<tr>
<td>JMS Message Consumers</td>
<td>353</td>
</tr>
<tr>
<td>JMS Messages</td>
<td>355</td>
</tr>
<tr>
<td>JMS Queue Browsers</td>
<td>358</td>
</tr>
<tr>
<td>JMS Exception Handling</td>
<td>358</td>
</tr>
<tr>
<td>Creating Robust JMS Applications</td>
<td>359</td>
</tr>
<tr>
<td>Using Basic Reliability Mechanisms</td>
<td>359</td>
</tr>
<tr>
<td>Using Advanced Reliability Mechanisms</td>
<td>364</td>
</tr>
<tr>
<td>Using the JMS API in Java EE Applications</td>
<td>368</td>
</tr>
<tr>
<td>Using @Resource Annotations in Enterprise Bean or Web Components</td>
<td>369</td>
</tr>
<tr>
<td>Using Session Beans to Produce and to Synchronously Receive Messages</td>
<td>369</td>
</tr>
<tr>
<td>Using Message-Driven Beans to Receive Messages Asynchronously</td>
<td>370</td>
</tr>
<tr>
<td>Managing Distributed Transactions</td>
<td>373</td>
</tr>
<tr>
<td>Using the JMS API with Application Clients and Web Components</td>
<td>375</td>
</tr>
<tr>
<td>Further Information about JMS</td>
<td>376</td>
</tr>
</tbody>
</table>

21 Java Message Service Examples ............................................................... 377

Writing Simple JMS Applications ................................................................. 378
  A Simple Example of Synchronous Message Receives.................................. 378
  A Simple Example of Asynchronous Message Consumption.......................... 388
  A Simple Example of Browsing Messages in a Queue................................... 394
  Running JMS Clients on Multiple Systems............................................... 398
  Undeploying and Cleaning the Simple JMS Examples .................................. 405

Writing Robust JMS Applications .................................................................. 406
  A Message Acknowledgment Example ....................................................... 406
  A Durable Subscription Example ............................................................ 409
  A Local Transaction Example ................................................................. 411

An Application That Uses the JMS API with a Session Bean .......................... 416
Writing the Application Components for the clientsessionmdb Example .................................................417
Creating Resources for the clientsessionmdb Example ........................................................................419
Running the clientsessionmdb Example ..............................................................................................419
An Application That Uses the JMS API with an Entity .....................................................................421
Overview of the clientmdbentity Example Application ..................................................................422
Writing the Application Components for the clientmdbentity Example .......................................423
Creating Resources for the clientmdbentity Example .......................................................................426
Running the clientmdbentity Example ...............................................................................................426
An Application Example That Consumes Messages from a Remote Server ................................429
Overview of the consumeremote Example Modules......................................................................430
Writing the Module Components for the consumeremote Example ............................................431
Creating Resources for the consumeremote Example .....................................................................431
Using Two Application Servers for the consumeremote Example .................................................431
Running the consumeremote Example ...............................................................................................432
An Application Example That Deploys a Message-Driven Bean on Two Servers ...........................436
Overview of the sendremote Example Modules ...........................................................................436
Writing the Module Components for the sendremote Example .....................................................438
Creating Resources for the sendremote Example ...........................................................................439
▼ To Enable Deployment on the Remote System ...........................................................................440
▼ To Use Two Application Servers for the sendremote Example ..................................................440
Running the sendremote Example ....................................................................................................441
▼ To Disable Deployment on the Remote System ............................................................................446
22 Bean Validation: Advanced Topics ..................................................................................................449
Creating Custom Constraints ............................................................................................................449
   Using the Built-In Constraints to Make a New Constraint ...........................................................449
Customizing Validator Messages ........................................................................................................450
   The ValidationMessages Resource Bundle ..............................................................................450
Grouping Constraints ..........................................................................................................................451
   Customizing Group Validation Order .........................................................................................451
23 Using Java EE Interceptors .............................................................................................................453
Overview of Interceptors ....................................................................................................................453
   Interceptor Classes ......................................................................................................................454
   Interceptor Lifecycle ...................................................................................................................454
26 Duke’s Tutoring Case Study Example ................................................................. 479
   Design and Architecture of Duke’s Tutoring .................................................... 479
   Main Interface ...................................................................................................... 481
      Java Persistence API Entities Used in the Main Interface ............................ 481
      Enterprise Beans Used in the Main Interface ................................................ 482
      Facelets Files Used in the Main Interface ....................................................... 483
      Helper Classes Used in the Main Interface .................................................... 484
      Properties Files ................................................................................................ 484
      Deployment Descriptors Used in Duke’s Tutoring ........................................... 485
   Administration Interface ...................................................................................... 486
      Enterprise Beans Used in the Administration Interface ................................. 486
      Facelets Files Used in the Administration Interface ....................................... 486
   Running the Duke’s Tutoring Case Study Application ........................................ 487
      Setting Up GlassFish Server ........................................................................... 487
      Running Duke’s Tutoring .............................................................................. 488

27 Duke’s Forest Case Study Example ................................................................. 491
   Design and Architecture of Duke’s Forest .......................................................... 492
      The events Project ............................................................................................. 494
      The entities Project .......................................................................................... 495
      The dukes-payment Project ............................................................................ 497
      The dukes-resources Project .......................................................................... 498
      The Duke’s Store Project ................................................................................. 498
      The Duke’s Shipment Project ......................................................................... 503
   Building and Deploying the Duke’s Forest Case Study Application .................. 506
      Prerequisite Task .............................................................................................. 506
      ▼ To Build and Deploy the Duke’s Forest Application Using NetBeans IDE 507
      ▼ To Build and Deploy the Duke’s Forest Application Using Ant .................. 508
   Running the Duke’s Forest Application .............................................................. 509
      ▼ To Register as a Duke’s Store Customer ..................................................... 509
      ▼ To Purchase Products .................................................................................. 509
      ▼ To Approve Shipment of a Product ............................................................. 510
      ▼ To Create a New Product ............................................................................ 510

Index ....................................................................................................................... 513
Preface

This tutorial is the second volume of a guide to developing enterprise applications for the Java Platform, Enterprise Edition 6 (Java EE 6) using GlassFish Server Open Source Edition.

Oracle GlassFish Server, a Java EE compatible application server, is based on GlassFish Server Open Source Edition, 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 http://glassfish.java.net/, is the first compatible implementation of the Java EE 6 platform specification. This lightweight, flexible, and open-source application server enables organizations not only to leverage the new capabilities introduced within the Java EE 6 specification, but also to add to their existing capabilities through a faster and more streamlined development and deployment cycle. Oracle GlassFish Server, the product version, and GlassFish Server Open Source Edition, the open-source version, are hereafter referred to as GlassFish Server.

Before You Read This Book

Before proceeding with this book, you should be familiar with Volume One of this tutorial, The Java EE 6 Tutorial: Basic Concepts. Both volumes assume that you have a good knowledge of the Java programming language. A good way to get to that point is to read the Java Tutorials, available at http://docs.oracle.com/javase/.

Related Documentation

Javadoc tool reference documentation for packages that are provided with GlassFish Server is available as follows.

- The API specification for version 6 of Java EE is located at http://docs.oracle.com/javaee/6/api/.
- The API specification for GlassFish Server, including Java EE 6 platform packages and nonplatform packages that are specific to the GlassFish Server product, is located at http://glassfish.java.net/nonav/docs/v3/api/.

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

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

For information about the JavaDB database for use with the GlassFish Server, see http://www.oracle.com/technetwork/java/javadb/overview/index.html.

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 http://glassfish-samples.java.net/.

## Typographic Conventions

Table P–1 describes the typographic changes that are used in this book.

<table>
<thead>
<tr>
<th>Typeface</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>AaBbCc123</td>
<td>The names of commands, files, and directories, and onscreen computer output</td>
<td>Edit your .login file. Use <code>ls -a</code> to list all files. <code>machine_name% you have mail</code>.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>What you type, contrasted with onscreen computer output</td>
<td><code>machine_name% su</code> Password:</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>A placeholder to be replaced with a real name or value</td>
<td>The command to remove a file is <code>rm filename</code>.</td>
</tr>
<tr>
<td>AaBbCc123</td>
<td>Book titles, new terms, and terms to be emphasized (note that some emphasized items appear bold online)</td>
<td>Read Chapter 6 in the User’s Guide. A cache is a copy that is stored locally. Do not save the file.</td>
</tr>
</tbody>
</table>
Default Paths and File Names

Table P–2 describes the default paths and file names that are used in this book.

