First Edition (January 2000)

This edition applies to Version 3.0 of WebSphere Application Server Advanced Edition for AS/400, Program Number 5733-WA2 or 5733-WA3 for use with the OS/400.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JLU  Building 107-2
3605 Highway 52N
Rochester, Minnesota 55901-7829

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 1999. All rights reserved.
Note to U.S Government Users - Documentation related to restricted rights - Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.
Contents

Figures .......................................................... vii
Tables ............................................................. xiii
Preface ............................................................. xv
The Team That Wrote This Redbook ............................... xv
Comments Welcome ........................................... xvi

Chapter 1. WebSphere Advanced Edition for AS/400 .............................. 1
  1.1 WebSphere .................................................. 1
  1.2 Overview of IBM WebSphere Application Server Advanced Edition 3.0 .................................. 2
  1.3 AS/400 Specific considerations .................................. 3
    1.3.1 International and North American Editions ................. 4
    1.3.2 Install Options ........................................ 4
    1.3.3 HTTP Server Support .................................. 4
    1.3.4 Database Support ..................................... 5
    1.3.5 Running the Administration Client ....................... 5
    1.3.6 Multi-instance Support ................................ 5
  1.4 Installation of WebSphere Application Server Advanced Edition 3.0 .............. 6
    1.4.1 AS/400 Installation Options ............................ 7
    1.4.2 Installing WebSphere Advanced Edition from the AS/400 System .......... 8
    1.4.3 Installing WebSphere from a PC ........................ 11
    1.4.4 Installing the Administration Client on a PC ........... 16
  1.5 Un-Installing WebSphere Application Server Advanced Edition 3.0 .............................. 17
  1.6 Configuring the WebSphere software .................................... 17
    1.6.1 Administrative Terminology ................................ 17
    1.6.2 Relationships among Administered Resources ............ 18
    1.6.3 Configuring Multiple Administrative Servers .............. 19
    1.6.4 Configuring an Administrative Server without a Sample Environment .... 20
    1.6.5 Exporting an Administrative Server Configuration ........ 20
    1.6.6 Importing an Administrative Server Configuration ........ 20
  1.7 Starting and Stopping the WebSphere Application Server .................. 20
    1.7.1 Start-up ............................................... 21
    1.7.2 Shutdown .............................................. 22
  1.8 Starting the Administrative Console ................................ 22
    1.8.1 Using the Start menu shortcut .......................... 23
    1.8.2 Using a command prompt ................................ 23
  1.9 DataSources .................................................. 24
    1.9.1 Creating and Installing a JDBC Driver .................... 24
    1.9.2 Creating a DataSource .................................. 28
    1.9.3 Adding DataSource information to an EJB Container ........ 29

Chapter 2. AS/400 WebSphere Servlet and JSP Development ...................... 33
  2.1 Servlet Support in WebSphere Advanced Edition 3.0 ........................ 33
  2.2 IBM development environments for WebSphere applications ................. 34
  2.3 To Port or not to Port, that is the Question ................................ 34
    2.3.1 Keeping Servlets at the Sun 2.0 Specification Level ........ 35
  2.4 Migrating Servlets ............................................ 38
    2.4.1 Migrating from IBM Connection Manager to DataSources ........ 38
    2.4.2 Migrating WebSphere Version 2.0 Servlets ................ 45
    1.9.3 Adding DataSource information to an EJB Container ........ 29
2.4.4 Summary of Migrating to the JSWDK 1.0 Specification ............ 59
2.5 JSP Support in WebSphere Version 3.0 ............................... 60
  2.5.1 To Port or not to Port, that is the Question ..................... 61
  2.5.2 Leaving JSPs at the Sun 0.91 Specification Level ............... 61

Chapter 3. Introduction to Enterprise JavaBeans ......................... 63
  3.1 The Enterprise JavaBeans (EJB) Specification ..................... 63
  3.2 Enterprise JavaBeans Architecture Definition ...................... 64
  3.3 How EJB Technology Fits into the Java Architecture ............. 65
  3.4 Why EJB is Important ............................................. 65
  3.5 Leveraging Java and EJB Technology ............................... 66
  3.6 EJB Architectural Overview ....................................... 67
    3.6.1 The EJB Server .................................................. 68
    3.6.2 Types of Components .......................................... 70
    3.6.3 Component Content ............................................. 71
  3.7 EJB Roles ............................................................ 72
    3.7.1 Enterprise JavaBean Provider ................................ 73
    3.7.2 Application Assembler ......................................... 73
    3.7.3 Application Deployer .......................................... 74
    3.7.4 Server Provider ................................................ 74
    3.7.5 Container Provider ............................................. 74
  3.8 Using EJB Technology on the AS/400 System ......................... 75
    3.8.1 AS/400 Java Overview ......................................... 75
    3.8.2 EJB Technology on the AS/400 System .......................... 76
  3.9 EJB Interoperability with Other Distributed Object Architectures 76
    3.9.1 CORBA ............................................................ 77
    3.9.2 COM ............................................................... 77
    3.9.3 IBM SanFrancisco ............................................. 77
    3.9.4 IBM Component Broker ....................................... 78
  3.10 Conclusion .......................................................... 78

Chapter 4. Overview of the Order Entry Application ..................... 79
  4.1 Overview of the Order Entry Application ........................... 79
    4.1.1 The ABC Company ................................................ 79
    4.1.2 The ABC Company Database .................................... 80
    4.1.3 A Customer Transaction ...................................... 80
    4.1.4 Application Flow ............................................... 81
    4.1.5 Customer Transaction Flow .................................... 82
    4.1.6 Database Table Structure .................................... 88
    4.1.7 Order Entry Application Database Layout .................... 88
    4.1.8 Database Terminology .......................................... 91

Chapter 5. Developing Enterprise JavaBeans ............................... 93
  5.1 Generating a CMP Entity Bean from an existing Database File ...... 93
    5.1.1 Importing a Schema ............................................ 93
    5.1.2 Creating an EJB Group ........................................ 98
    5.1.3 Deploying the Stock Entity Bean ............................... 107
    5.1.4 Exporting to the AS/400 System ................................ 113
    5.1.5 Installing a JAR file in the WebSphere Environment .......... 115
    5.1.6 Starting the Stock Bean ....................................... 123
    5.1.7 Creating a Test Client ....................................... 125
    5.1.8 Testing the Stock Bean ....................................... 126
  5.2 Developing a CMP Entity Bean with Mapping to an Existing Table .. 134
    5.2.1 Mapping the Stock Bean ..................................... 143
5.3 Developing a CMP Entity Bean and Generating Data for Mapping . . . . . 149
  5.3.1 Deployment in the WebSphere Environment ........................ 149
  5.3.2 Installing in the WebSphere Environment .......................... 151
  5.3.3 Table Creation and Deployment with VisualAge for Java .......... 158
5.4 Developing a BMP Entity Bean ........................................... 165
5.5 Deployment ...................................................................... 167

Chapter 6. AS/400 EJB Application Development Scenario ............... 169
  6.1 Order Entry Application Architecture with Objects .................. 169
  6.2 Business Data - Entity Enterprise JavaBeans ........................ 172
    6.2.1 Database Access - Using a Connection Pool .................... 172
    6.2.2 Persistence - Container or Bean Managed ....................... 175
    6.2.3 Container Managed Persistence .................................... 175
    6.2.4 Bean Managed Persistence ........................................ 175
  6.3 Business Processes - Session Enterprise JavaBeans ............... 176
    6.3.1 Three-Tier Versus Two-Tier Architecture ........................ 176
    6.3.2 Stateless or Stateful ............................................. 177
    6.3.3 Order Entry Example ............................................. 177
    6.3.4 Stateless Session Bean - OrderPlacement ......................... 177
    6.3.5 Stateful Session Bean - OrderEntryClerk ......................... 184
  6.4 Transactions ............................................................. 188
    6.4.1 Isolation Level .................................................... 189
    6.4.2 Transaction Attribute ............................................. 189
  6.5 Conclusion ............................................................... 189

Chapter 7. Building Java Applications with Enterprise JavaBeans .... 191
  7.1 Building Java Applications with VisualAge for Java .............. 191
    7.1.1 The ItemsDb Class ............................................... 192
    7.1.2 Using the ItemsDb Class ........................................ 199
  7.2 Building the Order Entry Application ................................ 200
    7.2.1 Application Flow Through the Java Client Order Entry Window 201
    7.2.2 Connecting to the Database ...................................... 202
    7.2.3 Retrieving the Customer List .................................. 204
    7.2.4 Retrieving the Item List ........................................ 209
    7.2.5 Verifying the Item and Adding It to the Order ................ 213
    7.2.6 Submitting the Order .......................................... 214
  7.3 Building Servlets ....................................................... 217
    7.3.1 Guaranteeing One Web Client per Session Bean ............... 220

Chapter 8. AS/400 Unique Considerations ................................ 223
  8.1 Java on the AS/400 System ............................................. 223
    8.1.1 Java Files in the Integrated File System ....................... 223
    8.1.2 Accessing AS/400 Resources from Java ........................ 223
    8.1.3 Java Commands and Tools ...................................... 223
  8.2 Java Performance Considerations .................................... 224
    8.2.1 General Java Performance Techniques .......................... 224
    8.2.2 AS/400-Specific Java Optimization ............................. 225
  8.3 DB2/400 Considerations .............................................. 226
    8.3.1 Native JDBC Driver versus AS/400 Toolbox JDBC Driver .... 226
    8.3.2 Journaling ....................................................... 229
  8.4 Debugging WebSphere Applications on the AS/400 System ........... 229
    8.4.1 Retrieving Java Source Files .................................. 230
# Figures

1. Using EJB components ................................................. 3
2. WebSphere Application Server ....................................... 5
3. Installing the base option ........................................... 8
4. Base Installation completion ........................................ 9
5. Installing the Server option ....................................... 10
6. Server installation completion ..................................... 11
7. Initial remote installation ......................................... 12
8. Install sign on ..................................................... 13
9. WebSphere components .............................................. 13
10. Installation directory ............................................... 14
11. Install progress .................................................... 15
12. Installation complete ............................................... 15
13. Installing the Administration Console .......................... 16
14. Administrative console ............................................ 19
15. WebSphere Application Server environment ....................... 21
16. WebSphere Advanced Administrative Console ...................... 25
17. Create a JDBCDriver .................................................. 25
18. Installing the driver ................................................ 26
19. Install Driver ....................................................... 27
20. Selecting the driver jar file ...................................... 27
21. Install driver ........................................................ 27
22. WebSphere Advanced Administrative Console ...................... 28
23. Create a DataSource .................................................. 28
24. DataSource Advanced settings pane ............................... 29
25. WebSphere Advanced Administrative Console ...................... 30
26. Select a DataSource ................................................ 30
27. DataSource tab ..................................................... 31
28. WebSphere V3: Typical Servlet scenarios ......................... 33
29. IBMJbdcConnSpec example code .................................. 36
30. Servlet chaining in WebSphere Standard Edition 2.0 .......... 37
31. Connection Manager Example: imports and variable definitions ........................................... 39
32. Retrieving a JDBC connection from the Connection Manager .............................................. 40
33. Returning a JDBC connection to the caller ........................................... 41
34. Validating ownership of the JDBC connection ........................................... 42
35. Connection Manager Example: Auxiliary methods ................. 42
36. DataSource Example: imports and variable definitions ........................................... 43
37. DataSource Example: Retrieving a JDBC connection from the DataSource ........................................... 44
38. DataSource Example: getConnection method ........................................... 45
39. DataSource Example: Auxiliary methods ........................................... 45
40. Application Architecture ............................................ 46
41. VAJ V3 package containing the AS/400 Toolbox classes ...... 48
42. VAJ V3 problem pane for V2 Servlets ........................................... 49
43. V2 Class Hierarchy utilizing the IBM Connection Manager classes ........................................... 50
44. Import statements .................................................... 50
45. The freeConnection method ...................................... 51
46. The getConnection method ........................................ 51
47. The JDBCPoolCatalog class ....................................... 52
48. The connectToDB method ........................................ 52
49. The getAllV method ............................................... 53
50. The getRecord method ............................................. 54
51. The init method ................................................................. 55
52. The outputItemInformation method .................................... 56
53. Servlet and JSP architecture .............................................. 56
54. Part 1 of the doPost method of the ItemJSPServlet ................. 57
55. Part 2 of the doPost method of the ItemJSPServlet ............... 58
56. EJB Architecture Overview ............................................. 67
57. EJB System Services Providers ......................................... 68
58. The EJB Server ............................................................. 69
59. Deployment Descriptors .................................................. 71
60. EJB Provider ............................................................... 73
61. EJB Assembler and Deployer .......................................... 74
62. Java Environment on the AS/400 System ............................. 76
63. The Company Structure ................................................ 79
64. RPG Application Flow .................................................. 81
65. Parts Order Entry ......................................................... 82
66. Select Customer .......................................................... 83
67. Parts Order Entry ........................................................ 84
68. Select Part ............................................................... 85
69. Parts Order Entry ......................................................... 85
70. Parts Order Entry ........................................................ 86
71. Parts Order Entry ........................................................ 86
72. Change Selected Order .................................................. 87
73. Completed Order ........................................................ 87
74. Printed Order ............................................................. 88
75. Table Relationships ..................................................... 88
76. VisualAge for Java Workbench ........................................ 94
77. Schema browser .......................................................... 94
78. Information Required dialog .......................................... 95
79. Database Connection Info ............................................... 95
80. Select Tables dialog ..................................................... 96
81. Schema browser .......................................................... 96
82. Table Editor .............................................................. 97
83. Schema browser .......................................................... 97
84. VisualAge for Java Workbench ........................................ 98
85. Create EJB Group SmartGuide ......................................... 99
86. Workbench EJB tab ........................................................ 100
87. Property pane ............................................................. 101
88. Removing methods ....................................................... 102
89. Removing methods ....................................................... 103
90. Checking removed methods ............................................ 104
91. Adding ejbCreate() methods .......................................... 105
92. Adding create() method to home interface ........................ 105
93. Adding create() method to home interface ........................ 106
94. Displaying the home interface methods ............................. 106
95. Generating the deployed code ......................................... 107
96. Generated classes ........................................................ 108
97. Changing properties ..................................................... 109
98. Properties panel .......................................................... 109
99. Isolation levels ............................................................ 111
100. Run-As_Mode values ...................................................... 111
101. Adding method control descriptors .................................. 112
102. Selecting the method to change ..................................... 112
103. Adding a control descriptor for a method ........................ 113
<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>Environment properties</td>
</tr>
<tr>
<td>105</td>
<td>Exporting a deployed Jar file</td>
</tr>
<tr>
<td>106</td>
<td>Export to a Deployed JAR file SmartGuide</td>
</tr>
<tr>
<td>107</td>
<td>WebSphere Advanced Administrative Console</td>
</tr>
<tr>
<td>108</td>
<td>Deployment dialog</td>
</tr>
<tr>
<td>109</td>
<td>Finding the JAR file</td>
</tr>
<tr>
<td>110</td>
<td>Confirm dialog</td>
</tr>
<tr>
<td>111</td>
<td>Deployment dialog</td>
</tr>
<tr>
<td>112</td>
<td>Information dialog</td>
</tr>
<tr>
<td>113</td>
<td>Topology tree of WebSphere Application Server Advanced Edition 3.0</td>
</tr>
<tr>
<td>114</td>
<td>Stock enterprise bean DataSource tab</td>
</tr>
<tr>
<td>115</td>
<td>Deployment properties</td>
</tr>
<tr>
<td>116</td>
<td>Entity tab</td>
</tr>
<tr>
<td>117</td>
<td>Configure resource security</td>
</tr>
<tr>
<td>118</td>
<td>Transaction tab</td>
</tr>
<tr>
<td>119</td>
<td>Isolation tab</td>
</tr>
<tr>
<td>120</td>
<td>Environment tab</td>
</tr>
<tr>
<td>121</td>
<td>Starting the Stock enterprise bean</td>
</tr>
<tr>
<td>122</td>
<td>Stock bean running</td>
</tr>
<tr>
<td>123</td>
<td>Creating a test client</td>
</tr>
<tr>
<td>124</td>
<td>Running the Test client</td>
</tr>
<tr>
<td>125</td>
<td>Connect dialog</td>
</tr>
<tr>
<td>126</td>
<td>Home interface page</td>
</tr>
<tr>
<td>127</td>
<td>Home interface constructors</td>
</tr>
<tr>
<td>128</td>
<td>Remote interface page</td>
</tr>
<tr>
<td>129</td>
<td>Displaying the results</td>
</tr>
<tr>
<td>130</td>
<td>Constructors for BigDecimal class</td>
</tr>
<tr>
<td>131</td>
<td>Stock bean test client remote interface page</td>
</tr>
<tr>
<td>132</td>
<td>Stock Test Home interface</td>
</tr>
<tr>
<td>133</td>
<td>Constructors page for BigDecimal class</td>
</tr>
<tr>
<td>134</td>
<td>Remote interface page</td>
</tr>
<tr>
<td>135</td>
<td>Attempting to use a deleted bean</td>
</tr>
<tr>
<td>136</td>
<td>Adding an enterprise bean</td>
</tr>
<tr>
<td>137</td>
<td>Create Enterprise Bean SmartGuide</td>
</tr>
<tr>
<td>138</td>
<td>Define bean class attributes and interfaces SmartGuide</td>
</tr>
<tr>
<td>139</td>
<td>Create CPM Field SmartGuide</td>
</tr>
<tr>
<td>140</td>
<td>Create quantity CMP field</td>
</tr>
<tr>
<td>141</td>
<td>Define Bean Class Attributes and Interfaces SmartGuide</td>
</tr>
<tr>
<td>142</td>
<td>StockBean class</td>
</tr>
<tr>
<td>143</td>
<td>Stock interface</td>
</tr>
<tr>
<td>144</td>
<td>Stock properties</td>
</tr>
<tr>
<td>145</td>
<td>StockKey class</td>
</tr>
<tr>
<td>146</td>
<td>New EJB Group Map</td>
</tr>
<tr>
<td>147</td>
<td>New Datastore Map</td>
</tr>
<tr>
<td>148</td>
<td>Map Browser- add table map</td>
</tr>
<tr>
<td>149</td>
<td>Table Map with no inheritance</td>
</tr>
<tr>
<td>150</td>
<td>Editing a property map</td>
</tr>
<tr>
<td>151</td>
<td>Property Map Editor</td>
</tr>
<tr>
<td>152</td>
<td>Property Map Editor</td>
</tr>
<tr>
<td>153</td>
<td>Property Map Editor</td>
</tr>
<tr>
<td>154</td>
<td>Property Map Editor</td>
</tr>
<tr>
<td>155</td>
<td>Map Browser- Property Map pane</td>
</tr>
<tr>
<td>156</td>
<td>Exporting an EJB JAR file</td>
</tr>
</tbody>
</table>
157. Export to an EJB JAR File SmartGuide .......................... 150
158. Creating an enterprise bean ....................................... 151
159. Deployment dialog .................................................. 152
160. Selecting the JAR file to deploy ............................... 152
161. Confirm dialog ..................................................... 153
162. Confirm dialog ..................................................... 153
163. Enterprise bean deploying message ............................. 153
164. Deploy information dialog ...................................... 154
165. Deployment dialog .................................................. 154
166. Enterprise bean create information dialog ..................... 154
167. Stock enterprise bean in the Topology pane ................. 155
168. Selecting the DataSource tab ..................................... 156
169. Work with Objects using PDM .................................... 156
170. Interactive SQL - Enter SQL Statements ...................... 157
171. Interactive SQL - Display Data ................................ 157
172. DSPFFD for the STOCKBEANTBL table ....................... 157
173. Adding a schema and map from an EJB ........................ 158
174. Schema browser .................................................. 159
175. Editing the Stock table .......................................... 159
176. Table editor ........................................................ 160
177. Creating a database file ......................................... 161
178. EJB server configuration panel .................................. 162
179. Properties for EJB server dialog .............................. 162
180. EJB Server Configuration ......................................... 163
181. Create Table - Problems Occurred dialog .................... 163
182. Display File Field Description output ......................... 164
183. VisualAge for Java - Map Browser .............................. 164
184. Create Enterprise Bean SmartGuide ........................... 165
185. Define Bean Class Attributes and Interfaces SmartGuide ... 166
186. Displaying the BMP Stock enterprise bean .................. 167
187. Application Object Diagram ..................................... 170
188. Object Relationships ............................................. 171
189. Using a DataSource .............................................. 173
190. Setting an environment in VisualAge for Java ............... 174
191. Changing deployment properties in WebSphere .............. 174
192. Code to get an environment variable .......................... 174
193. OrderPlacementBean properties ................................ 178
194. OrderPlacementBean properties ................................ 178
195. OrderPlacement home interface ................................ 179
196. OrderPlacement remote interface .............................. 180
197. OrderPlacementBean ............................................. 181
198. Session Bean Life-Cycle Methods ............................. 181
199. The placeOrder Method Processing ........................... 183
200. OrderEntryClerkBean properties .............................. 184
201. OrderEntryClerkBean properties .............................. 185
202. OrderEntryClerkBean ............................................ 185
203. Defining State Variables ....................................... 186
204. Initializing Values ................................................ 186
205. The setCustomer Method ....................................... 186
206. The addOrderLine Method ...................................... 187
207. The placeOrder Method ......................................... 187
208. OrderPlacement EJB properties ............................... 188
209. Support package .................................................. 191
210.Interfacing to the ItemsDb Class ........................................... 192
211.The ItemsDb Class Description ........................................ 193
212.The getInitialContext Method .......................................... 194
213.The connect Method ....................................................... 194
214.The getAllCustomers Method ........................................... 195
215.The getItems Method ...................................................... 196
216.The findRangeOfItems Method ......................................... 196
217.The verifyCustomer Method ............................................. 197
218.The submitOrder Method .................................................. 197
219.The connectStateless method .......................................... 198
220.The submitOrderStateless method ..................................... 199
221.Testing the ItemsDb Class in the VisualAge for Java Scrapbook 199
222.Java Console Messages ..................................................... 199
223.Retrieving Customers Using the Scrapbook .............................. 200
224.Displaying the Customer File Rows on the Java Console .......... 200
225.Parts Order Entry - Initial Window ..................................... 201
226.The OrderEntry Connection Menu ...................................... 202
227.Sign On to the System ...................................................... 202
228.The Sign-On Dialog in the VisualAge for Java Visual Composition Editor 203
229.OK Button Connections .................................................... 203
230.Order Entry Application with List Customers Button Enabled ........ 204
231.The SltCustWdw Class in the VisualAge for Java Visual Composition Editor 204
232.WindowOpened Event Connections ...................................... 205
233.The populateCustBox Method ............................................ 205
234.The populateCustBox Parameter ........................................ 206
235.Select a Customer .......................................................... 206
236.The custSelected Method ................................................. 207
237.The Customer Class .......................................................... 207
238.Customer Class Constructor .............................................. 208
239.The OrderEntryWdw setSelectedCust Method ......................... 208
240.Order Entry Window with Customer Data ................................ 209
241.The SltItemWdw Class in the VisualAge for Java Visual Composition Editor 209
242.WindowOpened Event Connections ...................................... 210
243.The populateCustBox Method ............................................ 210
244.The populateCustBox Parameter ........................................ 211
245.Select a Customer .......................................................... 211
246.The itemSelected Method ................................................ 212
247.The Item Class .............................................................. 212
248.Item Class Constructor ..................................................... 212
249.The OrderEntryWdw setSelectedItem Method ......................... 213
250.The Order Entry Window with an Item Selected ....................... 213
251.The addOrderItem Method ................................................. 214
252.Parts Order Ready to Submit .............................................. 214
253.The submitOrder Method .................................................. 215
254.A Successful Order Submission .......................................... 215
255.The ItemSession servlet ................................................... 218
256.Add to Cart button .......................................................... 218
257.The servlet shopping cart .................................................. 219
258.CartServlet servlet .......................................................... 219
259.The placeOrder method in the CartServlet .......................... 220
260.ItemSBCServlet .............................................................. 221
261.The getClerk method ........................................................ 221
262.Display Java Program (DSPJVAPGM) Results ....................... 226
263. Typical non-JTA JDBC driver definition .............................................227
264. Typical DataSource definition ..............................................................227
265. Finding the Application Server Job ......................................................228
266. Finding the QSQRVR Job .................................................................228
267. QSQRVR Job Log ...........................................................................229
268. Debugging EJB applications on the AS/400 system ..............................231
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>WebSphere Application Server Advanced Edition 3.0 software</td>
<td>7</td>
</tr>
<tr>
<td>2.</td>
<td>Dynamic DataSource settings</td>
<td>35</td>
</tr>
<tr>
<td>3.</td>
<td>DataSource packages</td>
<td>43</td>
</tr>
<tr>
<td>4.</td>
<td>URL encoding APIs</td>
<td>55</td>
</tr>
<tr>
<td>5.</td>
<td>JSWDK 1.0 APIs</td>
<td>60</td>
</tr>
<tr>
<td>6.</td>
<td>District Table Layout (Dstrct)</td>
<td>89</td>
</tr>
<tr>
<td>7.</td>
<td>Customer Table Layout (CSTMR)</td>
<td>89</td>
</tr>
<tr>
<td>8.</td>
<td>Order Table Layout (ORDERS)</td>
<td>90</td>
</tr>
<tr>
<td>9.</td>
<td>Order Line Table Layout (ORDLIN)</td>
<td>90</td>
</tr>
<tr>
<td>10.</td>
<td>Item Table Layout (ITEM)</td>
<td>91</td>
</tr>
<tr>
<td>11.</td>
<td>Stock Table Layout (Stock)</td>
<td>91</td>
</tr>
<tr>
<td>12.</td>
<td>Database Terminology</td>
<td>92</td>
</tr>
<tr>
<td>13.</td>
<td>EJB isolation levels</td>
<td>110</td>
</tr>
<tr>
<td>14.</td>
<td>AS/400 to Java mapping</td>
<td>138</td>
</tr>
</tbody>
</table>
Preface

WebSphere Application Server Advanced Edition 3.0 is the first of the WebSphere Application Server family to bring a true Java based IBM Enterprise Application environment to the AS/400 system. WebSphere Application Server Advanced Edition 3.0 enables you to utilize Enterprise JavaBeans (EJB) technologies to develop and deploy enterprise wide applications. If you are interested in using your AS/400 system as a Java server, this redbook is for you. It is intended for anyone who wants to use Java, and specifically Enterprise JavaBeans, on the AS/400 system.

EJB technology provides a component model for server applications. It allows you to easily separate user interfaces from business logic. The server-side business logic is packaged as Enterprise JavaBean components. Once they are written and deployed on a server, such as the AS/400 system, client programmers can use them with very little knowledge of how the beans actually work. The client programmer only has to know what methods the Enterprise JavaBeans support and how to call them. Another key advantage is that, whether you are writing a Java application, a Java servlet or even a Visual Basic program, it always works the same. You only need to call the methods provided by the Enterprise JavaBeans to handle the application processing.

In order to effectively use this redbook, you should be already familiar with the Java programming language and the concepts of Enterprise JavaBeans. This redbook focuses on building and deploying Enterprise JavaBeans and using them to access AS/400 resources. It provides many practical programming examples with detailed explanations of how they work. This redbook gives you a fast start on your way to using Enterprise JavaBeans on your AS/400 system.

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization Rochester Center.

**Bob Maatta** is a Senior Software Engineer from the United States at the IBM International Technical Support Organization, Rochester Location. He is the ITSO technical leader for AS/400 e-business application development. He writes extensively and teaches IBM classes worldwide on all areas of AS/400 client/server and e-business application development. Before joining the ITSO in 1995, he worked in the AS/400 National Technical Support Center as a Consulting Market Support Specialist. He has over 20 years of experience in the computer industry. He has worked on numerous computer platforms including S/390, S/38, AS/400 and Personal Computers. He is a Sun Certified Java Programmer and a Sun Certified Java Developer.

**Jeff Blight** is a Senior IT Specialist in the AS/400 Technical Support Group of IBM UK.

**Massimo Marasco** is a Senior IT Specialist working in the AS/400 Technical Support Group of IBM Italy, Segrate location. He is the EMEA South Region AS/400 Java Specialist. He is responsible for supporting customer and business partner AS/400 Java projects in Italy, Spain, Portugal, Greece, Israel and Turkey. Massimo joined IBM in 1980 in a S/34 application development department. He has 19 years
of experience in AS/400 Application Development (RPG, ILE RPG, VisualAge for RPG) and communications programming (APPC, CPI-C, TCP-IP Sockets, DDM, DRDA). He’s been working with Objected Oriented Languages (Smalltalk) since 1994 and with Java for more than two years.

Travis Nelson is a North Dakota State University Computer Science graduate. He is originally from Litchfield, MN, but now lives in Rochester, MN. He is on the e-business technologies and integration team in PartnerWorld for Developers, AS400 and works mostly with IBM WebSphere Application Server, Enterprise JavaBeans (EJB) and Java servlets. His writing has been published in Midrange Computing Magazine and on several websites across the internet.

Thanks to the following people for their invaluable contributions to this project:

Jim Beck
Chris Benson
Mike Koranda
Les Fullem
Larry Hall
Kevin Roberts
Craig Setera
Art Smet
Lisa Wellman

IBM Rochester Laboratory

Comments Welcome

Your comments are important to us!

We want our redbooks to be as helpful as possible. Please send us your comments about this or other redbooks in one of the following ways:

- Fax the evaluation form found in "ITSO Redbook Evaluation" on page 245 to the fax number shown on the form.
- Use the online evaluation form found at http://www.redbooks.ibm.com/
- Send your comments in an internet note to redbook@us.ibm.com
Chapter 1. WebSphere Advanced Edition for AS/400

WebSphere Application Server Advanced Edition 3.0 for AS/400 is the first of the WebSphere Application Server family to bring a true Java based IBM Enterprise Application environment to the AS/400 system. It enables you to utilize Enterprise JavaBeans technologies to develop and deploy your enterprise wide applications.

*** Note ***

We refer to WebSphere Application Server Advanced Edition 3.0 throughout this redbook. The version used when developing this redbook was version 3.02.

Within this chapter, we will cover the following topics:

- Overview of the WebSphere Application Server Advanced Edition 3.0 for AS/400
- AS/400 specific considerations
- Installing of the WebSphere Application Server Advanced Edition 3.0 for AS/400 Software
- Un-installing the WebSphere Application Server Advanced Edition 3.0 software
- Starting and stopping WebSphere Application Server Advanced Edition 3.0
- Creating a DataSource

1.1 WebSphere

WebSphere, a brand name for a set of IBM products, enables the building, deploying, and management of Java applications. WebSphere has the following four main components:

- WebSphere Application Server
- WebSphere Studio
- WebSphere Performance Pack
- WebSphere Site Analysis

The WebSphere Performance Pack provides Web-facilities management software that supports the rapid growth of high-volume Web sites. It brings together, in a single package, caching, load balancing and Web site replication support.