<table>
<thead>
<tr>
<th>Placeholder</th>
<th>Description</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>as-install</td>
<td>Represents the base installation directory for the GlassFish Server or the SDK of which the GlassFish Server is a part.</td>
<td>Installations on the Solaris operating system, Linux operating system, and Mac operating system: &lt;br&gt; <code>user's-home-directory/glassfish3/glassfish</code>&lt;br&gt;Windows, all installations: <code>SystemDrive:\glassfish3\glassfish</code></td>
</tr>
<tr>
<td>as-install-parent</td>
<td>Represents the parent of the base installation directory for GlassFish Server.</td>
<td>Installations on the Solaris operating system, Linux operating system, and Mac operating system: &lt;br&gt; <code>user's-home-directory/glassfish3</code>&lt;br&gt;Windows, all installations: <code>SystemDrive:\glassfish3</code></td>
</tr>
<tr>
<td>tut-install</td>
<td>Represents the base installation directory for the Java EE Tutorial after you install the GlassFish Server or the SDK and run the Update Tool.</td>
<td><code>as-install/docs/javaee-tutorial</code></td>
</tr>
<tr>
<td>domain-root-dir</td>
<td>Represents the directory in which a domain is created by default.</td>
<td><code>as-install/domains/</code></td>
</tr>
<tr>
<td>domain-dir</td>
<td>Represents the directory in which a domain's configuration is stored.</td>
<td><code>domain-root-dir/domain-name</code></td>
</tr>
</tbody>
</table>

Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.
Acknowledgments

The Java EE tutorial team would like to thank the Java EE specification leads: Roberto Chinnici, Bill Shannon, Kenneth Saks, Linda DeMichiel, Ed Burns, Roger Kitain, Ron Monzillo, Binod PG, Sivakumar Thyagarajan, Kin-Man Chung, Jitendra Kotamraju, Marc Hadley, Paul Sandoz, Gavin King, Emmanuel Bernard, Rod Johnson, Bob Lee, and Rajiv Mordani.

Thanks also to Alejandro Murillo for the original version of the connector example.

We would also like to thank the Java EE 6 SDK team, especially Carla Carlson, Snjezana Sevo-Zenzerovic, Adam Leftik, and John Clingan.

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

The EJB technology, Java Persistence API, and Criteria API chapters were written with extensive input from the EJB and Persistence teams, including Marina Vatkina and Mitesh Meswani.

We’d like to thank Sivakumar Thyagarajan for his reviews of the CDI chapters and Tim Quinn for assistance with the application client container. Thanks also to the NetBeans engineering and documentation teams, particularly Petr Jiricka, John Jullion-Ceccarelli, and Troy Giunipero, for their help in enabling NetBeans IDE support for the code examples.

Chang Feng, Alejandro Murillo, and Scott Fordin helped internationalize the Duke’s Tutoring case study.

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. Sheila Cepero helped smooth our path in many ways. Steve Cogorno provided invaluable help with our tools.
Finally, we would like to express our profound appreciation to Greg Doench, John Fuller, Elizabeth Ryan, Steve Freedkin, and the production team at Addison-Wesley for graciously seeing our manuscript to publication.
The Java EE 6 Tutorial: Basic Concepts introduces JavaServer Faces technology and Facelets, the preferred presentation layer for the Java EE platform. This chapter and the following chapters introduce advanced concepts in this area.

- This chapter describes the JavaServer Faces lifecycle in detail. Some of the complex JavaServer Faces applications use the well-defined lifecycle phases to customize application behavior.
- Chapter 5, “Composite Components: Advanced Topics and Example,” introduces advanced features of composite components.
- Chapter 6, “Creating Custom UI Components and Other Custom Objects,” describes the process of creating new components, renderers, converters, listeners, and validators from scratch.
- Chapter 7, “Configuring JavaServer Faces Applications,” introduces the process of creating and deploying JavaServer Faces applications, the use of various configuration files, and the deployment structure.

The following topics are addressed here:
- “The Lifecycle of a JavaServer Faces Application” on page 50
- “Partial Processing and Partial Rendering” on page 56
- “The Lifecycle of a Facelets Application” on page 56
- “User Interface Component Model” on page 57
The Lifecycle of a JavaServer Faces Application

The lifecycle of an application refers to the various stages of processing of that application, from its initiation to its conclusion. All applications have lifecycles. During a web application lifecycle, common tasks such as the following are performed:

- Handling incoming requests
- Decoding parameters
- Modifying and saving state
- Rendering web pages to the browser

The JavaServer Faces web application framework manages lifecycle phases automatically for simple applications or allows you to manage them manually for more complex applications as required.

JavaServer Faces applications that use advanced features may require interaction with the lifecycle at certain phases. For example, Ajax applications use partial processing features of the lifecycle. A clearer understanding of the lifecycle phases is key to creating well-designed components.

A simplified view of the JavaServer faces lifecycle, consisting of the two main phases of a JavaServer Faces web application, is introduced in “The Lifecycle of the hello Application” in The Java EE 6 Tutorial: Basic Concepts. This section examines the JavaServer Faces lifecycle in more detail.

Overview of the JavaServer Faces Lifecycle

The lifecycle of a JavaServer Faces application begins when the client makes an HTTP request for a page and ends when the server responds with the page, translated to HTML.

The lifecycle can be divided into two main phases, execute and render. The execute phase is further divided into subphases to support the sophisticated component tree. This structure requires that component data be converted and validated, component events be handled, and component data be propagated to beans in an orderly fashion.

A JavaServer Faces page is represented by a tree of components, called a view. During the lifecycle, the JavaServer Faces implementation must build the view while considering the state saved from a previous submission of the page. When the client requests a page, the JavaServer Faces implementation performs several tasks, such as validating the data input of components in the view and converting input data to types specified on the server side.

The JavaServer Faces implementation performs all these tasks as a series of steps in the JavaServer Faces request-response lifecycle. Figure 3–1 illustrates these steps.
FIGURE 3–1  JavaServer Faces Standard Request-Response Lifecycle

The Lifecycle of a JavaServer Faces Application
The request-response lifecycle handles two kinds of requests: initial requests and postbacks. An initial request occurs when a user makes a request for a page for the first time. A postback request occurs when a user submits the form contained on a page that was previously loaded into the browser as a result of executing an initial request.

When the lifecycle handles an initial request, it executes only the Restore View and Render Response phases, because there is no user input or action to process. Conversely, when the lifecycle handles a postback, it executes all of the phases.

Usually, the first request for a JavaServer Faces page comes in from a client, as a result of clicking a link or button component on a JavaServer Faces page. To render a response that is another JavaServer Faces page, the application creates a new view and stores it in the `javax.faces.context.FacesContext` instance, which represents all of the information associated with processing an incoming request and creating a response. The application then acquires object references needed by the view and calls the `FacesContext.renderResponse` method, which forces immediate rendering of the view by skipping to the Render Response phase of the lifecycle, as is shown by the arrows labelled Render Response in the diagram.

Sometimes, an application might need to redirect to a different web application resource, such as a web service, or generate a response that does not contain JavaServer Faces components. In these situations, the developer must skip the Render Response phase by calling the `FacesContext.responseComplete` method. This situation is also shown in the diagram, with the arrows labelled Response Complete.

The most common situation is that a JavaServer Faces component submits a request for another JavaServer Faces page. In this case, the JavaServer Faces implementation handles the request and automatically goes through the phases in the lifecycle to perform any necessary conversions, validations, and model updates, and to generate the response.

There is one exception to the lifecycle described in this section. When a component’s immediate attribute is set to true, the validation, conversion, and events associated with these components are processed during the Apply Request Values phase rather than in a later phase.

The details of the lifecycle explained in the following sections are primarily intended for developers who need to know information such as when validations, conversions, and events are usually handled and ways to change how and when they are handled. For more information on each of the lifecycle phases, download the latest JavaServer Faces Specification documentation from [http://jcp.org/en/jsr/detail?id=314](http://jcp.org/en/jsr/detail?id=314).

The JavaServer Faces application lifecycle execute phase contains the following subphases:

- “Restore View Phase” on page 53
- “Apply Request Values Phase” on page 53
Restore View Phase

When a request for a JavaServer Faces page is made, usually by an action such as when a link or a button component is clicked, the JavaServer Faces implementation begins the Restore View phase.

During this phase, the JavaServer Faces implementation builds the view of the page, wires event handlers and validators to components in the view, and saves the view in the FacesContext instance, which contains all the information needed to process a single request. All the application’s components, event handlers, converters, and validators have access to the FacesContext instance.

If the request for the page is an initial request, the JavaServer Faces implementation creates an empty view during this phase and the lifecycle advances to the Render Response phase, during which the empty view is populated with the components referenced by the tags in the page.

If the request for the page is a postback, a view corresponding to this page already exists in the FacesContext instance. During this phase, the JavaServer Faces implementation restores the view by using the state information saved on the client or the server.