The WebSphere Site Analysis tools provide site administration and analysis tools that can be used to administer and monitor usage of a Web site.

As a PC based set of tools, WebSphere Studio is used to build Web applications. It allows the integration of various technologies, for example, HTML, EJB, JSP (Java Server Pages) and servlets. VisualAge for Java is one of the IBM products included in the suite of tools.

WebSphere Application Server provides the application runtime environment. It includes the following three editions:
Introduction to WebSphere V3 Advanced Edition

The Standard Edition, which provides support for Java servlets, Java Server Pages (JSP) and XML Document Structure Services.


The Enterprise Edition, which enhances the Advanced Edition. It combines TXSeries, transactional application environment provided by IBM, with the full distributed object and business process integration capabilities of Component Broker.

In OS/400 V4R3, the IBM WebSphere Application Server for OS/400 is included with the IBM HTTP Server for AS/400 product (5769-DG1). The WebSphere Application Server for OS/400 is functionally equivalent to the WebSphere Application Server Version 1.1. The following list shows some of the features that are included:

- Java runtime support for server-side Java servlets
- Java Server Pages
- Java-based Application Server Manager
- Session Tracking framework
- Connectors to common database formats
- Database connection management

In OS/400 V4R4, the WebSphere Application Server is taken out of the IBM HTTP Server product and packaged as its own product, WebSphere Application Server for AS/400 (5769-AS1, a feature of OS/400). This facilitates its use with the HTTP Server for AS/400, as well as with other Web server technologies.

As with V4R3, the WebSphere Application Server for AS/400 continues to be available as a no-charge, optional licensed program. The V4R4 product is shipped with the base operating system. It provides WebSphere Version 1.1 support. It can be upgraded to WebSphere Application Server Standard Edition Version 2.0 through a group PTF. The Standard Edition includes support for Java Servlets, Java Server Pages and an XML parser.

There is a new version of the WebSphere Application Server - WebSphere Application Server Advanced Edition 3.0 for AS/400. It is available as program product 5733-WA2 (56-bit encryption support) or 5733-WA3 (128-bit encryption support). It is currently available in a beta version and planned for general availability in the first quarter of 2000. It is a chargeable licensed program option. The advanced edition includes support for Enterprise JavaBeans.

To find out more about WebSphere, visit the IBM Web site at:
http://www.software.ibm.com/webservers/appserv

1.2 Overview of IBM WebSphere Application Server Advanced Edition 3.0

The WebSphere Application Server Advanced Edition 3.0 is part of an extended brand of IBM products for e-business. It goes beyond the capabilities of the WebSphere Standard Edition product, servlets and Java Server Pages (JSP) support, to provide support for Enterprise JavaBeans (EJBs). EJBs allow you to develop sophisticated server-side components (or objects) for your business. As shown in Figure 1 on page 3, this model may include developing:
• Business applications
• Internet or Intranet based applications that require integration into existing AS/400 applications
• New business applications that require complex database integration with a heterogeneous, multi-database vendor environment.

![Diagram](image)

**Figure 1. Using EJB components**

When utilized with development products such as VisualAge for Java Enterprise Edition 3.0, WebSphere Application Server Advanced Edition 3.0 provides a modern development environment for building Java applications. This environment provides the facilities that provide similar levels of integration, security, and transactional support that was previously only available to the traditional AS/400 developers utilizing ILE languages. The component technology utilized, Enterprise JavaBeans, is the defacto standard within the Java community for building components.

WebSphere Application Server Advanced Edition 3.0 implements the EJB version 1.0 specification, with some 1.1 specification enhancements, particularly in the area of the finder helper methods. For further information on EJBs, refer to the Sun Microsystems Enterprise JavaBeans Technology Web page at URL:


### 1.3 AS/400 Specific considerations

There are several differences between the AS/400 version of WebSphere Application Server Advanced Edition 3.0 and the product shipped for the Windows NT and AIX platforms. The most important differences are discussed in this section.
1.3.1 International and North American Editions

There are two versions of WebSphere Application Server Advanced Edition 3.0 available for the AS/400 system. WebSphere Application Server Advanced Edition 3.0 can utilize Secure Sockets Layer (SSL) support for communications between itself and the security environment. Due to the United States export restrictions on encryption technologies, the AS/400 product ships with versions:

- 56-bit encryption support (5733-WA2) for international usage
- 128-bit encryption support (5733-WA3) for North American usage and restricted usage outside North America.

1.3.2 Install Options

Unlike WebSphere Application Server Advanced Edition 3.0 for NT and AIX, which have several install options depending on what facilities and features you require, the AS/400 product has only two options. They are base and server. Refer to section 1.4, “Installation of WebSphere Application Server Advanced Edition 3.0” on page 6 for further information.

1.3.3 HTTP Server Support

As shown in Figure 2, the WebSphere Application Server Advanced Edition 3.0 server is designed to integrate into existing HTTP servers by utilizing the HTTP server extension capabilities, such as ISAP for the Microsoft Server, NSAPI for the Netscape Server and ICAPI support for the IBM AS/400 HTTP.
1.3.4 Database Support

The AS/400 version of WebSphere Application Server Advanced Edition 3.0 can only utilize DB2/400 for the repository database, when it is installed as the primary server.

1.3.5 Running the Administration Client

The WebSphere Application Server Advanced Edition 3.0 administrative client provides a Java graphical user interface (GUI). Since the AS/400 system does not support native GUI devices, it cannot run directly on the AS/400 system. You must use a Windows or AIX workstation to run the administrative client locally. It is recommended that you do not use AS/400 remote AWT support to run the administrative client.

1.3.6 Multi-instance Support

The AS/400 version of WebSphere Application Server Advanced Edition 3.0 supports multi-instances of the WebSphere Application Server.
The motivation to create multiple HTTP server instances enabled to run WebSphere, is to have the ability to have concurrent but completely independent versions of the instances. Each server instance can read from its own set of properties files, create its own set of log and trace files, work within its own security model and is able to invoke its own Administration Manager interface without affecting any other HTTP server instances that might be running on the AS/400 system at the same time.

Being able to have multiple instances allows you to keep application developers independent of each other. For example, you could use one instance for development, one for testing, and one for serving the web site. Each of these instances will be independent. Changes made to one environment will not affect other environments. You will also find multiple instances of WebSphere useful if you are training several people on WebSphere or running a workshop. Each person can have their own instance running on the same AS/400 system. They will be able to make configuration changes, start and stop the WebSphere Application server without impacting others.

1.4 Installation of WebSphere Application Server Advanced Edition 3.0

The WebSphere Application Server Advanced Edition 3.0 software can be installed from an AS/400 5250 display session or from a personal computer which is LAN connected to the AS/400 system. The actual installation method, does not change regardless of how you install. First, SAVF images from the install CD-ROM are copied to the AS/400 system and then a remote command is issued to restore the license program option that you have selected.

When installing the WebSphere Application Server Advanced Edition 3.0 software, the AS/400 user profile you use must have *ALLOBJ authority. Preferably this profile should be of *SECOFR class or have the privileges associated with a *SECOFR class profile.

Note

This section does not cover all the possible user selectable options available at installation time. For a complete list of installation options available, please refer to the on-line documentation.
1.4.1 AS/400 Installation Options

Regardless of which option(s) you install onto your AS/400 system, you see the following AS/400 library and IFS directories created:

<table>
<thead>
<tr>
<th>AS/400 library or IFS directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QEJB</td>
<td>The product library for WebSphere Application Server Advanced Edition 3.0</td>
</tr>
<tr>
<td>/QIBM/ProdData/WebASAdv</td>
<td>The WebSphere Application Server Advanced Edition 3.0 product IFS objects</td>
</tr>
<tr>
<td>/QIBM/UserData/WebASAdv</td>
<td>The WebSphere Application Server Advanced Edition 3.0 user IFS objects, such as deployed EJBs, servlets and JSPs</td>
</tr>
</tbody>
</table>

Table 1. WebSphere Application Server Advanced Edition 3.0 software

1.4.1.1 Base Option Installation

Installing the Base option of WebSphere Application Server Advanced Edition 3.0, enables you to develop, compile and deploy Java client code that can utilize EJBs as server code. It is intended to support EJB client applications that run on AS/400 systems other than where the EJBs are deployed. The EJBs need to be deployed on a system that has the WebSphere Application Server Advanced Edition 3.0 EJS server installed. This may be an AS/400 system with the Server option (option 1) of the license program installed or an NT or AIX system that has the Enterprise Java Server (EJS) component installed. In any case, the client Java code that uses the EJBs, does need to be WebSphere Application Server Advanced Edition 3.0 based. Additionally, the Base option of WebSphere Application Server Advanced Edition 3.0 installs the components that allow the deployment of servlets and Java Server Pages (JSPs).

Note:

Once you have purchased a copy of WebSphere Application Server Advanced Edition 3.0, you can install the Base option of the license program on any number of AS/400 systems without additional charge.

1.4.1.2 Server Option (Option 1) Install

This component of the WebSphere Application Server Advanced Edition 3.0 license program product provides the Enterprise Application Support. When this option is installed, the following components are installed on your system:

- The Web Server plug-ins
- The Enterprise Java Server (including Administrative functionality)

The EJS server support provides the capabilities to utilize EJBs in your environment and the necessary support to deploy servlets and JSPs.

To install this option, you must either install both the Base and Server options concurrently or previously have installed the Base option.
1.4.2 Installing WebSphere Advanced Edition from the AS/400 System

When you install the WebSphere Application Server Advanced Edition 3.0 software from an AS/400 5250 display, you have the ability to run the install program from the QSHELL environment or via the RUNJVA command. The options available and the underlying install process are identical for both methods. You can use the Qshell Interpreter or the Run Java (RUNJVA) command to install WebSphere from the AS/400 system. In this section, we use the Qshell interpreter.

1.4.2.1 Pre-requisites for Installation

Please refer to the on-line documentation for the current pre-requisites for installation.

1.4.2.2 Installing the Base option

From a 5250 command line start the QSHELL environment. Once inside the QSHELL environment, change directory into the WebSphere installation directory and execute the Setup program. In Figure 7 on page 12 we show the install process, specifying the language option (2924) we wish to install.

```
QSH Command Entry

$ cd /Qopt/WebSphere

F3=Exit   F6=Print F9=Retrieve F12=Disconnect
F13=Clear F17=Top  F18=Bottom  F21=CL command entry

=== SETUP -language 2924 -option base
```

Figure 3. Installing the base option

You see the installation completion message when the base option has been successfully installed as shown in Figure 4.
The installation process creates an AS/400 library and two directories in the Integrated File System (IFS).

### 1.4.2.3 Install the Server Option

To install the Server option, start the QSHELL environment. Once inside the QSHELL environment, change the current directory to the WebSphere installation directory and execute the Setup program. In Figure 5, we show the install process for all parts of WebSphere Application Server Advanced Edition 3.0.
Should the base option already be installed, then change the SETUP command to include the option parameter to install the server code only:

```bash
SETUP -option server
```

You see the installation completion message when the option(s) have been installed as shown in Figure 6 on page 11.
Figure 6. Server installation completion

The installation process either creates the QEJB AS/400 library and the WebASAdv directory in the /QIBM/ProdData and /QIBM/UserData directory structures or updates the existing directories with the additional OS/400 objects, java classes and documentation for the Server option.

1.4.3 Installing WebSphere from a PC

You can also install WebSphere Application Server Advanced Edition 3.0 from a Windows 32-bit operating system LAN attached to the AS/400 system. The installation options available are identical to those available when installing from a 5250 screen. Within this section, we only show selected screens from a complete WebSphere Application Server Advanced Edition 3.0 installation.

QSH Command Entry
$ > cd /Qopt/WebSphere
$ > SETUP
Attaching Java Program to OS400/Install.Jar.
COPYING STREAM FILE TO SAVE FILE. (BASE CODE). 
RESTORING LICENSED PROGRAM. (BASE CODE).
COPYING STREAM FILE TO SAVE FILE. (BASE LANGUAGE). 
RESTORING LICENSED PROGRAM. (BASE LANGUAGE).
COPYING STREAM FILE TO SAVE FILE. (SERVER CODE). 
RESTORING LICENSED PROGRAM. (SERVER CODE).
COPYING STREAM FILE TO SAVE FILE. (SERVER LANGUAGE). 
RESTORING LICENSED PROGRAM. (SERVER LANGUAGE).
INSTALLATION COMPLETED SUCCESSFULLY

====>

F3=Exit  F6=Print  F9=Retrieve  F12=Disconnect
F13=Clear  F17=Top  F18=Bottom  F21=CL command entry

Before starting a remote installation from a PC, ensure as a minimum you have the following AS/400 host servers started:
- File server
- Remote Command server
- Signon server

If these host servers are not started, the installation process will not complete. Use the STRHOSTSVR command to start the host servers.

When you install the WebSphere Application Server Advanced Edition 3.0 software from a LAN connected PC, expect the install process to be a minimum of seven times longer than a native 5250 installation process. This difference will vary depending on the type and utilization of the LAN and the AS/400 workload.
1.4.3.1 Pre-requisites for Installation
Our experience from installing via a LAN connected PC is that you should use a PC similarly configured to that required for running the WebSphere Application Server Advanced Edition 3.0 Administrative console. During the installation process you will need a minimum of 10Mb of free disk space.

1.4.3.2 Installing the Software
Insert the WebSphere Application Server Advanced Edition 3.0 CD-ROM into the PC CD drive. This should automatically start the install process. If your PC environment does support this or it has been disabled, use Windows Explorer to double-click on the setup.bat file in root directory of the CD-Rom. As shown in Figure 7, you are presented with the initial screen of the WebSphere InstallShield program.

![Initial remote installation](image)

By clicking on the next button, you are presented with the AS/400 sign-on screen as shown in Figure 8 on page 13.
Enter the host name or the IP address of the AS/400 system on which you want to install WebSphere and an AS/400 user id and password, that has at least *ALLOBJ authority. Click on the next button to continue. If you enter any invalid information on this screen, you are prompted to re-enter the information.

Figure 9. WebSphere components
The install process checks to see if any components of the WebSphere Application Server Advanced Edition 3.0 are installed on the AS/400 system, before displaying the install option screen. As you can see, in Figure 9 on page 13, we did not have the product installed. Select the components and language options you wish to install and click on the Next button.

As shown in Figure 10, you are prompted for an installation directory.

![Choose Destination Directory dialog](image)

**Figure 10. Installation directory**

This is a slightly confusing dialog, as this directory is used to hold the PC based un-install program and not the actual WebSphere software. It is recommended that you specify a directory within a central AS400 system IFS. This enables you to run the un-install process from any PC. If you specify a local PC directory you can only run the un-install program from that PC.

Click on the next button to continue with the installation process. Should this directory not exist you will be prompted and asked if you wish to create it. Select **Yes**, if you are sure you wish to use this directory, otherwise select **No** and you will have the option of change the directory.

You are now presented with several windows that indicate what stage of the install process you are currently in. They look similar to Figure 11 on page 15.
As shown in Figure 12, when the installation is complete, you are presented with the completion screen. Clicking on the finish button, automatically removes any transient work files.
1.4.4 Installing the Administration Client on a PC

The WebSphere Administrative Console is used to manage the application servers. You must install the Administrative Console before using WebSphere Application Server.

To install the Administrative Console for WebSphere Application Server on your workstation, follow these steps:

1. Insert the WebSphere Application Server 3 Advanced Edition for Windows NT, WebSphere Application Server 3 Advanced Edition for AIX or WebSphere Application Server 3 Advanced Edition for Solaris CD-ROM in the CD-ROM drive of the Windows NT, AIX, or Solaris workstation that you plan to run the console on.

   **Note**

   Do not use the WebSphere Application Server Advanced Edition for AS/400 CD-ROM (which also came in your WebSphere Application Server Advanced Edition for AS/400 package) for this set of steps.

2. Step through the WebSphere Application Server InstallShield program. You should only install the WebSphere Administrative Console portion of WebSphere Application Server on your workstation. Keep these points in mind:
   - Ignore any messages that tell you to shut down your Web server.
   - On the Install Options page, select the Administrator's Console option. The dialog is illustrated by the following figure:

![Installing the Administration Console](image)

   **Figure 13. Installing the Administration Console**

   - When the program prompts you for a host name, enter the host name or IP address of your AS/400 system.
   - You must enter the host name or IP address exactly as it appears on your AS/400 system.
1.5 Un-Installing WebSphere Application Server Advanced Edition 3.0

WebSphere Application Server Advanced Edition 3.0 can be un-installed by using the DLTLICPGM command from a 5250 command line or from the PC based un-install bat file created if you installed from a PC.

After the license program has been deleted, the /QIBM/UserData/WebAsAdv directory is left on your system. Should you wish to re-install the product at a later stage, it is recommended that you remove this directory.

---

**Note**

If you chose to un-install from a PC, all the optional components of the WebSphere licensed program product are deleted. Should you wish to only remove the server option of WebSphere Application Server Advanced Edition 3.0, you must use the AS/400 DLTLICPGM command.

---

1.6 Configuring the WebSphere software

If you only install the Base option of the WebSphere Application Server Advanced Edition 3.0 license program, then there is nothing for you to configure. If you want to use the EJS support to deploy and serve Enterprise JavaBeans, you must use the Administrative Console to configure the application server support.

1.6.1 Administrative Terminology

WebSphere Application Server Advanced Edition 3.0 provides administrators with a single system view of applications and resources, such as servlets, that are typically deployed across multiple machines in a distributed environment. An administrator working on one physical machine can remotely administer resources located on another physical machine. In the WebSphere administrative model:

- A physical machine is called a node. Each node contains an administrative server for administering the resources on it. Each administrative server stores its administrative data in an administrative repository, in the case of the AS/400 system a DB2/400 database. The person installing IBM WebSphere Application Server specifies which administrative repository a given administrative server will use.
- An administrative domain is a set of one or more nodes whose administrative servers share an administrative repository, allowing the nodes to be aware of one another and distribute applications among themselves.
- Each administrative domain has its own topology, comprised of the nodes in the administrative domain, and the resources those nodes contain.
- The resources in a node, such as servlet class files and enterprise bean JAR files, are represented as administrative resources in the administrative domain. An administrative resource, such as a servlet, holds configuration information about a "real" resource, such as a servlet file installed on a node. It provides a way to start, stop, and otherwise manage the real resource, perhaps remotely.
- The topology reveals the containment hierarchy, which is simply a tree illustrating how the administrative resources in the topology are related. The hierarchy shows which resources are parents or children of other resources.
The Types tab of the WebSphere Administrative Console demonstrates the containment hierarchy.

- The WebSphere Administrative Console enables administrators to access the administrative server on each node in the administrative domain and provides a view of the domain's topology. It supplies task wizards for managing and combining resources in the topology.

1.6.2 Relationships among Administered Resources

It is important to understand the containment hierarchy that determines how the resources are interrelated. The containment hierarchy imposes a structure on the topology of the administrative domain. Becoming familiar with the hierarchy will make your administrative tasks seem easier to accomplish.

**Nodes**

Use nodes to specify machines to which you can distribute servers, servlets, enterprise beans, applications, and other resources for workload management. Nodes are physical machines in an administrative domain. Each node must contain an administrative server. In this way, a node also represents the administrative server process on the node. For example, the administrative console can show that the node resource is in the "running" state. That indicates not only that the physical machine is running, but that the administrative server on the machine is also running. The administrative server must be running in order for the node to be active and operational in the administrative domain. In the containment hierarchy, nodes contain application servers, generic servers, servlet redirectors, OLT controllers, and the children of these resources. Nodes are contained by the administrative domains to which they belong.

**Application Servers**

Use application servers to extend your Web server's capabilities to handle requests for enterprise beans, servlets, Web applications, and the applications that contain them. An application server consists of a Java virtual machine (JVM) configuration and an enterprise bean server process for hosting enterprise beans and applications comprised of enterprise beans and other resources. The "application server" should not be confused with the WebSphere Application Server Advanced Edition 3.0 product. The WebSphere Application Server Advanced Edition 3.0 product can include one or more application server processes for each machine on which it is installed. In the containment hierarchy, application servers contain EJB containers and enterprise beans. They can also contain a servlet engine (one per application server) and their children. Application servers are contained by nodes. Multiple application servers can exist on the same node.

**EJB Containers**

Use EJB containers to configure container support for enterprise beans. An EJB Container is a runtime resource used to contain and host enterprise beans. Properties on the container are used to fine tune the runtime to the requirements of the enterprise beans. In the containment hierarchy, EJB containers contain enterprise bean resources. EJB containers are contained by application servers.

**Enterprise Beans**

Use enterprise beans to configure enterprise bean JAR files you want to manage individually or as part of applications. Each icon inside the container represents a deployed enterprise bean. An enterprise bean is a Java component that can be
combined with other enterprise beans and Java components to create a distributed, client/server application. There are two types of enterprise beans. An entity bean encapsulates permanent data, which is stored in a data source (database), and provides associated methods to manipulate the data. A session bean encapsulates ephemeral (nonpermanent) data and business logic associated with a client session. In the containment hierarchy, enterprise beans do not contain other resources. Enterprise beans are contained by EJB containers.

Figure 14 shows the topology pane of the WebSphere Administrative Console and demonstrates the relationship of the resources.

**Figure 14. Administrative console**

### 1.6.3 Configuring Multiple Administrative Servers

A single administrative server allows you to run many application servers. Each application server runs in its own process, so in most cases a single administrative server will handle your scalability and isolation needs. Additionally, a single administrative server allows you to use a single Administrative Console to manage all the server resources. However, since all your application servers in an administrative domain share the same Persistent Name Server name space, you may want to create multiple administrative servers on a single AS/400 system for any of the following reasons:
You may want to create separate development environments for different developers. This allows them to have different versions of the same objects in their own name space.

You may want to create separate development and test environments.

If you want to create multiple administrative servers on one AS/400 system, see the Administration section of the on-line documentation.

1.6.4 Configuring an Administrative Server without a Sample Environment

Unlike the installation process for the NT and AIX versions of WebSphere Application Server Advanced Edition 3.0, the AS/400 installation process configures the Administration properties to automatically install the sample environment. The WebSphere samples gallery provides a set of small, generic samples that show how to perform common Web application tasks. It is installed in the repository when the server is first started. Under certain circumstances, such as when you are importing another Administrative Server configuration, this may not be a desirable.

To create an Administrator Server instance, without the sample configuration, you must edit the admin.properties file in the properties directory of the Application Server and change the install.initial.config entry to be false.

1.6.5 Exporting an Administrative Server Configuration

It is possible to export an existing Administrative Server configuration as an XML document. It can then be imported to another Administrative Server on the same or another platform.

1.6.6 Importing an Administrative Server Configuration

You can use this facility to import the configuration from one Administrative Server to another to simplify updating your Administrative Server Configuration with new configuration information via a set of XML information.

One the primary uses for this facility is for the transferring of an NT or AIX based Administrative Server to the AS/400 system. After importing the XML configuration, you have the responsibility of moving all your application code into the correct directories on the AS/400 system.

1.7 Starting and Stopping the WebSphere Application Server

WebSphere Application Server Advanced Edition 3.0 runs in its own AS/400 subsystem, called QEJBSBS. The QEJBSBS subsystem is installed with the product. A monitor job, an administrative server job, and one or more application server jobs run in the QEJBSBS subsystem. The administrative server configures, monitors, and manages your application server jobs. The administrative server also sends completion and error messages to the job log.

HTTP servers run in a separate subsystem (QHTTPSVR), and each HTTP server instance starts multiple jobs. WebSphere Application Server plug-in code runs in the HTTP server job that communicates with the administrative server and one or more application servers. To start and stop HTTP server instances, use the STRTCPSVR or ENDTCPSVR command on AS/400, or use Operations Navigator.
1.7.1 Start-up

You must start the TCP/IP support before the QEJBSBS subsystem can start. The CL command to start up the QEJBSBS subsystem is:

STRSBS QEJB/QEJBSBS

As shown in Figure 15, when the QEJBSBS subsystem is active, the WebSphere Application Server environment is ready for use. Two jobs make up the administrative server environment. The jobs are a monitor job, QEJBMNTR, and an administrative server job, called QEJBADMIN.

![Figure 15. WebSphere Application Server environment](image)

The monitor job starts the administrative server and attempts to restart it when it has ended abnormally. By default, the QEJBSBS subsystem starts only these two jobs.

If the administrative server job stops, you can start it without stopping and restarting the subsystem if your user profile has *JOBCTL authority. You can also start additional administrative servers. To verify that the administrative server is running, use WRKACTJOB or in Operations Navigator, look for QIBM_WSA_ADMIN under server jobs.

You can configure the WebSphere administrative server to automatically start at AS/400 system start-up.

TCP/IP must be active before the WebSphere Application Server environment can start. Ensure that the STRTCP command runs before the STRSBS QEJB/QEJBSBS command in your start-up program or in your autostart job.

Do either of the following:

- Change the system start-up program by adding the following line:
  
  STRSBS QEJB/QEJBSBS

- Add an autostart job entry to the QSYSWRK subsystem, using the following command:
1.7.2 Shutdown

The WebSphere Application Server environment is shutdown when all the administrative servers are stopped and the QEJBSBS subsystem has ended. An administrative server is shutdown when the administrative server monitor job, the administrative server job, and the application server jobs associated with the administrative server are ended.

To stop the administrative server, do one of the following:

- From the WebSphere Administrative Console, stop all the nodes that represent administrative servers running on your AS/400 system. Using the Administrative Console is the recommended way to shut down the WebSphere Administrative Server. The next two methods should be used only if this method fails to shut down the WebSphere Administrative Server.
- From the AS/400 command line, use the ENDJOB command to end the administrative server (QEJBADMIN) and server monitor (QEJBMNTR) jobs. Ending an administrative server job ends the administrative server job. Be sure to also end the server monitor job. If it detects that the administrative server job has been ended, it will restart it. Using the Administrative Console is the recommended way to shut down the WebSphere Administrative Server.
- End the QEJBSBS subsystem. When the subsystem is ended, all running jobs also end. To end the QEJBSBS subsystem, type `endsbs qejbsbs` from an AS/400 command line. This action may or may not update the administrative repository with the correct status of the node, because the order in which the jobs in the subsystem are terminated is unpredictable. Using the Administrative Console is the recommended way to shut down the WebSphere Administrative Server.

Any shutdown method preserves the current run state of the application servers. The administrative server attempts to restore the running state during the next start-up.

1.8 Starting the Administrative Console

The WebSphere Application Server Advanced Edition 3.0 must be started before you can start the Administrative Console. To start the WebSphere Administrative Console and connect it to your AS/400 system, you must specify the AS/400 host name and optionally specify the persistent bootstrap port that the administrative server is using on AS/400. The WebSphere Administrative Console uses port 900 by default. If you changed the default port with the `admin.bootstrapPort` parameter when starting the administrative server, you need to specify that port for the WebSphere Administrative Console. The `admin.bootstrapPort` is specified in the `admin.properties` file.

To start the WebSphere Administrative Console on a Windows NT workstation, you can either use the Start menu shortcut that was created during installation on
Windows NT, or run a batch file from a command prompt. To start on AIX or Solaris, you need to run a shell script from a command prompt.

To determine the host name, follow these steps:

1. Enter the Configure TCP/IP (CFGTCP) command on the AS/400 command line.
2. Select option 12 (Change TCP/IP Domain).
3. Note the Host name value. This value should be used as the Host name parameter. The Host name parameter is case sensitive.

It is required that you have a host name entered on the AS/400 system. The WebSphere Administrative Console will not connect if the entry is not present. If you do not have a host name entry, add it. In addition, if the host name is in lowercase, you must also use the lowercase name when connecting with the WebSphere Administrative Console.

This method works in most simple cases. In more complicated scenarios with systems that have multiple IP addresses, multiple alias names, or multiple domain name server (DNS) entries, there may be some additional TCP/IP configuration required to connect.

### 1.8.1 Using the Start menu shortcut

To use the Start menu shortcut, you must change the shortcut to specify the host and optionally the port.

1. Find the shortcut for the WebSphere Administrative Console. Use Windows NT Explorer to navigate down to the folder where the shortcut profile is stored. By default it should be in a directory structure that looks like this:
   
   C:\WINNT\Profiles\All Users\Start Menu\Programs\IBM WebSphere\Application Server V3.0\Administrator's Console.link.

2. Right-click on the Administration Client shortcut and select the Properties view.

3. Go to the Shortcut notebook page. Change the Target field to add a host and optionally add a port. For example, if myAS400 is your host name and 12345 is your port number, you would change the target field from C:\WebSphere\AppServer\bin\adminclient.bat to C:\WebSphere\AppServer\bin\adminclient.bat myAS400 12345

   The host name should be in upper case.

4. Start the WebSphere Administrative Console on Windows NT. From the Start menu, click Programs-->IBM WebSphere-->Application Server V3.0-->Administrator's Console.

### 1.8.2 Using a command prompt

The batch file, named adminclient.bat, can be used to start the Administration Console from a command prompt. You must specify the host and optionally the port.

1. Open a command prompt, and enter the following two commands:
   
   cd <install-root>\bin
   adminclient <host> <port>

   where <install-root> is the directory where the WebSphere Administrative Console is installed, <host> is the host name of your AS/400 system, and <port> is the AS/400 port number that you intend to use. The port number should match
what was specified for the admin.bootstrapPort parameter in the admin.properties file.

The amount of time that it takes for the WebSphere Administrative Console to start depends on the number of objects that you have configured. It is not uncommon for this to take several minutes when configured with a large number of objects. While the WebSphere Administrative Console is starting up, you see an animated icon of two circling arrows. When the animation stops, the console is done starting up and is ready to use. The "Console Ready" message is also displayed in the message area in the bottom of the WebSphere Administrative Console window.

### 1.9 DataSources

WebSphere Application Server Advanced Edition 3.0 implements the Sun Java servlet API 2.1. Beyond this, it introduces a new method for database connection pooling, termed DataSources. A DataSource is an abstraction for media protocol-handlers. A DataSource manages the life-cycle of the media source by providing a simple connection protocol. It also provides enhanced extended functionality such as session and user profile support.

A DataSource object represents the logical name of a JDBC-enabled database used by entity beans to store persistent data. This shields the enterprise bean developer from the underlying physical location of the database. A DataSource is associated with a JDBC driver object. A driver can have many DataSources associated with it. A DataSource also defines the pool of connections to a database.

To create a data source, a JDBC driver must exist. JDBC drivers must first be created, by creating a driver model, which is the logical representation of the database driver. The model is then replicated (installed) on each node in the administration domain where a driver installation exists.

#### 1.9.1 Creating and Installing a JDBC Driver

To create a JDBC driver, go to the Types tab of the WebSphere Application Server Advanced Edition 3.0 Administrative Console, select JDBCDrivers, click the mouse right button and choose Create...
As shown in Figure 17 on page 25, the Create a JDBCDriver panel is displayed.