Apply Request Values Phase

After the component tree is restored during a postback request, each component in the tree extracts its new value from the request parameters by using its decode (processDecodes()) method. The value is then stored locally on each component.

If any decode methods or event listeners have called the renderResponse method on the current FacesContext instance, the JavaServer Faces implementation skips to the Render Response phase.

If any events have been queued during this phase, the JavaServer Faces implementation broadcasts the events to interested listeners.

If some components on the page have their immediate attributes (see “The immediate Attribute” in The Java EE 6 Tutorial: Basic Concepts) set to true, then the validations, conversions, and events associated with these components will be processed during this phase. If any conversion fails, an error message associated with the component is
generated and queued on FacesContext. This message will be displayed during the Render Response phase, along with any validation errors resulting from the Process Validations phase.

At this point, if the application needs to redirect to a different web application resource or generate a response that does not contain any JavaServer Faces components, it can call the FacesContext.responseComplete method.

At the end of this phase, the components are set to their new values, and messages and events have been queued.

If the current request is identified as a partial request, the partial context is retrieved from the FacesContext, and the partial processing method is applied.

**Process Validations Phase**

During this phase, the JavaServer Faces implementation processes all validators registered on the components in the tree, by using its `validate(processValidators)` method. It examines the component attributes that specify the rules for the validation and compares these rules to the local value stored for the component. The JavaServer Faces implementation also completes conversions for input components that do not have the `immediate` attribute set to true.

If the local value is invalid, or if any conversion fails, the JavaServer Faces implementation adds an error message to the FacesContext instance, and the lifecycle advances directly to the Render Response phase so that the page is rendered again with the error messages displayed. If there were conversion errors from the Apply Request Values phase, the messages for these errors are also displayed.

If any `validate` methods or event listeners have called the `renderResponse` method on the current FacesContext, the JavaServer Faces implementation skips to the Render Response phase.

At this point, if the application needs to redirect to a different web application resource or generate a response that does not contain any JavaServer Faces components, it can call the FacesContext.responseComplete method.

If events have been queued during this phase, the JavaServer Faces implementation broadcasts them to interested listeners.

If the current request is identified as a partial request, the partial context is retrieved from the FacesContext, and the partial processing method is applied.
Update Model Values Phase

After the JavaServer Faces implementation determines that the data is valid, it traverses the component tree and sets the corresponding server-side object properties to the components’ local values. The JavaServer Faces implementation updates only the bean properties pointed at by an input component’s value attribute. If the local data cannot be converted to the types specified by the bean properties, the lifecycle advances directly to the Render Response phase so that the page is re-rendered with errors displayed. This is similar to what happens with validation errors.

If any `updateModel()` methods or any listeners have called the `renderResponse` method on the current `FacesContext` instance, the JavaServer Faces implementation skips to the Render Response phase.

At this point, if the application needs to redirect to a different web application resource or generate a response that does not contain any JavaServer Faces components, it can call the `FacesContext.responseComplete` method.

If any events have been queued during this phase, the JavaServer Faces implementation broadcasts them to interested listeners.

If the current request is identified as a partial request, the partial context is retrieved from the `FacesContext`, and the partial processing method is applied.

Invoke Application Phase

During this phase, the JavaServer Faces implementation handles any application-level events, such as submitting a form or linking to another page.

At this point, if the application needs to redirect to a different web application resource or generate a response that does not contain any JavaServer Faces components, it can call the `FacesContext.responseComplete` method.

If the view being processed was reconstructed from state information from a previous request and if a component has fired an event, these events are broadcast to interested listeners.

Finally, the JavaServer Faces implementation transfers control to the Render Response phase.

Render Response Phase

During this phase, JavaServer Faces builds the view and delegates authority to the appropriate resource for rendering the pages.
If this is an initial request, the components that are represented on the page will be added to the component tree. If this is not an initial request, the components are already added to the tree, so they need not be added again.

If the request is a postback and errors were encountered during the Apply Request Values phase, Process Validations phase, or Update Model Values phase, the original page is rendered again during this phase. If the pages contain \texttt{h:message} or \texttt{h:messages} tags, any queued error messages are displayed on the page.

After the content of the view is rendered, the state of the response is saved so that subsequent requests can access it. The saved state is available to the Restore View phase.

**Partial Processing and Partial Rendering**

The JavaServer Faces lifecycle spans all of the execute and render processes of an application. It is also possible to process and render only parts of an application, such as a single component. For example, the JavaServer Faces Ajax framework can generate requests containing information on which particular component may be processed and which particular component may be rendered back to the client.

Once such a partial request enters the JavaServer Faces lifecycle, the information is identified and processed by a \texttt{javax.faces.context.PartialViewContext} object. The JavaServer Faces lifecycle is still aware of such Ajax requests and modifies the component tree accordingly.

The execute and render attributes of the \texttt{f:ajax} tag are used to identify which components may be executed and rendered. For more information on these attributes, see Chapter 4, “Using Ajax with JavaServer Faces Technology.”

**The Lifecycle of a Facelets Application**

The JavaServer Faces specification defines the lifecycle of a JavaServer Faces application. For more information on this lifecycle, see “The Lifecycle of a JavaServer Faces Application” on page 50. The following steps describe that process as applied to a Facelets-based application.

1. When a client, such as a browser, makes a new request to a page that is created using Facelets, a new component tree or \texttt{javax.faces.component.UIViewRoot} is created and placed in the \texttt{FacesContext}.
2. The \texttt{UIViewRoot} is applied to the Facelets, and the view is populated with components for rendering.
3. The newly built view is rendered back as a response to the client.
4. On rendering, the state of this view is stored for the next request. The state of input components and form data is stored.

5. The client may interact with the view and request another view or change from the JavaServer Faces application. At this time the saved view is restored from the stored state.

6. The restored view is once again passed through the JavaServer Faces lifecycle, which eventually will either generate a new view or re-render the current view if there were no validation problems and no action was triggered.

7. If the same view is requested, the stored view is rendered once again.

8. If a new view is requested, then the process described in Step 2 is continued.

9. The new view is then rendered back as a response to the client.

**User Interface Component Model**

In addition to the lifecycle description, an overview of JavaServer Faces architecture provides better understanding of the technology.

JavaServer Faces components are the building blocks of a JavaServer Faces view. A component can be a user interface (UI) component or a non-Ul component.

JavaServer Faces UI components are configurable, reusable elements that compose the user interfaces of JavaServer Faces applications. A component can be simple, such as a button, or can be compound, such as a table, composed of multiple components.

JavaServer Faces technology provides a rich, flexible component architecture that includes the following:

- A set of `javax.faces.component.UIComponent` classes for specifying the state and behavior of UI components
- A rendering model that defines how to render the components in various ways
- A conversion model that defines how to register data converters onto a component
- An event and listener model that defines how to handle component events
- A validation model that defines how to register validators onto a component
- A navigation model that defines page navigation and the sequence in which pages are loaded

This section briefly describes each of these pieces of the component architecture.
User Interface Component Classes

JavaServer Faces technology provides a set of UI component classes and associated behavioral interfaces that specify all the UI component functionality, such as holding component state, maintaining a reference to objects, and driving event handling and rendering for a set of standard components.

The component classes are completely extensible, allowing component writers to create their own custom components. See Chapter 6, “Creating Custom UI Components and Other Custom Objects,” for more information.

The abstract base class for all components is `javax.faces.component.UIComponent`. JavaServer Faces UI component classes extend the `UIComponentBase` class (a subclass of `UIComponent`), which defines the default state and behavior of a component. The following set of component classes is included with JavaServer Faces technology:

- **UIComponent**: Represents a single column of data in a `UIData` component.
- **UICommand**: Represents a control that fires actions when activated.
- **UIData**: Represents a data binding to a collection of data represented by a `javax.faces.model.DataModel` instance.
- **UIForm**: Represents an input form to be presented to the user. Its child components represent (among other things) the input fields to be included when the form is submitted. This component is analogous to the `form` tag in HTML.
- **UIGraphic**: Displays an image.
- **UIInput**: Takes data input from a user. This class is a subclass of `UIOutput`.
- **UIMessage**: Displays a localized error message.
- **UIMessages**: Displays a set of localized error messages.
- **UIOutcomeTarget**: Displays a hyperlink in the form of a link or a button.
- **UIOutput**: Displays data output on a page.
- **UIPanel**: Manages the layout of its child components.
- **UIParameter**: Represents substitution parameters.
- **UISelectBoolean**: Allows a user to set a `boolean` value on a control by selecting or deselecting it. This class is a subclass of the `UIInput` class.
- **UISelectItem**: Represents a single item in a set of items.
- **UISelectItems**: Represents an entire set of items.
- **UISelectMany**: Allows a user to select multiple items from a group of items. This class is a subclass of the `UIInput` class.
- **UISelectOne**: Allows a user to select one item from a group of items. This class is a subclass of the `UIInput` class.
- **UIViewParameter**: Represents the query parameters in a request. This class is a subclass of the UIInput class.