As shown in Figure 17:

- Enter a name for the driver.
- Enter the name of the JDBC driver implementation class. In this case we use the native AS/400 JDBC driver.
- Enter the URL prefix supported by the JDBC driver.
- Decide whether your driver will be JTA enabled or not.
- Click the Create button.
Next, we need to install the driver. As shown in Figure 18, go to the Topology tab of the WebSphere Application Server Advanced Edition 3.0 Administrative Console, select the just created JDBC driver, click the right mouse button and select **Install**...

Important

The AS/400 Toolbox for Java JDBC driver cannot be JTA enabled.

The AS/400 Native JDBC Driver can be JTA enabled, in this case the database URL prefix must be:

```
jdbc:jta:db2
```

Figure 18. Installing the driver

As shown in Figure 19 on page 27, from the Install Driver dialog, select the node in which you want to install the driver from the listbox and click the Browse button.
As shown in Figure 20 on page 27, select the db2_classes.jar jar file from the /QIBM/ProdData/Java400/ext directory and click the Open button.

As shown in Figure 21, the Install driver dialog appears again. Click the Install button to install the JDBC driver.
1.9.2 Creating a DataSource

In this section, we create a DataSource. As shown in Figure 22 on page 28, go to the Types tab of the WebSphere Application Server Advanced Edition 3.0 Administrative Console, select DataSources, click the right mouse button and choose Create...

![WebSphere Advanced Administrative Console](image1)

Figure 22. WebSphere Advanced Administrative Console

As shown in Figure 23, you now see the Create a DataSource dialog.

![Create a DataSource](image2)

Figure 23. Create a DataSource

As shown in Figure 23:

- Enter the name for the DataSource.
- Enter the database name. In the case of the AS/400 native JDBC driver, you can use the value "LOCAL."
You can also use the name of the \*LOCAL entry of the WRKRDBDIRE AS/400 command. It is optionally followed by a slash and a default library name.
- Enter the name of the just created and installed JDBC driver.
- Select the Advanced tab.

As shown in Figure 24, you now see the Advanced setting panel.

![Create a DataSource](image)

**Figure 24. DataSource Advanced settings pane**

In the panel shown in Figure 24, you may change the default values for the DataSource connection pool. Click the Create button to create the DataSource.

### 1.9.3 Adding DataSource information to an EJB Container

To add DataSource information to an EJB Container, go to the Topology tab of the WebSphere Application Server Advanced Edition 3.0 Administrative Console. As shown in Figure 25 on page 30, select the EJB Container to which you want to add the DataSource. In the right part of the panel select the DataSource tab and click the Change button.
Figure 25. WebSphere Advanced Administrative Console

As shown in Figure 26, you now see the Select a DataSource dialog.

Select the just created DataSource from the list shown in Figure 26 and click the OK button. As shown in Figure 27 on page 31, the DataSource name now appears in the corresponding field of the DataSource tab of the EJB container properties.
Figure 27. DataSource tab

As shown in Figure 27, enter a valid database user id and password in the corresponding fields, then click the Apply button. You can also use the value QEJBSV FR for the user id. In this case, the password will be automatically entered. When you see the console message indicating that the action completed successfully, the DataSource is available to use.
Chapter 2. AS/400 WebSphere Servlet and JSP Development

In this chapter, we cover AS/400 servlet and JSP application development for the WebSphere Application Server Advanced Edition 3.0 environment.

2.1 Servlet Support in WebSphere Advanced Edition 3.0

Unlike earlier versions, the WebSphere Application Server Advanced Edition 3.0 environment exists independent of the HTTP server utilizing its resources. On a single AS/400 system, this enables any number of HTTP servers to utilize the same resources of a single WebSphere Application Server as shown in Figure 28.

![Figure 28. WebSphere V3: Typical Servlet scenarios](image)

If we review Figure 28, and look at how WebSphere Application Server Advanced Edition 3.0 changes the sequence of communication between a browser and a servlet, we appear to have an additional step in the process. That is, the Web Server interface to the Application Server. In WebSphere version 2.0 this step still occurs and would be more obvious if you configured your environment to run "out of process". In this case the Java Application Server runs in its own job. Indeed, this may be a good analogy to use when thinking about the WebSphere Application Server Advanced Edition 3.0 environment.

WebSphere Application Server Advanced Edition 3.0 implements version 2.1 of the Sun Java servlet class libraries. WebSphere Standard Edition 2.0 implements version 2.0. There are numerous changes between the version 2.0 and version 2.1 specification. We discuss some of these changes later in this chapter. The JSWDK (JavaServer Web Development Kit) 1.0 is the reference implementation for JavaServer Pages 1.0 technology and the Java servlet API 2.1. The JSWDK is available at no charge from the Sun Microsystems web site. For a complete definition of the Sun Java servlet API specification, refer to the Sun on-line documentation at:
2.2 IBM development environments for WebSphere applications

There are two IBM application development environments that can be utilized for developing WebSphere applications.

- IBM WebSphere Studio version 3.0, provides a combined development environment for servlets and JSPs
- VisualAge for Java Enterprise Edition 3.0 provides a servlet and Enterprise JavaBean development environment.

For writing this redbook, we utilized VisualAge for Java Enterprise Edition 3.0 as our development environment. We required the ability to easily integrate servlets with both the AS/400 Toolbox for Java classes and Enterprise JavaBeans. While IBM WebSphere Studio is a simpler tool to use to achieve servlet/JSP generation, it would have required us to hand modify the generated servlets and import them into VisualAge for Java for integration into our ultimate integrated JSP/Servlet/EJB environment.

2.3 To Port or not to Port, that is the Question

Under certain circumstances, you can take IBM WebSphere Standard Edition version 2.0 servlets and deploy them in a WebSphere Application Server Advanced Edition 3.0 environment, without making any Java coding changes. In this case, you do not take advantage of the new features offered by WebSphere Advanced Edition. This is not the recommendation that we make.

As previously stated, WebSphere Application Server Advanced Edition 3.0 implements the Sun Java servlet API 2.1. Beyond this, it introduces a new method for database connection pooling, termed DataSources. A DataSource is an abstraction for media protocol-handlers. A DataSource manages the life-cycle of the media source by providing a simple connection protocol. A DataSource also provides enhanced extended functionality such as session and user profile support.

In WebSphere Application Server Advanced Edition 3.0, the servlet 2.0 APIs, the IBM Connection Manager APIs, WebSphere V2.0 session and user profile support are supported as deprecated APIs. In some cases, particularly in session support, the V2.0 APIs return NULL values from the method calls.

It is understood that IBM will remove support for the IBM Connection Manager APIs in WebSphere version 4.0. At the time of the writing of this redbook we are unaware of any IBM plans to discontinue support for the servlet 2.0 API specification and the WebSphere version 2.0 session, user profile and personalization API support within WebSphere Advanced edition.
2.3.1 Keeping Servlets at the Sun 2.0 Specification Level

If you choose not to migrate your servlet API 2.0 servlets to the new 2.1 specification, you should be aware of several differences between WebSphere Application Server version 2.0 and version 3.0. Three of the most important differences that may force you to convert are in the areas of the IBM Connection Manager support, IBM session Management support and servlet chaining.

2.3.1.1 IBM Connection Manager Issues

The IBM Connection Manager classes have been deprecated within WebSphere Application Server Advanced Edition 3.0 and there is no Administrative Client GUI functionality provided for configuration. When the IBM Connection Manager classes are used, they are transparently mapped into a dynamically created DataSource. These DataSources are internal to WebSphere Application Server Advanced Edition 3.0 and are not displayed within the Administration Client topology pane. As there is no means provided to configure any settings, each DataSource created will be generated with the equivalent of the following WebSphere Standard Edition IBM Connection manager pool settings:

<table>
<thead>
<tr>
<th>Connection Pool setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>reapTime</td>
<td>-1 (Disabled)</td>
</tr>
<tr>
<td>connectionTimeOut</td>
<td>300000</td>
</tr>
<tr>
<td>maxIdleTime</td>
<td>-1</td>
</tr>
<tr>
<td>minConnections</td>
<td>1</td>
</tr>
<tr>
<td>maxConnections</td>
<td>500</td>
</tr>
<tr>
<td>maxAge</td>
<td>-1 (Disabled, No garbage collection)</td>
</tr>
</tbody>
</table>

Table 2. Dynamic DataSource settings

The dynamically generated DataSources are not created with the pool name specified in the IBMJdbcConnSpec object. Instead, the pools created by the IBM Connection Manager method calls are organized by appending the database URL and the user identification specified onto "was20". For example, if we execute the following code:

```java
Connection Pool setting Value
reapTime -1 (Disabled)
connectionTimeOut 300000
maxIdleTime -1
minConnections 1
maxConnections 500
maxAge -1 (Disabled, No garbage collection)
```
We can effectively consider the name of the DataSource or pool to be:

`was20jdbc:db2://as20a999512b`. This has implications if you use multiple connection pools in your environment which use the JDBC URL and the same user identifier. Once the DataSource is created you cannot change any attributes. For example, you may have two connection pools in a WebSphere version 2.0 application. Both connection pools use the same URL and user id. When a servlet uses either of these pools it uses the same WebSphere V3 DataSource. If you wish to keep distinct connections you will need to migrate to the new WebSphere version 3.0 DataSources.

### 2.3.1.2 Servlet Chaining and Filtering

WebSphere Application Server Advanced Edition 3.0 has changed the way in which servlet chaining works. The servlet chaining method of WebSphere Standard Edition 2.0, depicted in Figure 30 on page 37, provides each servlet in the chain with the original HTTP request as input and the independent output of each servlet is ultimately merged to produce the final HTML document.
The servlet chaining methodology of WebSphere Application Server Advanced Edition 3.0 passes the request object and the servlet out stream of the first servlet to the next servlet in the chain until the last servlet returns the response to the client. Only the last servlet can set the HTTP response header.

As the request object is passed from one servlet to another, additional information can be stored in the request object for the next servlet to utilize. Unlike servlet filtering, the chain or list of servlets can be dynamically modified.

Note
The terms use to describe the process of Servlet chaining and Servlet filtering have been changed in the WebSphere Application Server Advanced Edition 3.0 documentation. The new terms are Servlet list filtering and Mime-based filtering respectively.

### 2.3.1.3 Session and Session Context Issues
If you used the IBM extension to the standard session support in WebSphere Standard Edition version 2.0 by utilizing the public classes in `com.ibm.servlet.personalization.sessiontracking` package, you must migrate your servlets to the new IBM extension for version 3.0 provided in the `com.ibm.websphere.servlet.session.IBMSession` package. The version 2.0 classes have been deprecated. While you can still compile your servlets using the old classes. Specifically, the IBMSessionData class type cast will still work. The functions will return null or constant values, and no processing or setting of values will occur.
All clustering with WebSphere Application Server is now handled through a database, and the version 2.0 concept of session cluster client and server is no longer valid. If you store non-serializable objects in your session objects, then these session objects cannot be stored or retrieved from the session database. This has implications not only for clustering but also persistence. To make your applications portable to a clustered environment, you must make any objects placed in a session object serializable.

Unless specifically configured, a single session maps to a single row in the session database table. The maximum amount of data that can be stored in a single row is approximately 32K. If this is not large enough, you can configure session management to use multiple rows to hold your session information. Please refer to the WebSphere Application Server Advanced Edition 3.0 on-line documentation for further information.

2.4 Migrating Servlets

We believe that it is essential to understand and feel comfortable with the migration from the WebSphere version 2.0 Connection Manager support to WebSphere Application Server Advanced Edition 3.0 support. In this section, we look at this in greater depth. Later we look at migrating the servlets from the redbook SG24-5635 *Building AS/400 Applications for IBM WebSphere Application Server 2.0 Standard Edition* to the WebSphere Application Server Advanced Edition 3.0 environment.

2.4.1 Migrating from IBM Connection Manager to DataSources

The `DatabaseConnectivity` class detailed below, provides a simple object wrapper for interfacing with the IBM Connection Manager support available with WebSphere version 2.0. This code snippet is not complete and is used to highlight conversion issues. Figure 31 on page 39 shows the class description.
Figure 31, shows the import statements required for a java class that utilizes the IBM Connection Manager support. The most important package to import is "com.ibm.servlet.connmgr.*" which provides the IBM Connection Manager functionality. For simplification, we set the connection properties using variables. In a real application, you should retrieve these values using initialization parameters or from a resource bundle.

Figure 32 on page 40 shows the default constructor.
public DataBaseConnectivity()
{
    try
    {
        url = "jdbc:" + Db + ":" + DbName;
        try
        {
            // Create JDBC connection specification.
            connSpec = new IBMJdbcConnSpec(poolName, // pool name
                true, // waitRetry
                jdbcDriver, // Remaining four
                url, // parameters are
                userId, // specific for a
                userPassword); // JDBC connection.
            // Get a reference to the connection manager.
            connMgr = IBMConnMgrUtil.getIBMConnMgr();
            // Get a Connection object (dataConn). This is an object from the java.sql package and it is used for JDBC access.
            cmConn = (IBMjdbcConn) connMgr.getIBMConnection(connSpec);
            dataConn = cmConn.getJdbcConnection();
        }
        catch (Exception e)
        {
            System.out.println("DataBaseConnectivity Exception occurred: (Build connection) " + e.getMessage());
            e.printStackTrace(System.out);
        }
    }
    catch (Exception e)
    {
        System.out.println("DataBaseConnectivity Exception occurred: (Other) " + e.getMessage());
        e.printStackTrace(System.out);
    }
}

Figure 32. Retrieving a JDBC connection from the Connection Manager

Within the default constructor we get a JDBC connection from the Connection Manager pool:

- We create a specification object, containing the information required to connect to the underlying data server.
- We retrieve a reference to the connection manager.
- We retrieve a connection manager connection.
- We retrieve the JDBC connection from the connection pool.

Figure 33 on page 41 shows the getConnection method which returns a Connection object.
When a JDBC connection is requested, we ensure that we still have exclusive use of the connection and it has not been reclaimed by the IBM Connection Manager. This can occur when we have a long running servlet and the IBM Connection Manager pool time out values are set too low. We use the `isCMConnectionValid()` method, shown in Figure 34 on page 42, to validate a connection.
Figure 34. Validating ownership of the JDBC connection

The isCMConnectionValid() method checks to ensure that the Connection Manager connection and JDBC connection are still valid and allocated to us. If not, we attempt to rebuild the connection.

Figure 35 shows some auxiliary methods available with this class.

```java
private int isCMConnectionValid() throws Exception, SQLException {
    try {
        if (!cmConn.verifyIBMConnection()) {
            // Get valid cmConn and dataConn objects if cmConn
            // has "timed-out".
            cmConn = (IBMjdbcConn) connMgr.getIBMConnection(connSpec);
            dataConn = cmConn.getJdbcConnection();
            return ConnectionRebuilt;
        } else {
            return ConnectionValid;
        }
    } catch (IBMConnMgrException e) {
        System.out.println("isCMConnectionValid: IBMConnMgr exception: " + e.getMessage());
        e.printStackTrace();
        throw new Exception("isCMConnectionValid: IBMConnMgr exception: " + e.getMessage());
    }
}
```

Figure 35. Connection Manager Example: Auxiliary methods

WebSphere Application Server Advanced Edition 3.0 uses the JDBC 2.0 core and optional packages (formerly JDBC 2.0 standard extension) style and APIs for database connection pooling. This positions it for Java 2 support. You should be aware that the underlying JDBC connection object returned is a JDBC 1.0 connection object, because WebSphere Application Server Advanced Edition 3.0 will only run in a Java 1.1 Runtime Environment (JRE).

Both AS/400 JDBC drivers, the native and AS/400 Toolbox for Java drivers are already JDBC 2.0 compliant. If you require this level of functionality within your connections, you should cast the returned JDBC connection object to the
corresponding driver object, which are DB2Connection and AS400JDBCCConnection for the native and toolbox drivers respectively. However, this greatly reduces the portability of your code.

### 2.4.1.1 DataSource Version

Next, we show a DataSource version of the IBM Connection Manager wrapper class. As shown in Figure 36, we name the class DataBaseConnectivity.

```java
package WebSphereV3;
import java.sql.*;
import java.util.*;
import com.ibm.db2.jdbc.app.stdext.javax.sql.*;
import com.ibm.ejs.dbm.jdbcext.*;
import javax.naming.*;

public class DataBaseConnectivity
{
  // Hard code datasource connectivity information. The information
  // could be brought in at runtime (from an external property file
  // identified) or passed into a method .

  // Use to communicate with connection manager.

  static String userId = "Jeff";
  static String userPassword = "Jeff";
  static String v3DataSourceName = "jdbc/MyDataSource";
  private Context v3Ctx = null;
  private DataSource v3DS = null;
  private Connection dataConn = null;
}
```

**Figure 36. DataSource Example: imports and variable definitions**

Figure 36 shows the new imports statements for the DataSource support. These are explained in Table 3.

<table>
<thead>
<tr>
<th>Import statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.ibm.db2.jdbc.app.stdext.javax.sql</td>
<td>This package provides the JDBC DB2 optional package support for JNDI and distributed transaction support.</td>
</tr>
<tr>
<td>com.ibm.ejs.dbm.jdbcext</td>
<td>This package provides the WebSphere connection pooling support.</td>
</tr>
</tbody>
</table>

**Table 3. DataSource packages**

Notice, the new fields required to support the new JDBC 2.0 style lookup for the DataSource support:

- Context: Used to access the Java servers JNDI naming service.
- DataSource: Used to define the data source.

Once again for simplification, we set the properties using variables.

Figure 37 on page 44 shows the default constructor method.
As shown in Figure 37:

- We use JNDI to get access to the DataSource (or any other object) from the naming space. We create a Hashtable object to hold the necessary parameters to access the naming space.
- We specify IBM EJS CNIInitialContextFactory (com.ibm.ejs.ns.jndi.CNIInitialContextFactory) as the Context factory object we wish to use.
- We get a Context object for the EJS JNDI interface.
- Using the Context object, we do a JNDI lookup for the logical name of the DataSource.
- The Context lookup returns a java object, which we cast to a DataSource object. All DataSource names exist in the 'jdbc' subcontext of the root.

As shown in Figure 38 on page 45, we use the getConnection() method to get the JDBC connection.

```java
publicDataBaseConnectivity()
{
    try
    {
        Hashtable parms = new Hashtable();
        parms.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.ejs.ns.jndi.CNIInitialContextFactory");
        v3Ctx = new InitialContext(parms);
        v3DS = (DataSource)v3Ctx.lookup(v3DataSourceName);
        dataConn = v3DS.getConnection(userId, userPassword);
    }
    catch (Exception e)
    {
        System.out.println("DataBaseConnectivity Exception occurred: (Build connection) " + e.getMessage());
        e.printStackTrace(System.out);
    }
}
```
When the JDBC connection is requested, we ensure that we still have the exclusive use of the connection and that has not been reclaimed by the DataSource or closed by the caller. If the connection has been closed, we attempt to request a new connection. The WebSphere Application Server Advanced Edition 3.0 DataSource objects have associated time-outs that you can evaluate if you have long running servlets.

Figure 39 shows an auxiliary method available for use by other objects.

As you can see, our code example becomes much simpler using a DataSource, as we do not have to deal with the complexities of the IBM Connection Manager support.

### 2.4.2 Migrating WebSphere Version 2.0 Servlets

In this section of the chapter, we migrate some of the servlets described in the redbook *SG24-5635 “Building AS/400 Applications for IBM WebSphere Application Server 2.0 Standard Edition”* to the IBM WebSphere Application Server 3.0 Advanced Edition environment. We look at the changes required to
allow them to function in the WebSphere Application Server Advanced Edition 3.0 environment.

2.4.2.1 SG24-5635 Redbook Servlet Refresher

The architecture of the version 2.0 servlets was designed to show you the benefit of a proper application architecture. That is, you can use the same functionality for applets, servlets, and applications with only minor changes. To demonstrate this, the example application is structured as follows:

![Diagram of Application Architecture]

**Figure 40. Application Architecture**

The application consists of multiple packages:

1. **access**
   - This package contains the classes responsible for database access.
   - JDBC CatalogSupport class is the super class for the JDBC based access classes. It provides methods that are shared among several JDBC based access classes. The database access is implemented using JDBC. The JDBC ItemsCatalog and JDBC PoolCatalog classes extend the JDBC CatalogSupport class. The JDBC ItemsCatalog class uses standard JDBC support to access the AS/400 system, while the JDBC PoolCatalog class uses WebSphere Standard Edition V2.0 Connection Manager support.

2. **nservlets**
   - This package contains classes that provide the end user interface. The servlet examples implement the HttpServlet abstract class. For example, the ItemServlet class in this package manages all the user interface oriented work like getting and interpreting the user responses that are received in the form of HTTP requests and generating the response in the form of HTML. To
get the items information that is included in the generated HTML, the ItemServlet class forwards the database access work to the JDBCItemCatalog class.

Please refer to redbook SG24-5635 for further information.

2.4.2.2 Migrating the Servlets

The servlet examples were developed using VisualAge for Java version 2.0. We export the servlets into a Jar file and then import them into a VisualAge for Java V3.0 project. We display the Problem pane. We immediately notice that the IBM Enterprise Toolkit for AS/400 V3.0 feature does supply all the base AS/400 Toolbox for Java classes used in these servlets.

We remove the IBM Enterprise Toolkit for AS/400 feature and create another project as shown in Figure 41 and import all the AS/400 Toolbox for Java classes from the AS/400 Toolbox for Java - Modification 2 Jar file into it.

Note

The beta version of VisualAge for Java Enterprise Edition 3.0 used to write this redbook did not contain all the base AS/400 Toolbox for Java classes required. To solve the problem, we replaced the AS/400 Toolbox for Java classes provided with the IBM Enterprise Toolkit for AS/400 feature with the classes from the AS/400 Toolbox for Java - modification 2 (5769-JC1). This may be a problem only found with the VisualAge for Java Enterprise Edition 3.0 beta driver. It may not be a problem with the released version.
After importing the AS/400 Toolbox for Java classes, we again display the Problem pane of the version 2.0 servlets project. As you can see in Figure 42 on page 49, most of the migration issues exist in the area of the IBM Connection Manager APIs and deprecated Java Servlet APIs. There are also some less obvious issues which we discuss later.
Using the information from section 2.4.1, “Migrating from IBM Connection Manager to DataSources” on page 38, we migrate these servlets to the new WebSphere version 3.0 DataSource support.

**The Access Package Classes**

There is a distinct class hierarchy within the access package that uses the version 2.0 IBM Connection Manager support as shown in Figure 43 on page 50.

We start by changing the JDBCCatalogSupportClass to use version 3.0 DataSource support and change the associated subclass to use the changed and enhanced methods.
Of the two sub-classes, only the JDBCPoolCatalog class uses any methods that use the IBM Connection Manager APIs. The JDBCItemsCatalog class does not require any modification.

**Changing the JDBCCatalogSupport Class**

We start by changing the import statements to match those required for the version 3.0 DataSource support as shown in Figure 44.

```java
package access;
import java.util.*;
import java.sql.*;
import com.ibm.db2.jdbc.app.stdextjavax.sql.*;
import com.ibm.ejs.dbm.jdbcext.*;
import javax.naming.*;
```

As we now work directly with JDBC connections and not the IBM Connection Manager, we enhance the freeConnection and getConnection methods to take advantage of this. As shown in Figure 45 on page 51, the freeConnection method now simply closes the JDBC connection and the underlying DataSource connection pooling mechanism reclaims the connection.
As shown in Figure 46, within the getConnection method, we now return a JDBC connection object instead of the connection manager connection. Notice the changes to the parameter list, which now consists of the DataSource object and a user id and password only.

```java
public void freeConnection(Connection dataConn)
{
    try
    {
        if (!dataConn.isClosed())
        {
            dataConn.close();
        }
    }
    catch (SQLException e)
    {
        System.out.println("release connection: "+ e.getMessage());
    }
    return;
}
```

```java
public Connection getConnection(DataSource v3DS, String userId, String userPassword)
{
    Connection dataConn = null;
    try
    {
        System.out.println("JDBCPoolCatalog: retrieving connection");
        dataConn = v3DS.getConnection(userId, userPassword);
    }
    catch (SQLException e)
    {
        System.out.println(e.getMessage());
    }
    return dataConn;
}
```

**Changing the JDBCPoolCatalog Class**

Next, we migrate the JDBCPoolCatalog class to the version 3.0 environment. We start by changing the import statements to match those required for the version 3.0 DataSource support as shown in Figure 47 on page 52.
As shown in Figure 48, we use the connectToDB() method to get the JDBC connection. We pass in the DataSource, the user id and password.

```java
package access;
import java.util.*;
import java.sql.*;
import com.ibm.db2.jdbc.app.stdext.java.sql.*;
import com.ibm.ejs.dbm.jdbcext.*;
import javax.naming.*;

public class JDBCPoolCatalog extends JDBCCatalogSupport
{
    private String connUserId = null;
    private String connUserPassword = null;
    private Context v3Ctx = null;
    private DataSource v3DS = null;

    public String connectToDB(String dataSourceName, String userid, String password)
    {
        try
        {
            connUserId = userid;
            connUserPassword = password;
            Hashtable parms = new Hashtable();
            parms.put(Context.INITIAL_CONTEXT_FACTORY, "com.ibm.ejs.ns.jndi.CNInitialContextFactory");
            v3Ctx = new InitialContext(parms);
            v3DS = (DataSource) v3Ctx.lookup(dataSourceName);
        }
        catch (Exception e)
        {
            System.out.println("JDBCPoolCatalog: (ConnectToDB) " + e.getMessage());
            e.printStackTrace(System.out);
            return "connection unsuccessfull";
        }
        System.out.println("connected successfully");
        return "connection successful";
    }
}
```

As shown in Figure 48:
- We use JNDI to get access to the DataSource (or any other object) from the naming space. We create a Hashtable object to hold the necessary parameters to access the naming space.
- We specify IBM EJS CNInitialContextFactory (com.ibm.ejs.ns.jndi.CNInitialContextFactory) as the Context factory object we wish to use.
- We get a Context object for the EJS JNDI interface.
- Using the Context object, we do a JNDI lookup for the logical name of the DataSource.
- The Context lookup returns a java object, which we cast to a DataSource object. All Datasource names exist in the "jdbc" subcontext of the root.
As shown in Figure 49, in the `getAllV()` method we call the `getConnection()` method to get the JDBC connection.

```java
public java.util.Vector getAllV()
{
  java.util.Vector aDataVector = new Vector();
  java.sql.ResultSet aResultSet = null;
  Connection dataConn = null;
  try
  {
    dataConn = getConnection(v3DS, connUserId, connUserPassword);
  }
  catch (Exception e)
  {
    System.out.println("Exception caught: " + e.getMessage());
    e.printStackTrace();
    //      throw e;
  }
  PreparedStatement psAllRecord = null;
  try
  {
    psAllRecord = dataConn.prepareStatement("select * from ITEM");
    aResultSet = psAllRecord.executeQuery();
    aDataVector = getRows(aResultSet);
  }
  catch (SQLException ex)
  {
    ex.printStackTrace();
  }
  finally
  {
    try
    {
      if (null != psAllRecord)
      {
        psAllRecord.close();
      }
      freeConnection(dataConn);
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
    return aDataVector;
  }
```

Figure 49. The `getAllV` method

As shown in Figure 50 on page 54, we make the same change to the `getRecord` method.
Finally, we update the servlets that utilize the new JDBCPoolCatalog class. For brevity, we only show the changes required to a single servlet, the ItemPoolServlet. Figure 51 on page 55 shows the init() method for the ItemPoolServlet.
We change the servlet initialization parameters to retrieve the name of the version 3.0 DataSource and pass this, along with the user id and password, into the new database connection method.

This servlet init() method also highlights another change to the IBM recommendations for specifying initialization parameters for servlets. WebSphere version 2.0 recommends the use of property files or XML based servlet files for initialization parameters. The recommendation for version 3.0 is that initialization parameters be defined with servlet definitions inside the Administrative Client.

**Using URL Encoding with WebSphere 3.0**

The original URL encoding APIs have changed for the Servlet 2.1 specification. The original APIs have been deprecated by two new APIs as shown in Table 4.

<table>
<thead>
<tr>
<th>Servlet 2.0 API</th>
<th>Servlet 2.1 API</th>
</tr>
</thead>
<tbody>
<tr>
<td>encodeUrl</td>
<td>encodeURL</td>
</tr>
<tr>
<td>encoderedirectUrl</td>
<td>encoderedirectURL</td>
</tr>
</tbody>
</table>

Table 4. URL encoding APIs

One of the highlighted warnings from VisualAge for Java 3.0, was for the CartAppletServlet uses the depreciated encodeUrl method of the servlet response object. As shown in Figure 52 on page 56, simply replacing this method with the new encodeURL method removes this warning.
Coding Servlets to find Resources

For the WebSphere Application Server Advanced Edition 3.0 support, the paths for a web application resources are resolved relative to the document root of the web application. For example, the methods ServletContext.getRealPath() and HttpServletRequest.getPathTranslated() return the application’s document root.

2.4.3 Java Server Pages Support

For simplification of National Language Support (NLS) or the good design strategy of separating the presentation layer from the business logic, perform business logic within a servlet and pass the response data to a Java Server Page (JSP) for display as shown Figure 53.

Figure 53. Servlet and JSP architecture

Figure 54 shows the code to set attributes and call a JSP from a servlet when using WebSphere Standard Edition version 2.0. We need to cast the HttpServletRequest and HttpServletResponse objects to their respective HttpServiceRequest and HttpServiceResponse objects.
The setAttribute and getAttribute methods of the HttpServletRequest class have been depreciated and are replaced by the setAttribute method of the servlet API 2.1 HttpServletRequest class. Version 2.1 of the servlet API introduces a RequestDispatcher object for each servlet or JSP. It handles all requests from any client. This may be a browser or in this case another servlet. It forwards the response and request objects to the specific servlet or JSP. This is similar to servlet filtering and shares some of the same restrictions. If a servlet retrieves a ServletOutputStream object or PrintWriter object from its response object, then an IllegalStateException exception will be thrown in the application.

The other important change that is not immediately obvious, is that the location of the JSP is relative to the web application document root. If the call to the JSP fails, this could be the problem.