- **UIViewRoot**: Represents the root of the component tree.

In addition to extending UIComponentBase, the component classes also implement one or more behavioral interfaces, each of which defines certain behavior for a set of components whose classes implement the interface.

These behavioral interfaces, all defined in the javax.faces.component package unless otherwise stated, are as follows:

- **ActionSource**: Indicates that the component can fire an action event. This interface is intended for use with components based on JavaServer Faces technology 1.1_01 and earlier versions. This interface is deprecated in JavaServer Faces 2.

- **ActionSource2**: Extends ActionSource, and therefore provides the same functionality. However, it allows components to use the Expression Language (EL) when they are referencing methods that handle action events.

- **EditableValueHolder**: Extends ValueHolder and specifies additional features for editable components, such as validation and emitting value-change events.

- **NamingContainer**: Mandates that each component rooted at this component have a unique ID.

- **StateHolder**: Denotes that a component has state that must be saved between requests.

- **ValueHolder**: Indicates that the component maintains a local value as well as the option of accessing data in the model tier.

- **javax.faces.event.SystemEventListenerHolder**: Maintains a list of javax.faces.event.SystemEventListener instances for each type of javax.faces.event.SystemEvent defined by that class.

- **javax.faces.component.behavior.ClientBehaviorHolder**: Adds the ability to attach javax.faces.component.behavior.ClientBehavior instances such as a reusable script.

UICommand implements ActionSource2 and StateHolder. UIOutput and component classes that extend UIOutput implement StateHolder and ValueHolder. UIInput and component classes that extend UIInput implement EditableValueHolder, StateHolder, and ValueHolder. UIComponentBase implements StateHolder.

Only component writers will need to use the component classes and behavioral interfaces directly. Page authors and application developers will use a standard component by including a tag that represents it on a page. Most of the components can be rendered in different ways on a page. For example, a UICommand component can be rendered as a button or a hyperlink.
The next section explains how the rendering model works and how page authors can choose to render the components by selecting the appropriate tags.

## Component Rendering Model

The JavaServer Faces component architecture is designed such that the functionality of the components is defined by the component classes, whereas the component rendering can be defined by a separate renderer class. This design has several benefits, including the following:

- Component writers can define the behavior of a component once but create multiple renderers, each of which defines a different way to render the component to the same client or to different clients.
- Page authors and application developers can change the appearance of a component on the page by selecting the tag that represents the appropriate combination of component and renderer.

A render kit defines how component classes map to component tags that are appropriate for a particular client. The JavaServer Faces implementation includes a standard HTML render kit for rendering to an HTML client.

The render kit defines a set of `javax.faces.render.Renderer` classes for each component that it supports. Each `Renderer` class defines a different way to render the particular component to the output defined by the render kit. For example, a `UISelectOne` component has three different renderers. One of them renders the component as a set of radio buttons. Another renders the component as a combo box. The third one renders the component as a list box. Similarly, a `UICommand` component can be rendered as a button or a hyperlink, using the `h:commandButton` or `h:commandLink` tag. The command part of each tag corresponds to the `UICommand` class, specifying the functionality, which is to fire an action. The `Button` or `Link` part of each tag corresponds to a separate `Renderer` class that defines how the component appears on the page.

Each custom tag defined in the standard HTML render kit is composed of the component functionality (defined in the `UIComponent` class) and the rendering attributes (defined by the `Renderer` class).

The section “Adding Components to a Page Using HTML Tags” in *The Java EE 6 Tutorial: Basic Concepts* lists all supported component tags and illustrates how to use the tags in an example.

The JavaServer Faces implementation provides a custom tag library for rendering components in HTML.
Conversion Model

A JavaServer Faces application can optionally associate a component with server-side object data. This object is a JavaBeans component, such as a managed bean. An application gets and sets the object data for a component by calling the appropriate object properties for that component.

When a component is bound to an object, the application has two views of the component’s data:

- The model view, in which data is represented as data types, such as int or long.
- The presentation view, in which data is represented in a manner that can be read or modified by the user. For example, a java.util.Date might be represented as a text string in the format mm/dd/yy or as a set of three text strings.

The JavaServer Faces implementation automatically converts component data between these two views when the bean property associated with the component is of one of the types supported by the component’s data. For example, if a UISelectBoolean component is associated with a bean property of type java.lang.Boolean, the JavaServer Faces implementation will automatically convert the component’s data from String to Boolean. In addition, some component data must be bound to properties of a particular type. For example, a UISelectBoolean component must be bound to a property of type boolean or java.lang.Boolean.

Sometimes you might want to convert a component’s data to a type other than a standard type, or you might want to convert the format of the data. To facilitate this, JavaServer Faces technology allows you to register a javax.faces.convert.Converter implementation on UIOutput components and components whose classes subclass UIOutput. If you register the Converter implementation on a component, the Converter implementation converts the component’s data between the two views.

You can either use the standard converters supplied with the JavaServer Faces implementation or create your own custom converter. Custom converter creation is covered in Chapter 6, “Creating Custom UI Components and Other Custom Objects.”

Event and Listener Model

The JavaServer Faces event and listener model is similar to the JavaBeans event model in that it has strongly typed event classes and listener interfaces that an application can use to handle events generated by components.

The JavaServer Faces specification defines three types of events: application events, system events, and data-model events.
Application events are tied to a particular application and are generated by a UIComponent. They represent the standard events available in previous versions of JavaServer Faces technology.

An event object identifies the component that generated the event and stores information about the event. To be notified of an event, an application must provide an implementation of the listener class and must register it on the component that generates the event. When the user activates a component, such as by clicking a button, an event is fired. This causes the JavaServer Faces implementation to invoke the listener method that processes the event.

JavaServer Faces supports two kinds of application events: action events and value-change events.

An action event (class javax.faces.event.ActionEvent) occurs when the user activates a component that implements javax.faces.component.ActionSource. These components include buttons and hyperlinks.

A value-change event (class javax.faces.event.ValueChangeEvent) occurs when the user changes the value of a component represented by UIInput or one of its subclasses. An example is selecting a check box, an action that results in the component's value changing to true. The component types that can generate these types of events are the UIInput, UISelectOne, UISelectMany, and UISelectBoolean components.

Value-change events are fired only if no validation errors are detected.

Depending on the value of the immediate property (see “The immediate Attribute” in The Java EE 6 Tutorial: Basic Concepts) of the component emitting the event, action events can be processed during the invoke application phase or the apply request values phase, and value-change events can be processed during the process validations phase or the apply request values phase.

System events are generated by an Object rather than a UIComponent. They are generated during the execution of an application at predefined times. They are applicable to the entire application rather than to a specific component.

A data-model event occurs when a new row of a UIData component is selected.

There are two ways to cause your application to react to action events or value-change events that are emitted by a standard component:

- Implement an event listener class to handle the event and register the listener on the component by nesting either an f:valueChangeListener tag or an f:actionListener tag inside the component tag.
- Implement a method of a managed bean to handle the event and refer to the method with a method expression from the appropriate attribute of the component's tag.
See “Implementing an Event Listener” on page 117 for information on how to implement an event listener. See “Registering Listeners on Components” in The Java EE 6 Tutorial: Basic Concepts for information on how to register the listener on a component.

See “Writing a Method to Handle an Action Event” in The Java EE 6 Tutorial: Basic Concepts and “Writing a Method to Handle a Value-Change Event” in The Java EE 6 Tutorial: Basic Concepts for information on how to implement managed bean methods that handle these events.

See ”Referencing a Managed Bean Method” in The Java EE 6 Tutorial: Basic Concepts for information on how to refer to the managed bean method from the component tag.

When emitting events from custom components, you must implement the appropriate event class and manually queue the event on the component in addition to implementing an event listener class or a managed bean method that handles the event. “Handling Events for Custom Components” on page 119 explains how to do this.

Validation Model

JavaServer Faces technology supports a mechanism for validating the local data of editable components (such as text fields). This validation occurs before the corresponding model data is updated to match the local value.

Like the conversion model, the validation model defines a set of standard classes for performing common data validation checks. The JavaServer Faces core tag library also defines a set of tags that correspond to the standard javax.faces.validator.Validator implementations. See “Using the Standard Validators” in The Java EE 6 Tutorial: Basic Concepts for a list of all the standard validation classes and corresponding tags.

Most of the tags have a set of attributes for configuring the validator’s properties, such as the minimum and maximum allowable values for the component’s data. The page author registers the validator on a component by nesting the validator’s tag within the component’s tag.