As shown in Figure 55 on page 58, we convert the doPost method of the ItemJSPServlet example from the WebSphere Standard Edition redbook.
Notice the changes in Figure 55. We remove the use of the PrintWriter object and the outputHeader method, as this is not valid when we utilize the RequestDispatcher.
In Figure 56 we show the rest of the changes required for the servlet 2.1 API.

```java
session.putValue("sessionlist.items", parts);
}

// First create the indexed bean to pass to the JSP
ItemInfo[ ] items = new ItemInfo[parts.size()];
for (int i=0;i<parts.size();i++) {
    ItemInfo anItem = new ItemInfo();
    String[ ] aPart = (String[ ]) parts.elementAt(i);
    anItem.setItemNumber(aPart[0]);
    anItem.setItemDescription(aPart[1]);
    anItem.setItemPrice(aPart[2]);
    items[i] = anItem;
}

ItemQueryResults results =
    (ItemQueryResults)Beans.instantiate(this.getClass().getClassLoader(),"shopData.ItemQueryResults");
results.setNumberOfElements(new Integer(parts.size()));
results.setItemList(items);
flexLog("ItemJSPServlet: ready to call items.jsp");
//((com.sun.server.http.HttpServiceRequest) request).setAttribute("results", results);
request.setAttribute("results", results);
RequestDispatcher rd = GetServletContext().getRequestDispatcher("/jsp/items.jsp");
rd.forward(req, res);
return;
} catch (Throwable e) {
    flexLog("Exception occurred inside ItemJSPServlet: "+e);
    printError(out, e);
}
} // end of doPost()
```

Figure 56. Part 2 of the doPost method of the ItemJSPServlet

### 2.4.4 Summary of Migrating to the JSWDK 1.0 Specification

The JSWDK (JavaServer Web Development Kit) 1.0 is the reference implementation of JavaServer Pages 1.0 technology and the Java servlet API 2.1. Refer to the JSWDK specification for complete information concerning new and deprecated APIs. Table 5 on page 60 highlights a few of the new and deprecated classes and methods.
## Table 5. JSWDK 1.0 APIs

<table>
<thead>
<tr>
<th>Method or Class</th>
<th>Status and recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>RequestDispatcher</td>
<td>New. Use the forward method to forward a servlet response from one servlet to a second servlet for further processing. Use the include method to include part of the one servlet's response in the body of another servlet's response.</td>
</tr>
<tr>
<td>HttpSessionContext</td>
<td>Deprecated. See Session state for tips for sharing session information.</td>
</tr>
<tr>
<td>HttpSession.getSessionContext</td>
<td>Deprecated. For security reasons, no equivalent.</td>
</tr>
<tr>
<td>HttpSession.getMaxInactiveInterval</td>
<td>New. Sets the maximum time a session will be maintained by the servlet engine without a client request.</td>
</tr>
<tr>
<td>ServletRequest.getRealPath</td>
<td>Deprecated. Use ServletContext.getRealPath.</td>
</tr>
<tr>
<td>ServletContext.getServlet</td>
<td>Deprecated. Use ServletContext.getRequestDispatcher</td>
</tr>
<tr>
<td>ServletContext.getResource</td>
<td>New. Use this method to obtain a servlet resource by requesting its URL.</td>
</tr>
<tr>
<td>ServletContext.getResourceAsStream</td>
<td>New. Use this method to obtain a servlet resource (as an InputStream) from its servlet context.</td>
</tr>
<tr>
<td>encodeUrl and encodeRedirectUrl methods of HttpServletResponse</td>
<td>Deprecated. But the fix is easy. Change Url to URL in the method names.</td>
</tr>
<tr>
<td>HttpSession.isRequestedSessionIdFromUrl</td>
<td>Deprecated. Another easy fix. Change Url to URL in the method name.</td>
</tr>
<tr>
<td>HttpServletResponse.callPage()</td>
<td>Deprecated. Migrate to javax.servlet.RequestDispatcher().</td>
</tr>
</tbody>
</table>

### 2.5 JSP Support in WebSphere Version 3.0

WebSphere Application Server Advanced Edition 3.0 implements both the 0.91 and 1.0 versions of the Sun Java Server Pages specification. For a complete definition of the Sun JSP specification, refer to the Sun on-line documentation at:

2.5.1 To Port or not to Port, that is the Question
WebSphere Application Server Advanced Edition 3.0 provides support for the 0.91 JSP specification and as such you can simply re-use your existing JSPs with some minor changes.

The question of porting comes down to the requirement of the new 1.0 specification functionality. If you require JSP version 1.0 functionality, then you must migrate all the JSPs in the web application to the version 1.0 specification. This is required because within the definition of your web application you specify what level of JSP support you require.

If you need to have multiple JSP version support for your application then you must create two web applications. Under these circumstances, we recommend that you migrate your existing JSPs to version 1.0 for simplicity.

You should plan on migrating 0.91 level JSPs to the 1.0 specification. Eventually support for the 0.91 specification will be dropped by both Sun and IBM.

2.5.2 Leaving JSPs at the Sun 0.91 Specification Level
If you do not choose to migrate to the JSP 1.0 specification, you may need to make changes to your JSP code depending on your WebSphere Application Server Advanced Edition 3.0 setup.

2.5.2.1 Document Location
As discussed earlier in this chapter, an important change to be aware of when migrating an existing WebSphere version 2.0 JSP application to the WebSphere Application Server Advanced Edition 3.0 environment is the document root. If the JSP produces output that references images, sound clips or contains the HTML <Form> </Form> tags, you need to ensure that you correctly use the version 3.0 document root.

2.5.2.2 Batch Compile
WebSphere Application Server Advanced Edition 3.0 introduces a batch compile mechanism for JSPs. It supports the 0.91 specification level only. We recommend that you batch compile all of the JSP files associated with an application. Once the compiled files are placed on the system, the Application Server will monitor the JSP file for changes and automatically re-compile changed files. Please refer to the on-line documentation for more information.
Chapter 3. Introduction to Enterprise JavaBeans

JavaBeans have become widely accepted in the Java programming community as the way to create the client side for applications. JavaBeans allow application programmers to create reusable components and build applications with those components. The flexibility and ease-of-use associated with the component model, and the tools that make use of JavaBeans have helped application developers greatly. Extending this concept to server-side, business components adds considerable complexity. Developers must address the following items:

- Persisting the component data in a data source
- Distributing the components across a network
- Managing transactions
- Building the necessary security required by business applications

Enterprise JavaBeans (EJB) technology provides a logical extension to the JavaBeans concept. It is targeted at server-tier business logic development. It provides interfaces that insulate the programmer from the complexities and dependencies that are unique to a platform.

Enterprise JavaBeans technology is critical for the development of robust, Java-based business applications on the AS/400 system. In this chapter, we discuss the basics of EJB technology, what it offers application developers, its key features, and how EJB technology fits into the AS/400 system environment.

3.1 The Enterprise JavaBeans (EJB) Specification

Business applications that support a wide variety of users within or across business domains are usually very complex. The complexity of the domain requires application developers to understand the underlying domain technology and the business domain sufficiently to design and implement applications. Advancements in technology have increased the capabilities that can be integrated into business applications. At the same time, the new technology has increased the complexity of application development.

The Enterprise JavaBean (EJB) specification was defined by Sun Microsystems Inc. to address the issues concerned with developing applications. The mission of the EJB specification is to define an architecture for application development, which accomplishes the following goals:

- **Make it easier to write business applications** — By separating the business application from system services, the application developer can focus on the needs of the business application, which includes the business logic and the representation of business data.

- **Increase manageability of the systems environment** — By keeping the business logic and business data resident in a server environment, the application code is centrally managed. Administration of the application environment is simplified, allowing easier distribution of enhancements, fixes and extensions to the business application.

- **Promote reuse** — By defining business objects as components, an application developer can create new components, reuse existing components, or purchase components. By using existing components or
purchasing components, the application developer assembles the application rather than developing and testing new code.

Version 1.1 of the EJB specification defines the fundamental mission and concepts for creating enterprise server applications. Subsequent versions of the EJB specification will expand and fine tune the specification to define all aspects of the EJB environment.

3.2 Enterprise JavaBeans Architecture Definition

The Enterprise JavaBeans (EJB) architecture defines the fundamental concepts for Java server-based business applications. The architecture defined in the EJB specification, Version 1.1, is available from Sun Microsystems on the Internet at the Web site: http://www.java.sun.com/products.ejb/

The EJB architecture is a component based architecture for the development and deployment of server-based business applications. It allows the separation of business applications from the underlying system services. The following list breaks these statements down to help unravel the definition of the EJB architecture.

- **Component based** — The EJB architecture uses the concept of components to represent business objects, business logic or business tasks within the application.

- **Architecture** — The EJB architecture provides the structure for developing Java-based, object-oriented business applications. It is not a tool for building these business applications. However, it is an architecture for defining the application components.

- **Development and deployment** — The EJB architecture is Java-based, which allows many existing integrated development environments (IDE) to support creating and testing of the components. The application developer selects from a variety of tools that support the EJB specification and provides the services needed by the application. The details of deployment are separated from the business components. Therefore, deployment considerations are customized to the needs of the business environment without affecting the business components.

- **Server based business application** — In recent years, we have seen a movement toward client/server computing based environments, with increasing portions of the application residing on the client. The server is given the role of a data repository. By returning the application to the server, the ability to manage the application is simplified. The EJB specification allows you to distribute the application components across multiple servers to meet scalability and business structure needs.

- **Separation of the business application from the underlying system services** — With the EJB architecture, the application server environment manages and supports the system services. This removes the requirement for the business application to deal with low-level application programming interfaces (APIs) to perform its functions. However, these low-level APIs are still available and the application can make use of them.

Since the beginning of Java, it has mainly been used for client-side processing and Internet based applications. The EJB architecture is being accepted as the
standard for building object-oriented business applications in Java. Many server vendors are using this specification to define server environments, which is making the EJB specification a reality. Many businesses are now using Java and the EJB technology for building and deploying business applications on servers.

### 3.3 How EJB Technology Fits into the Java Architecture

EJB technology complements and extends the existing Java architecture. EJB technology extends the promise of Java by providing an application server environment that handles system services that would otherwise be handled by the application code. This allows EJB technology to extend the ease-of-use of Java by allowing the application developer to focus on business applications and let the application server focus on system services.

JavaBeans and Enterprise JavaBeans technology may seem to be on a collision course, but they actually complement each other. JavaBeans technology defines a component model with conventions which enable development tools to examine a JavaBean’s components to determine its capabilities. The development tools expose the properties and capabilities of a JavaBean to application developers. The application developer sets bean properties or connects the beans methods to other JavaBeans. Many JavaBeans components that exist today are used in graphical interface development.

EJB technology does not implement the same interfaces as JavaBean technology, nor does it follow the JavaBean event model. EJB technology has its own component model with distinct interfaces and conventions. It supports representing business logic and business data within an application environment. EJB components are intended for the server-side of business applications, while JavaBeans components fit nicely with the client-side of the application. To be complete, the EJB component model includes an application server environment which provides the necessary services to secure the application, provide transactions, and persist business data.

EJB technology allows business applications to take advantage of the internet, intranet and extranet by making it easier to extend existing business applications. EJB technology provides the extensions to the Java language that are needed before the business application developers take Java seriously.

### 3.4 Why EJB is Important

EJB technology is important both to application providers and the businesses that use their applications. The Java language allows application providers to develop applications that are platform independent. This is valuable to application users because it allows them to select the software that they need, independent of the hardware platform, that fits their business environment. If a user has a mixture of hardware, they can still run the same application in these different environments. At the same time, the application developer does not need multiple versions of the same application, with each version tuned to the specific platform.

This allows the application developers to focus on the business problem they are trying to solve and let the EJB server handle system services. EJB technology allows the developer to define the deployment environment at installation time, rather than when the application is designed. This allows the developer to
customize the application to the business domain, instead of to both the business domain and the specific hardware platform system.

Keeping the business logic and data on the server ensures that every user is working with the same master data and the same business logic. This also provides the application user better protection of intellectual assets that have been incorporated within the application. Servers, especially the AS/400 system, have better security packages than many client systems. When a business is connected to the Internet, ensuring that corporate data and intelligence is safe from unauthorized access is important.

The Java language, as the application development language of choice, is the stated direction of IBM. The IBM Server division is moving forward to make Java its language of choice. Many computer industry leaders have committed to making Java the language of electronic business, the Internet, and business applications. By moving to Java and EJB technology, application providers position themselves for the future.

### 3.5 Leveraging Java and EJB Technology

Java increases the productivity of developers writing applications and simplifies maintenance. The very nature of object-oriented technologies provides the basis for this statement. As a language, Java has taken the best of other object-oriented languages and applied various safeguards to minimize the problems programmers have experienced in the past. EJB technology makes developing applications with Java measurably easier. It hides all the system services details that are present when dealing with Java directly.

The EJB component model defines the infrastructure for creating an environment that separates system services from application business logic. The EJB server and EJB container deal with all the system service infrastructure included in the following list:

- Transactions
- Persistence
- Resources
- Security

This allows applications access to all of these services without dealing with their complexity. This translates into making application development activity easier. The developer can focus on business logic application development.

The EJB component model represents business logic and business data as components. Business data is reused within the same domain in various aspects of the business. For example, assume the customer information is used in the following ways:

- In an order entry application when a customer places or updates an order
- In accounting for creating invoices
- In marketing for target advertising campaigns

By creating the customer as a component, the applications that support these different organizations use the same customer component, which minimizes development and testing. As the development team identifies and creates a
repository of business object components, they begin to assemble applications to satisfy different end-user requirements.

EJB technology also leverages existing applications currently running business environments. It allows the extension of existing applications to provide new and additional functions for the business. The EJB specification identifies Common Object Request Broker Architecture (CORBA) as the means to provide inter-operability between different programming languages. This allows applications written in different languages, possibly running on different platforms, to interact with each other.

Even without CORBA, EJB objects can be used to wrapper existing application logic. For example, you can wrapper Component Object Model (COM) objects as EJB components and use them as part of a Java application.

3.6 EJB Architectural Overview

The EJB Specification consists of the following two major units:

- Components
  - Entity beans
  - Session beans

- Services
  - EJB container
  - EJB server

Figure 57 shows the relationship between entity beans and session beans, which are defined in the following list.

**Figure 57. EJB Architecture Overview**

**Entity beans**

These components represent business objects and contain business data. Because an entity bean contains business data, its contents are persisted for later use. Entity beans often reflect a row within an application.
An entity bean has methods to manage its data (get and set methods) and can support business logic pertaining to its business data.

Session beans
These objects perform business processes or tasks within a business process. A client uses a session bean to complete a particular task. Session beans are transient (their data is not persisted) and only exist for the life of the transaction. Session beans usually perform activities like obtaining or storing business data by using the entity beans or performing business logic that is maintained separately from the business data.

3.6.1 The EJB Server

In the EJB architecture, system services can be broken down into two distinct areas, which are often linked: the enterprise bean server and the enterprise bean container.

Figure 58 shows the relationship between the EJB container and the EJB server, which are defined in the following list.

### System Services Provider

**EJB container provider**
- Moves data between EJBean and database (or application)
- Protects EJBean provider from having to know EJB server interfaces
- Uses EJB server system services

**EJB server provider**
- System services such as transactions, security
- May provide a container to persist data to a specific data source
- May publish interfaces, so others can provide containers

![EJB System Services Providers Diagram]

**Figure 58. EJB System Services Providers**

**EJB containers**

Serve as the means to insulate the enterprise bean developer from the specifics of the EJB server services such as transaction management, security, and object distribution. It provides a simple interface for the enterprise bean and accesses the system services for it. This interface is referred to as the component contract for the Enterprise JavaBean.

The container is defined as a separate mechanism from the EJB server. The clarity of this separation is at the discretion of the tool vendor. The container manages Enterprise JavaBean objects. It manages the life cycle of the object (creation, maintenance and deletion), implements the security for the object and coordinates distributed transactions involving the object. By
performing these activities the container eliminates the need for bean developers to concern themselves with these issues.

**EJB server**

Allows the application developer to obtain the system services required by the application without directly dealing with lower level APIs. Figure 59 shows the architecture of the EJB server.