In addition to validators that are registered on the component, you can declare a default validator which is registered on all UIInput components in the application. For more information on default validators, see ”Using Default Validators” on page 159.
The validation model also allows you to create your own custom validator and corresponding tag to perform custom validation. The validation model provides two ways to implement custom validation:

- Implement a `Validator` interface that performs the validation.
- Implement a managed bean method that performs the validation.

If you are implementing a `Validator` interface, you must also:

- Register the `Validator` implementation with the application.
- Create a custom tag or use an `f:validator` tag to register the validator on the component.

In the previously described standard validation model, the validator is defined for each input component on a page. The Bean Validation model allows the validator to be applied to all fields in a page. See “Using Bean Validation” in *The Java EE 6 Tutorial: Basic Concepts* and Chapter 22, “Bean Validation: Advanced Topics,” for more information on Bean Validation.

**Navigation Model**

The JavaServer Faces navigation model makes it easy to define page navigation and to handle any additional processing that is needed to choose the sequence in which pages are loaded.

In JavaServer Faces technology, *navigation* is a set of rules for choosing the next page or view to be displayed after an application action, such as when a button or hyperlink is clicked.

Navigation can be implicit or user-defined. Implicit navigation comes into play when user-defined navigation rules are not available. For more information on implicit navigation, see “Implicit Navigation Rules” on page 164.

User-defined navigation rules are declared in zero or more application configuration resource files, such as `faces-config.xml`, by using a set of XML elements. The default structure of a navigation rule is as follows:

```xml
<navigation-rule>
  <description></description>
  <from-view-id></from-view-id>
  <navigation-case>
    <from-action></from-action>
    <from-outcome></from-outcome>
    <if></if>
    <to-view-id></to-view-id>
  </navigation-case>
</navigation-rule>
```
User-defined navigation is handled as follows:

- Define the rules in the application configuration resource file.
- Refer to an outcome string from the button or hyperlink component’s action attribute. This outcome string is used by the JavaServer Faces implementation to select the navigation rule.

Here is an example navigation rule:

```xml
<navigation-rule>
    <from-view-id>/greeting.xhtml</from-view-id>
    <navigation-case>
        <from-outcome>success</from-outcome>
        <to-view-id>/response.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```

This rule states that when a command component (such as an h:commandButton or an h:commandLink) on greeting.xhtml is activated, the application will navigate from the greeting.xhtml page to the response.xhtml page if the outcome referenced by the button component’s tag is success. Here is the h:commandButton tag from greeting.xhtml that specifies a logical outcome of success:

```xml
<h:commandButton id="submit" action="success" value="Submit" />
```

As the example demonstrates, each navigation-rule element defines how to get from one page (specified in the from-view-id element) to the other pages of the application. The navigation-rule elements can contain any number of navigation-case elements, each of which defines the page to open next (defined by to-view-id) based on a logical outcome (defined by from-outcome).

In more complicated applications, the logical outcome can also come from the return value of an action method in a managed bean. This method performs some processing to determine the outcome. For example, the method can check whether the password the user entered on the page matches the one on file. If it does, the method might return success; otherwise, it might return failure. An outcome of failure might result in the logon page being reloaded. An outcome of success might cause the page displaying the user’s credit card activity to open. If you want the outcome to be returned by a method on a bean, you must refer to the method using a method expression, with the action attribute, as shown by this example:

```xml
<h:commandButton id="submit"
    action="#{userNumberBean.getOrderStatus}" value="Submit" />
```

When the user clicks the button represented by this tag, the corresponding component generates an action event. This event is handled by the default
javax.faces.event.ActionListener instance, which calls the action method referenced by the component that triggered the event. The action method returns a logical outcome to the action listener.

The listener passes the logical outcome and a reference to the action method that produced the outcome to the default javax.faces.application.NavigationHandler. The NavigationHandler selects the page to display next by matching the outcome or the action method reference against the navigation rules in the application configuration resource file by the following process:

1. The NavigationHandler selects the navigation rule that matches the page currently displayed.
2. It matches the outcome or the action method reference that it received from the default javax.faces.event.ActionListener with those defined by the navigation cases.
3. It tries to match both the method reference and the outcome against the same navigation case.
4. If the previous step fails, the navigation handler attempts to match the outcome.
5. Finally, the navigation handler attempts to match the action method reference if the previous two attempts failed.
6. If no navigation case is matched, it displays the same view again.

When the NavigationHandler achieves a match, the render response phase begins. During this phase, the page selected by the NavigationHandler will be rendered.

The Duke's Tutoring case study example application uses navigation rules in the business methods that handle creating, editing, and deleting the users of the application. For example, the form for creating a student has the following h:commandButton tag:

```html
<h:commandButton id="submit" action="#{adminBean.createStudent(studentManager.newStudent)}"
    value="#{bundle['action.submit']}"/>
```

The action event calls the dukestutoring.ejb.AdminBean.createStudent method:

```java
public String createStudent(Student student) {
    em.persist(student);
    return "createdStudent";
}
```

The return value of createdStudent has a corresponding navigation case in the faces-config.xml configuration file:

```xml
<navigation-rule>
    <from-view-id>/admin/student/createStudent.xhtml</from-view-id>
```

User Interface Component Model

The Java EE 6 Tutorial
After the student is created, the user is returned to the Administration index page.

For more information on how to define navigation rules, see “Configuring Navigation Rules” on page 161.

For more information on how to implement action methods to handle navigation, see “Writing a Method to Handle an Action Event” in The Java EE 6 Tutorial: Basic Concepts.

For more information on how to reference outcomes or action methods from component tags, see “Referencing a Method That Performs Navigation” in The Java EE 6 Tutorial: Basic Concepts.
This page intentionally left blank
Index

Numbers and Symbols
@Alternative annotation, 251–253
@ApplicationScoped annotation, 143
@AroundInvoke annotation, 454
@AroundTimeout annotation, 454
@Asynchronous annotation, 242
@Context annotation, 193–196
@CookieParam annotation, 193–196
@CustomScoped annotation, 143
@Decorator annotation, 262–263
@Delegate annotation, 262–263
@Disposes annotation, 256
@FormParam annotation, 193–196
@GroupSequence annotation, 451–452
@HeaderParam annotation, 193–196
@ManagedBean annotation, 142–143
@MatrixParam annotation, 193–196
@MessageDriven annotation, 419
@MultipartConfig annotation, 175–176
@NoneScoped annotation, 143
@Observe annotation, 258–259
@PathParam annotation, 193–196
@PostConstruct annotation, 454
    session beans using JMS, 418
@PreDestroy annotation, 454
    session beans using JMS, 418
@Produces annotation, 254–256
@QueryParam annotation, 193–196
@RequestScoped annotation, 143
@Resource annotation
    JMS resources, 228, 350, 351
@ResourceDependency annotation, 81
@ResourceScoped annotation, 143
@SessionScoped annotation, 143
@ViewScoped annotation, 143

A
acknowledge method, 360
acknowledging messages, See message
acknowledgment
action events, 61–63, 65, 117–119
    ActionEvent class, 117, 118
    actionListener attribute, 102
    ActionListener implementation, 117
        118–119
    f:actionListener tag, 98
        processAction(ActionEvent) method, 118
action method, 65
administered objects, 345, 349–351
    See also connection factories, destinations
        creating and removing, 381–383
Administration Console, 35
    starting, 42–43
Ajax
    error handling, 76–77
    event attribute of f:ajax tag, 74
    example, 81–85
    execute attribute of f:ajax tag, 74–75
    grouping components, 78–79
    immediate attribute of f:ajax tag, 75
    listener attribute of f:ajax tag, 75
    loading JavaScript resource library, 79–81
    monitoring events, 75–76
Ajax (Continued)
  onerror attribute of f:ajax tag, 76–77
  onevent attribute of f:ajax tag, 75–76
  overview, 70
  receiving responses, 77–78
  render attribute of f:ajax tag, 77–78
  request lifecycle, 78
  sending requests, 73–75
  using JavaScript API directly, 80–81
  using with Facelets, 71–73
  using with JavaServer Faces technology, 69–85
alternatives
  CDI, 251–253
  example, 265–270
annotations, 3
  interceptor metadata, 454
  JAX-RS, 193–196
Ant tool, 40–41
appclient tool, 35
applet container, 15
applets, 9, 10
application client container, 15
application clients, 8–9
  securing, 331–332
application clients, JMS
  building, 383, 386, 390
  examples, 228, 378–405
  packaging, 392
  running, 384–386, 386–388, 390–392, 392–393
  running on multiple systems, 398–405
asadmin tool, 35
asynchronous message consumption, 348
  See also message-driven beans
  JMS client example, 388–393
asynchronous method invocation
  calling asynchronous business
    methods, 243–244
  cancelling, 243–244
  checking status, 244
  creating asynchronous business methods, 242
  example, 244–248
  java.util.concurrent.Future<V>
    interface, 241
  retrieving results, 243
  session beans, 241–248
  authentication
    certificate-based mutual, 317
    client, 316, 319
    mutual, 317–321
    user name/password-based mutual, 318
  AUTO_ACKNOWLEDGE mode, 360
  auto commit, 28

B
  bean-managed transactions, 374
  Bean Validation, 30
    advanced, 449–452
    custom constraints, 449–450
    groups, 451–452
    localization, 450
    messages, 450
    ordering, 451–452
    resource bundles, 450
    bundles, See resource bundles
    BytesMessage interface, 357

C
  CallbackHandler interface, 331
capture-schema tool, 35
CDI, See Contexts and Dependency Injection
  (CDI) for the Java EE platform
certificate authorities, 312
certificates
  client, 320–321
digital, 311–316
  server, 315–316
  server, generating, 312–314
  using for authentication, 314
character encodings, 188–189
  character sets, 188
  CLIENT_ACKNOWLEDGE mode, 360
client certificates, generating, 320–321
  client ID, for durable subscriptions, 364
  clients
    authenticating, 316, 319
Index

securing, 331–332
commit method (JMS), 366–368
component binding, 133, 137–138
  binding attribute, 133, 137
component-managed sign-on, 332, 333
component rendering model, 57, 60
  decode method, 53, 111, 119, 124
  decoding, 99, 106
  delegated implementation, 99
  direct implementation, 99
  encode method, 125
  encodeBegin method, 109
  encodeChildren method, 109
  encodeEnd method, 109, 115
  encoding, 99, 106
HTML render kit, 120, 165
render kit, 60
Renderer class, 60
RenderKit implementation, 165
Renderer implementation, 165
RenderKit class, 60
RenderKit implementation, 165
component tag attributes
  action attribute, 103
  actionListener attribute, 102
  alt attribute, 102
  binding attribute, 133, 137
  converter attribute, 126
  immediate attribute, 103
  rendered attribute, 137
  value attribute, 103, 133, 134–136
  var attribute, 187
component tags, 60, 62
  h:graphicImage tag, 102
composite components
  advanced features, 87–93
  attributes, 87–88
  default attribute, 87
  example, 89–93
  f:validateBean tag, 89
  f:validateRegex tag, 89
  f:validateRequired tag, 89
  invoking managed beans, 88
  method-signature attribute, 88
  name attribute, 87
  required attribute, 87
  type attribute, 88
  validating values, 89
concurrent access to entity data, 297–299
conditional HTTP requests, JAX-RS, 199–200
configuring JavaServer Faces applications
  Application class, 145
  configuration resource files, 144–146
  configuring managed beans, 142–143, 146–155
  configuring navigation rules
    See configuring navigation rules
element attribute, 162
faces-config.xml files, 162
error message registration, 128
including the classes, pages, and other resources, 173
javax.faces.application.CONFIG_FILES context parameter, 144
registering custom converters, 160
registering custom renderers, 165–167
registering custom UI components, 167–168
registering custom validators, 159–160
registering messages, 155–158
specifying where UI component state is saved, 114
configuring managed beans, 103, 146–155
configuring navigation rules, 64, 161–164
  from-action element, 162
  from-view-id element, 162
  navigation-case element, 162, 163
  navigation-rule element, 162
to-view-id element, 162
connection factories, 350
  creating, 231, 381–382
  injecting resources, 228, 350
  specifying for remote servers, 400–401
Connection interface (JMS), 351
ConnectionFactory interface (JMS), 350
connections, JMS
  introduction, 351
  managing in enterprise bean applications, 370
connectors, See Java EE Connector architecture
container-managed sign-on, 332, 333
containers, 13–15
See also EJB container
application client, 15
configurable services, 13
nonconfigurable services, 13
services, 13
web, 14
Contexts and Dependency Injection (CDI) for the
Java EE platform, 29
advanced topics, 251–264
alternatives, 251–253
converting managed beans to JAX-RS root
resource classes, 199
decorators, 262–263
disposer methods, 256
events, 257–260
examples, 265–289
integrating with JAX-RS, 198–199
interceptors, 260–262
observer methods, 258–259
producer fields, 254–256
producer methods, 254–256
specialization, 253
stereotypes, 263–264
conversion model, 57, 61
See also converters
converter attribute, 126
Converter implementations, 61, 126
Converter interface, 124
converting data between model and
presentation, 61
model view, 124
presentation view, 124
carver tags, f: converter tag, 126
converters, 53, 57
custom converters, 61, 126–127
standard, 61
converting data, See conversion model
createBrowser method, 394
criteria queries, string-based, 293–295
cryptography, public-key, 312
custom converters
binding to managed bean properties, 138–139
creating, 123–127
getAsObject(FacesContext, UIComponent, String) method, 124
getAsObject method, 124
getAsString(FacesContext, UIComponent, Object) method, 124
getAsString method, 125
registering, 160
using, 126–127
custom objects
See also custom renderers, custom tags, custom
UI components, custom validators
custom converters, 126–127
using, 121–123
using custom components, renderers and tags
together, 99–100
custom renderers
creating the Renderer class, 115–116
determining necessity of, 98–99
performing decoding, 111–112
performing encoding, 109–111
registering with a render kit, 165–167
custom tags, 64, 99–100
getRendererType method, 117
identifying the renderer type, 115
specifying, 131
tag library descriptor, 105, 131
custom UI components
creating, 95–139
creating component classes, 106–114
custom objects, 121
delegating rendering, 114–117
determining necessity of, 97–98
handling events emitted by, 119–120
queueEvent method, 111
registering
See registering custom UI components
restoreState(FacesContext, Object)
method, 113–114
saveState(FacesContext) method, 113–114
saving state, 113–114
steps for creating, 105–106
custom validators, 128–133
binding to managed bean properties, 138–139
custom validator tags, 131
domains, 41
downloading, GlassFish Server, 38
DUPS_OK_ACKNOWLEDGE mode, 361
durable subscriptions, 364–366
extamples, 409–410, 416–421

e

eager attribute, managed beans, 143
EAR files, 17
EIS tier, 12
  security, 332–336
EJB container, 14
  See also embedded enterprise bean container
message-driven beans, 370–372
onMessage method, invoking, 230–231
EL, method-binding expressions, 65
embedded enterprise bean container
  See also EJB container, enterprise beans
creating, 237–238
developing applications, 236
examples, 239–240
initializing enterprise bean modules, 237–238
overview, 235
running applications, 236
session bean references, 238
shutting down, 238
enterprise applications, 3
enterprise beans, 11, 25–26
  See also embedded enterprise bean container
converting to JAX-RS root resource
classes, 198–199
finding, 238
implémentor of business logic, 11
integrating with JAX-RS, 198–199
interceptors, 453–461
testing, 239–240
Enterprise Information Systems, See EIS tier
entities, controlling caching, 304–305
entity data
lock modes, 299–302
optimistic locking, 297, 298–299
pessimistic locking, 298, 301–302
event and listener model, 57, 61–63

data encryption, 316
databases, EIS tier, 6
debugging, Java EE applications, 45–46
decorators
  CDI, 262–263
e  example, 286–289
delivery modes, 361–362
  JMSDeliveryMode message header field, 356
DeliveryMode interface, 361–362
Dependency Injection for Java (JSR 330), 29–30
deployer roles, 21
deployment descriptors, 17
  Java EE, 18
  runtime, 18
  web application, 168
Destination interface, 350–351
destinations, 350–351
  See also queues, temporary destinations, topics
creating, 231, 381–382
injecting resources, 228, 350
JMSDestination message header field, 356
temporary, 363, 424, 438–439
development roles, 19–21
application assemblers, 20
application client developers, 20
application component providers, 20
application deployers and administrators, 21
enterprise bean developers, 20
Java EE product providers, 19
tool providers, 19
web component developers, 20
digital signatures, 312
disposer methods, CDI, 256
event and listener model (Continued)
See also action events
binding listeners to managed bean properties, 138–139
Event class, 62
event handlers, 53, 105
event listeners, 53, 54, 55
handling events of custom UIcomponents, 119–120
implementing event listeners, 117–119
Listener class, 62
queueEvent method, 111
events
CDI, 257–260
example, 280–286
examples, 37–46
Ajax, 81–85
asynchronous method invocation, sessionbeans, 244–248
building, 43
CDI, 265–289
composite components, 89–93
connectors, 463–467
directory structure, 44
Duke's Bookstore case study, 471–478
Duke's Forest case study, 491–511
Duke's Tutoring case study, 479–489
embedded enterprise bean container, 239–240
file upload using servlets, 177–181
interceptors, 460–461
Java EE Connector architecture, 463–467
JAX-RS, 209–223
JMS asynchronous message consumption, 388–393
JMS durable subscriptions, 409–410
JMS local transactions, 411–416
JMS message acknowledgment, 406–408
JMS message-driven beans, 227–233
JMS on multiple systems, 398–405, 429–436, 436–447
JMS queue browsing, 394–398
JMS synchronous message consumption, 378–388
JMS with entities, 421–429
JMS with session beans, 416–421
message-driven beans, 227–233
required software, 37–41
resource adapters, 463–467
exceptions, JMS, 358
expiration of JMS messages, 362–363
JMSExpiration message header field, 356

F
Facelets
f:ajax tag, 71–73
using Ajax with, 71–73
Facelets applications
lifecycle, 56–57
using JavaScript in, 80–81
faces-config.xml file, 144–146
FacesContext class, 52, 123
Apply Request Values phase, 53
custom converters, 124
performing encoding, 110
Process Validations phase, 54
Update Model Values phase, 55
Validator interface, 129
form parameters, JAX-RS, 195–196

G
getPart method, 176–177
getParts method, 176–177
getRollbackOnly method, 374
GlassFish Server
downloading, 38
enabling debugging, 46
installation tips, 38
server log, 45–46
starting, 41–42
stopping, 41
tools, 35–36
H
handling events, See event and listener model
HTTP, over SSL, 316
HTTPS, 312

I
implicit objects, 136–137
binding component values to, 136–137
InitialContext interface, 32
initializing properties with the managed-property element
initializing Array and List properties, 153
initializing managed-bean properties, 153–155
initializing Map properties, 152–153
initializing maps and lists, 155
referencing a context initialization parameter, 151–152
interceptors, 453–461
CDI, 260–262
classes, 454
example, 460–461
example (CDI), 280–286
lifecycle, 454
using, 455–459
internationalization, 183–189
internationalizing JavaServer Faces applications
f:loadBundle tag, 187
using the FacesMessage class to create a message, 157
ISO 8859 character encoding, 188

J
JAAS, 34, 331–332
login modules, 332
JACC, 31
JAF, 33
JAR files, 17
JASPIC, 31
Java API for XML Binding (JAXB), 33
using with JAX-RS, 202–208
Java API for XML Processing (JAXP), 33
Java API for XML Web Services, See JAX-WS
Java Authentication and Authorization Service, See JAAS
Java Authentication Service Provider Interface for Containers (JASPIC), 31
Java Authorization Contract for Containers, See JACC
Java BluePrints, 44
Java Database Connectivity API, See JDBC API
Java DB, 35
starting, 43
stopping, 43
Java EE applications, 6–12
debugging, 45–46
running on more than one system, 429–436, 436–447
tiers, 6–12
Java EE clients, 8–9
See also application clients
web clients, 8
Java EE components, 8
Java EE Connector architecture, 30
example, 463–467
Java EE modules, 17, 18
application client modules, 19
EJB modules, 18
resource adapter modules, 19
web modules, 18
Java EE platform
APIs, 21–31
JMS and, 344
overview, 4–5
Java EE security model, 13
Java EE servers, 14
Java EE transaction model, 13
Java Message Service (JMS) API, See JMS
Java Naming and Directory Interface API, 32–33
See also JNDI
Java Persistence API, 28
Java Servlet technology, 26
See also servlets
Java Transaction API, 28
JavaBeans Activation Framework (JAF), 33
JavaBeans components, 9
Index

JavaMail API, 31
  example, 463–467

JavaServer Faces applications
  configuring
    See configuring JavaServer Faces applications
JavaServer Faces core tag library
  f:actionListener tag, 98
  f:ajax tag, 71–73
  f:converter tag, 126
  f:validator tag, 64, 128
    custom validator tags, 131
JavaServer Faces standard HTML render kit
  library, 60, 165
  html_basic TLD, 120
JavaServer Faces standard HTML tag library, See
  component tags
JavaServer Faces standard UI components, 58, 95
  UIComponent component, 125
JavaServer Faces technology, 10, 26–27
  See also component rendering model,
    component tags, conversion model, event
    and listener model, FacesContext class,
    JavaServer Faces standard UI components,
    lifecycle of a JavaServer Faces application, UI
    component behavioral interfaces, UI
    component classes, validation model
  composite components, 87–93
  FacesServlet class, 169
  partial processing, 56
  partial rendering, 56
  partial state saving, 107, 113–114
  using Ajax with, 69–85
JavaServer Pages Standard Tag Library, See JSTL

JAX-RS, 29
  accessing XML documents, 202–208
  advanced features, 193–223
  annotations, 193–196
  conditional HTTP requests, 199–200
  converting CDI managed beans to root resource
    classes, 199
  converting enterprise beans to root resource
    classes, 198–199
  examples, 209–223
  extracting Java type of request or response, 196
  form parameters, 195–196
  integrating with CDI, 198–199
  integrating with EJB technology, 198–199
  path parameters, 194–195
  query parameters, 195
  request headers, 193–196
  resource class methods, 197–198
  runtime content negotiation, 200–202
  runtime resource resolution, 197–198
  static content negotiation, 200–202
  subresource locators, 197
  subresource methods, 197
  subresources, 197–198
  URI, 193–196
  using JSON representations, 208
  using with JAXB, 202–208

JAX-WS, 34

JAXB, 33
  examples, 209–223
  generating Java entity classes from XML
    schema, 206–207
  generating XML schema from Java
    classes, 204–205
  Java entity classes, 204–205
  returning Java entity classes in JAX-RS
    resources, 204–205
  schema generator, 204–205
  using with JAX-RS, 202–208
  with JSON, 208
  xjc schema compiler tool, 206–207

JAXBElement, in JAX-RS resource
  methods, 206–207

JAXP, 33

JDBC API, 32

JMS, 30
  achieving reliability and performance, 359–368
  administered objects, 349–351
  application client examples, 378–405
  architecture, 345
  basic concepts, 345–348
  definition, 342
  examples, 227–233, 377–447
  introduction, 341–344
  Java EE platform, 344, 368–375
messaging domains, 346–348
programming model, 348–358
JMS CorrelationID message header field, 356
JMS DeliveryMode message header field, 356
JMS Destination message header field, 356
JMS Exception class, 358
JMS Expiration message header field, 356
JMS MessageID message header field, 356
JMS Priority message header field, 356
JMS Redelivered message header field, 356
JMS ReplyTo message header field, 356
JMS Timestamp message header field, 356
JMS Type message header field, 356
JNDI, 32–33
data source naming subcontexts, 33
enterprise bean naming subcontexts, 33
environment naming contexts, 33
jms naming subcontext, 350
namespace for JMS administered objects, 349–351
naming contexts, 32
naming environments, 32
naming subcontexts, 32
jsf.js file, 79–81
JSON, with JAXB and JAX-RS, 208
JSTL, 28
JTA, 28
JUnit, 239–240
getRendererType method (Render Response phase), 117
immediate attribute, 103
Invoke Application phase, 55
performing encoding (Render Response phase), 109
Process Validations phase, 54
Render Response phase, 55–56
renderResponse method, 52, 53, 54, 55
responseComplete method, 52, 54, 55
Restore View phase, 53
saving state, 114
Update Model Values phase, 55
updateModels method, 55
Validator interface, 130
views, 53
local transactions, 366–368
localization, 183–189
Bean Validation, 450
log, server, 45–46
login modules, 331–332

K
key pairs, 312
keystores, 311–316
managing, 312
keytool utility, 312

L
lifecycle callback events, intercepting, 457–458
lifecycle of a JavaServer Faces application, 50–56
action and value-change event processing, 62
Apply Request Values phase, 53, 111
custom converters, 124, 125

M
managed bean creation facility, 146–155
managed bean declarations, 103
key-class element, 152
list-entries element, 150
managed-bean element, 147–150, 154
managed-bean-name element, 149
managed-property element, 150–155
map-entries element, 150, 152
map-entry element, 152
null-value elements, 150
value element, 150
managed bean properties, 133
managed beans
See also value binding
custom component alternative, 98
event and listener model, 62
configuring in JavaServer Faces technology, 142–143
conversion model, 61
managed beans (Continued)
  loading JavaScript, 81
Managed Beans specification, 29
MapMessage interface, 357
message acknowledgment, 360–361
  bean-managed transactions, 375
  message-driven beans, 371
message bodies, 357
message consumers, 353–355
message consumption, 348
  asynchronous, 348, 388–393
  synchronous, 348, 378–388
message-driven beans, 25
  coding, 229–231, 419, 424–425, 439
  examples, 227–233, 416–421, 421–429,
  429–436, 436–447
  introduction, 370–372
  onMessage method, 230–231
  requirements, 229–231
message headers, 356
message IDs, JMSMessageID message header
  field, 356
Message interface, 357
message listeners, 354–355
  examples, 389, 424, 438–439
message producers, 352–353
message properties, 356
message selectors, 355
MessageConsumer interface, 353–355
MessageListener interface, 354–355
MessageProducer interface, 352–353
messages
  integrity, 316
  queueing messages, 155
  using the FacesMessage class to create a
    message, 157
messages, JMS
  body formats, 357
  browsing, 358
  definition, 345
  delivery modes, 361–362
  expiration, 362–363
  headers, 356
  introduction, 355–357
  persistence, 361–362
  priority levels, 362
  properties, 356
messaging, definition, 341–342
messaging domains, 346–348
  common interfaces, 347–348
  point-to-point, 346–347
  publish/subscribe, 347
method binding
  method-binding expressions, 65, 163
  method expressions, 112
  method expressions, 62
method invocations, intercepting, 456–457
mutual authentication, 317–321

N
naming contexts, 32
naming environments, 32
navigation model, 64–67
  action attribute, 103
  action methods, 161
  configuring navigation rules, 161–164
  logical outcome, 161
  NavigationHandler class, 66
NetBeans IDE, 39–40
NON_PERSISTENT delivery mode, 361

O
ObjectMessage interface, 357
objects, administered, 349–351
  creating and removing, 381–383
observer methods, CDI, 258–259
onMessage method
  introduction, 354–355
  message-driven beans, 230–231, 371

P
package-appclient tool, 35
Part interface (Servlet), 176
path parameters, JAX-RS, 194–195
persistence
  concurrent access to entity data, 297–302
  JMS example, 421–429
  JMS messages, 361–362
  locking strategies, 297–302
  second-level cache, 303–308
  string-based criteria queries, 293–295
PERSISTENT delivery mode, 361
point-to-point messaging domain, 346–347
  See also queues
POJOs, 4
priority levels, for messages, 362
  JMSPriority message header field, 356
producer fields
  CDI, 254–256
  example, 273–280
producer methods
  CDI, 254–256
  example, 271–273
programming model, JMS, 348–358
providers, JMS, 345
public key certificates, 316
public-key cryptography, 312
publish/subscribe messaging domain
  See also topics
    durable subscriptions, 364–366
    introduction, 347

Q
query parameters, JAX-RS, 195
Queue interface, 350–351
QueueBrowser interface, 358
  JMS client example, 394–398
queues, 350–351
  browsing, 358, 394–398
  creating, 350–351, 381–382
  injecting resources, 228
  temporary, 363, 424

R
realms, certificate, 314
recover method, 361
redelivery of messages, 360, 361
  JMSRedelivered message header field, 356
registering custom converters, 160
  converter element, 160
  render-kit element, 165
  renderer element, 165
registering custom renderers, 165–167
registering custom UI components, 105, 167–168
  component element, 167
registering custom validators, 159–160
  validator element, 159
registering messages, 155–158
  resource-bundle element, 156
reliability, JMS
  advanced mechanisms, 364–368
  basic mechanisms, 359–363
  durable subscriptions, 364–366
  local transactions, 366–368
  message acknowledgment, 360–361
  message expiration, 362–363
  message persistence, 361–362
  message priority levels, 362
  temporary destinations, 363
Remote Method Invocation (RMI), and
  messaging, 341–342
request headers, JAX-RS, 193–196
Request objects, JAX-RS, 201
request/reply mechanism
  JMSCorrelationID message header field, 356
  JMSReplyTo message header field, 356
  temporary destinations and, 363
requests, retrieving a locale, 185
Required transaction attribute, 375
resource adapters, 30
  example, 463–467
security, 334–335
resource bundles, 183
  Bean Validation, 450
resources, JMS, 370
RESTful web services, 29
rollback method (JMS), 366–368
S
SAAJ, 34
schemagen, JAXB, 204–205
schemagen tool, 35
scopes, using in JavaServer Faces technology, 143
security
  application clients, 331–332
callback handlers, 331
  component-managed sign-on, 333
  container-managed sign-on, 333
EIS applications, 332–336
JAAS login modules, 332
login forms, 331
login modules, 331–332
programmatic login, 332
resource adapters, 334–335
send method, 352–353
server authentication, 316
server certificates, 311–316
server log, 45–46
servers, Java EE
  deploying on more than one, 429–436, 436–447
  running JMS clients on more than one, 398–405
servlets, 10
  uploading files with, 175–181
session beans, 25
  See also asynchronous method invocation examples, 416–421
Session interface, 352
sessions, JMS, 352
  managing in enterprise bean applications, 370
setRollbackOnly method, 374
sign-on
  component-managed, 332, 333
  container-managed, 332, 333
SOAP messages, 16, 34
SOAP with Attachments API for Java (SAAJ), 34
SQL, 32
SSL, 316
standard converters, 61
standard validators, 63
stereotypes, CDI, 263–264
StreamMessage interface, 357
string-based criteria queries, 293–295
subresources, JAX-RS, 197–198
subscription names, for durable subscribers, 364
synchronous message consumption, 348
  JMS client example, 378–388
T
temporary JMS destinations, 363
  examples, 424, 438–439
testing
  enterprise beans, 239–240
  unit, 239–240
TextMessage interface, 357
timeout events, intercepting, 458–459
timestamps, for messages, JMSTimestamp message header field, 356
Topic interface, 350–351
topics, 350–351
  creating, 350–351, 381–382
durable subscriptions, 364–366
temporary, 363, 438–439
transactions
  bean-managed, 374
  container-managed, 373
distributed, 373–375
  examples, 411–416
JMS and enterprise bean applications, 370
local, 366–368
Required attribute, 375
truststores, 311–316
  managing, 312
U
UI component behavioral interfaces, 59
  ActionSource interface, 59, 62, 106, 117
  ActionSource2 interface, 59, 106
  ClientBehaviorHolder interface, 59
  ConvertibleValueHolder interface, 59
  EditableValueHolder interface, 59, 107
NamingContainer interface, 59, 107
StateHelper interface, 107, 112, 113–114
StateHolder interface, 59, 107, 113–114
SystemEventListenerHolder interface, 59
ValueHolder interface, 59, 107
UI component classes, 58, 60, 97
See also custom UI components
javax.faces.component package, 106
UIColumn class, 58
UICommand class, 58, 60
UIComponent class, 57, 60
UIComponentBase class, 58, 106, 109
UIData class, 58
UIForm class, 58
UIGraphic class, 58
UIInput class, 58, 62
UIMessage class, 58
UIMessages class, 58
UIOutput class, 58, 61
UIPanel class, 58
UIParameter class, 58
UISelectBoolean class, 58
UISelectItem class, 58
UISelectItems class, 58
UISelectMany class, 58
UISelectOne class, 58, 60
UIViewRoot class, 59
Unicode character set, 188
US-ASCII character set, 188
UserTransaction interface, message-driven beans, 374
UTF-8 character encoding, 189

V
validating input, See validation model
validation
   customizing, 449–450
groups, 451–452
localization, 450
messages, 450
ordering, 451–452
validation model, 57, 63–64
See also validators
Validator implementation, 63, 132
Validator interface, 64
custom validator tags, 131
implementing, 129
Validator implementation classes, 63
validator tags
   composite components, 89
f:validator tag, 64, 131
validators, 53, 57
custom validators, 132–133
default, 159
value binding
   component instances to bean properties
      See component binding
component values and instances to managed bean properties, 133–138
component values to implicit objects, 136–137
component values to managed bean properties, 134–136
value attribute, 103, 133, 134–136
value-binding expressions, 134
value expressions, 112, 137
value-change events, 62, 117
f:valueChangeListener tag, 98
processValueChange(ValueChangeEvent) method, 117
ValueChangeEvent class, 117
ValueChangeListener class, 117
ValueChangeListener implementation, 117
Variant class, JAX-RS, 201

W
W3C, 33
WAR files, 17
web applications
   establishing the locale, 185
   internationalizing and localizing, 183
   parsing and formatting localized dates and numbers, 187
   providing localized messages, 184
   retrieving localized messages, 186
   setting the resource bundle, 186
web clients, 8
Index

web components, 10
  applets bundled with, 10
  JMS and, 375
  types, 10
  utility classes bundled with, 10
web container, 14
web modules, 18
web services, 15–16
web.xml file, 168
WSDL, 16
wsgen tool, 36
wsimport tool, 36

X
xjc schema compiler tool, JAXB, 206–207
xjc tool, 35
XML, 15–16