The EJB server carries the majority of the burden of dealing with the system environment by managing and coordinating the allocation of resources to the application. The following list describes the key system services (APIs) that the application server supports:

- **Security** — In the EJB component model, security is both granular and flexible. It is granular because it is configured at the component level or at the method level. It is flexible because it is configured outside the application code by using utilities provided by the application server provider.

- **Transactions** — In the EJB component model, transactions take two basic forms (it is important to notice that the EJB specification does not specifically distinguish between these two forms. This breakdown is to simplify the explanation). The following list describes the two basic forms:

  - **Database transactions** — Database transactions are granular and are configured by settings in the component. Database transactions reflect state changes in the component which are reflected in the database.

  - **Business transactions** — Business transactions represent business processes or tasks which can involve a variety of components and business logic. Business transactions often encapsulate changes to many of the components involved in the transaction.
**Persistence** — The EJB architecture eliminates the need for the components to deal with the persistence mechanism. Using JDBC as the database management technology, the application server interacts with a wide variety of data stores on the market today.

The EJB server supports these capabilities and manages many of the resources which are common within a Java application. These resources include thread pools and caching of objects. It is important to notice that many of the low-level APIs associated with Java programming are hidden from the application developer.

The EJB server provider is typically a company that produces middleware or a company that produces operating systems, such as IBM. Currently, there is no standard for the interface between a container and an EJB server. This may be introduced in a later release of the EJB specification. The EJB server provider often provides a container as well. EJB server support is currently available on the AS/400 system with WebSphere Application Server Advanced Edition 3.0, BEA WebLogic Application Server (formerly known as Tengah), BlueStone Software’s Sapphire/Web and Novera’s jBusiness.

### 3.6.2 Types of Components

The components (entity beans and session beans) break down into more discrete groupings:

**Note:** Entity beans are either container managed or bean managed. Session beans are *stateful* or *stateless*.

- **Container managed entity beans** — Delegate the reading and writing of bean attributes to the persistent datastore to the container that holds them. This allows the bean provider to set up a mapping schema from the bean attributes to the database columns outside of the entity bean itself. It also allows the entity bean developer to keep the persistence details separate from the business object (entity bean). Greater flexibility is achieved because the persistence information may be modified without affecting the entity bean. This makes the bean more reusable and portable.

- **Bean managed entity beans** — Support the situation where the bean developer needs to have more control over persisting a bean. Bean managed entity beans allow the bean provider to control the reading and writing of the bean attributes to the database. This gives the bean provider greater flexibility in providing their own persistence strategy. This can include persisting to a variety of databases or file types or providing nested database transactions. All the code necessary to map the beans attributes to the database are part of the bean itself.

Session beans are transient objects and perform operations on behalf of the client, such as accessing the database through the entity bean or performing business logic. Session beans can be involved in a transaction. However, they may not be recoverable in the case of a system crash. A session bean is stateless or stateful.

- **Stateless session beans** — Perform activities for the client but do not maintain any data. They can perform business logic and calculations. However, no instance variables are defined within the bean.
A stateful session bean — has data and maintains that data for the life of a transaction. If this data must be persisted, it must be forwarded to entity beans. Stateful session beans minimize the amount of interaction between the client and the server, making the application more efficient. However, it is imperative that any data that needs preserving is captured and forwarded.

Entity beans and session beans provide the heart of the business application by representing both the business objects (data) and the business logic (processes and tasks).

### 3.6.3 Component Content

A component consists of interfaces, the class, any required utility classes, and a deployment descriptor. Previous discussion of the components focused on the class itself. However, in a distributed environment, a component is accessed through its interfaces. As defined in the following list, each component has two interfaces:

- **Home interface** — Contains the methods for creating, deleting, and locating (finding) a particular instance of a bean.
- **Remote interface** — Contains the business methods that may be performed on a bean. The client interacts with the remote interface for the entity or session bean.

One of the key advantages of using EJBs is how easy it is to customize a bean component. The deployment descriptor provides the means to make this possible. Figure 60 shows how deployment descriptors are used.

![EJB Deployment Descriptors](image)

- **Deployment Descriptor is key**
- **Deployment tools**
  - Provide customization of EJB Deployment Descriptors
  - Generate classes which "mix-in" EJB business logic and container provider services for transactions, security, and persistence.

*Figure 60. Deployment Descriptors*
Each Enterprise JavaBean class requires a deployment descriptor. The Deployment Descriptor objects are used to establish the runtime service settings for an enterprise bean. These settings tell the EJB container how to manage and control the enterprise bean. The settings can be set at application assembly or application deployment time.

The DeploymentDescriptor object specifies how to create and maintain an Enterprise Bean object. This object defines, among other things, the enterprise bean class name, the JNDI namespace that represents the container, the Home interface name, the Remote interface name, and the Environment Properties object name. The DeploymentDescriptor object contains an array of ControlDescriptor objects, which specify the transaction semantics that should be applied to the enterprise bean, and an array of AccessControlEntry objects, which specify the security rules that should be applied to the enterprise bean. These settings are described in the following list:

- Bean JNDI Name
- Enterprise Bean Class Name
- Home Interface Class Name
- Remote Interface Class Name
- Environment Properties
- Reentrancy
- Control Descriptor (per bean or per method)
  - Run As Mode (Client, System, Specified Identity)
  - Isolation Level (read committed, read uncommitted, repeatable read, serializable)
  - Transaction Attribute (Bean Managed, Mandatory, Not Supported, Required, Requires New, Supports)
- Access Control Entry (per bean or per method)
  - method name
  - identities
  - (Entity Only)
    - Container Managed Fields
    - Primary Key Class Name
  - (Session Only)
    - Session Time-out
    - State Management Type (Stateful or Stateless)

These are the key settings found within the deployment descriptor. They provide flexible use of a component by customizing the component at runtime.

### 3.7 EJB Roles

The EJB specification identifies various application development and deployment roles. Both tool vendors and application providers have a variety of opportunities to take advantage of EJB technology and play an active part in providing applications using the technology. The EJB specification in the following list identifies these major roles:
• Enterprise JavaBean provider
• Application assembler
• Application deployer
• Server provider
• Container provider

3.7.1 Enterprise JavaBean Provider

As shown in Figure 61 on page 73, the Enterprise JavaBean provider provides the components for building business applications. Domain expertise is a critical characteristic of these providers. The objective is to create business components which are usable in a variety of business applications. The components implement a business process or a business object. EJB technology allows the provider to focus on business needs allowing them to develop components, without requiring them to have extensive knowledge of the system services.

3.7.2 Application Assembler

As shown in Figure 62, the application assembler, who is also an expert in the business domain, constructs the application. They are typically responsible for building the user interfaces for the application and providing the additional classes needed to complete the application. The assembler can customize Enterprise JavaBeans by changing the deployment information contained in the deployment descriptor.
3.7.3 Application Deployer

The application deployer is responsible for deploying the application in a specific system environment. The deployer is usually an expert on the function and features of the platform and the supporting technologies. The deployer maps the application to the platform environment and can make adjustments for items, such as security and the data store. The deployer can customize the Enterprise JavaBeans by changing the deployment information.

3.7.4 Server Provider

The EJB specification divides the server functions into two components: the server and the container. The EJB server provider produces the middleware (server) that communicates between the platforms operating system environment and the container or bean managed beans. The middleware, created by the server provider, provides the management system services shown in the following list:

- Distributed transactions
- Object distribution
- Security
- Data persistence

The server provider can elect to include the container as part of their product.

3.7.5 Container Provider

The container provider produces the EJB container which insulates the Enterprise JavaBean developer from the specifics of the EJB server services, such as transaction management, security and persistence. The container provider ensures that the container provides a simple interface to the Enterprise JavaBean. The container accesses system services for the EJB.
3.8 Using EJB Technology on the AS/400 System

The Java programming language provides numerous advantages as an application development language. For the AS/400 system, the main focus of Java application development has been client/server oriented. Because the AS/400 system does not have a native graphical user interface, it is not optimized for the display of client windows and graphics. In the client/server environment, the AS/400 system normally plays the role of a server. The Java programs run on the client and access the AS/400 database using the AS/400 host server programs.

Starting with OS/400 V4R2, Java programs can run on the AS/400 system. The main disadvantage in using Java for developing business applications is the extra programming required to manage the services required for the runtime environment.

EJB technology provides the means to make Java a viable language for AS/400 business application development. It focuses on server Java. EJB technology returns the focus of the application developer to the business application. The developer does not deal with system level services. The AS/400 system provides a secure and scalable environment for running Java applications, while EJB application servers provide server and container support.

3.8.1 AS/400 Java Overview

The Java environment on the AS/400 system is Java-compatible, which means it conforms to the Sun standard and can run 100% pure Java code without modification or re-compilation. The Java Virtual Machine (JVM) is implemented in the AS/400 machine interface (MI), providing a high degree of integration with the underlying system. The integrated JVM includes an advanced garbage collection algorithm which improves Java performance and scalability. In addition to the integrated, compatible runtime, the AS/400 Developer Kit for Java provides the additional commands, tools, and classes needed for Java development on the AS/400 system.

Like other Java environments, Java source code files on the AS/400 system are ASCII text files stored in the integrated file system (IFS). These are compiled into platform-independent class files (also stored in IFS) that are interpreted by the JVM at runtime. The AS/400 Java environment also includes a Java transformer that further optimizes Java class files and is used to create permanent, optimized 64-bit AS/400 program objects.

The Qshell interpreter is also required for Java development on the AS/400 system. Qshell is a command interpreter, based on POSIX and X/Open standards, which provides the Java development tools that are typically required for program development. With a few exceptions, the Java tools support the syntax and options available on most Java platforms.

Figure 63 on page 76 shows the different pieces that comprise the Java environment on the AS/400 system.
The integrated JVM is part of the system licensed internal code (SLIC) on the AS/400 system which is installed on every system. The AS/400 Developer Kit for Java is packaged as a separate, no-charge licensed program product (LPP) that must be installed on the AS/400 system. The Qshell interpreter is an OS/400 option that must be installed.

### 3.8.2 EJB Technology on the AS/400 System

An EJB server runs as an application above OS/400 on the AS/400 system. For the most part, current EJB servers are written in Java and require the AS/400 Developer Kit for Java on the system. The AS/400 strategy is to support application servers that deliver back-end systems integration and business-to-business e-commerce opportunities.

### 3.9 EJB Interoperability with Other Distributed Object Architectures

The EJB architecture supports distribution using the standard Java remote method invocation (RMI). Every enterprise bean has a home interface and a remote interface that declare the methods a client can indirectly invoke on the enterprise bean. The objects that implement these interfaces are Java RMI remote objects provided by the container. At runtime, the client gets a reference to the home or remote object, and invokes methods on that reference. The EJB container intercepts the method request and provides services before passing the request on to the enterprise bean. The communication between the references
(or stubs) on the client and the objects on the server is defined by the Java RMI specification.

### 3.9.1 CORBA

In addition to distributed object support using RMI, the EJB architecture also provides for EJB servers based on the Common Object Request Broker Architecture (CORBA). Through a standard EJB to CORBA mapping, an Interface Definition Language (IDL) interface can be generated from the remote interface. This mapping can be implicit when RMI over IIOP (Internet inter-ORB protocol) is used.

In addition to on-the-wire interoperability, CORBA-based EJB servers are required to use the OMG COS Naming Service for publishing and resolving the EJB home interface objects. The OMG Object Transaction Service is also required for transaction support in a CORBA-based EJB server.

These requirements allow a non-Java CORBA client to access enterprise beans using a CORBA-based EJB server. The client can use both CORBA and EJB objects within a transaction, and the transaction can span multiple CORBA-based EJB servers. WebSphere Application Server Advanced Edition 3.0 does not support CORBA clients.

### 3.9.2 COM

While the EJB architecture provides no specific support for Microsoft’s Component Object Model (COM), it is possible to wrap the client interface to an enterprise bean with a COM object to make it accessible from Visual Basic or Visual C++. Conversely, it is possible to wrap a COM object to make it accessible from Java.

There are a number of products available which provide COM to Java interface support. Linar’s J-Integra is one such product. It is a COM-Java bridging tool. Using J-Integra you can access ActiveX Components as though they were Java Objects, and you can access pure Java objects as though they were ActiveX Components.

For more information about J-Integra see:

http://www.linar.com/

### 3.9.3 IBM SanFrancisco

IBM SanFrancisco currently provides an infrastructure very similar in function to an EJB server. The SanFrancisco architecture predates the Enterprise JavaBean specification and was the basis for part of the EJB specification. SanFrancisco business objects are conceptually analogous to session beans and entity beans. For example, persistent entity business objects in SanFrancisco are very much like container-managed entity beans. SanFrancisco Command objects may perform much of the same function as a session bean. Although there is a significant amount of function in common between the SanFrancisco infrastructure and the EJB specification, they each provide some capabilities that the other does not. More significantly, they have much different scopes. The EJB specification addresses just the infrastructure. SanFrancisco addresses the infrastructure, but it also provides a rich set of business objects. These objects are designed to support
both general purpose business application development and also domain specific frameworks that target specific business application domains like General Ledger and Warehouse Management.

Work is in progress to support SanFrancisco using an EJB server for its infrastructure. Its entity objects will be converted to entity beans, and EJB programming interfaces will replace some of the current SanFrancisco programming interfaces. This will allow SanFrancisco components to be supported in a wide-range of EJB servers, as well as providing interoperability with CORBA objects on CORBA-based EJB servers. As part of IBM’s Enterprise JavaBeans strategy, SanFrancisco components provide a large set of cooperating application business components for the WebSphere family of EJB servers provided by IBM.

3.9.4 IBM Component Broker

Component Broker is comprised of two elements, Component Broker Connector (CBConnector) and Component Broker Toolkit (CBToolkit). CBConnector is an application runtime environment for business objects which includes a CORBA object request broker (ORB) and CORBA object services. CBToolkit is a set of development tools for this environment. Like SanFrancisco, Component Broker was conceived prior to the EJB specification but is extending its architecture to support enterprise beans as a standard component model for Component Broker Business Objects.

One of the strengths of Component Broker is its capability to connect components to existing applications and data. As part of the WebSphere Application Server, Component Broker containers enables enterprise beans to connect to a variety of back-end systems for persistence. WebSphere Application Server Advanced Edition 3.0 for AS/400 does not provide Component Broker support.

3.10 Conclusion

The Enterprise JavaBean architecture provides a component model for server applications. With EJBs, you get rapid application development and the rich graphical interfaces of a client without sacrificing the thin client manageability and the security of a server. EJBs accomplishes this by making it easy to partition an application into a user interface and the business logic. The user interface can be specified in a client application (written in Java, Visual Basic, PowerBuilder, and so on), or in HTML using a Java servlet. The server side business logic is packaged as Enterprise JavaBean components. Enterprise JavaBeans are easily deployed anywhere on the network, reused within other business applications running on disparate platforms, and easily managed from a remote console.

Enterprise JavaBean technology is not a tool to build applications. Rather, it is the architecture for defining components that can be used with a variety of tools. A stated goal of the EJB specification is to be the standard component architecture for building distributed object-oriented business applications in the Java programming language. Enterprise JavaBeans make it possible to build distributed applications by combining components developed using tools from different vendors.
Chapter 4. Overview of the Order Entry Application

In this chapter, we cover the RPG Order Entry application example. This application is representative of a commercial application, although it does not include all of the necessary error handling that a business application requires.

This section introduces the application and specifies the database layout. In Chapter 7, “Building Java Applications with Enterprise JavaBeans” on page 191, we convert the RPG Order Entry application to a web-enabled application. The goal is to use the existing RPG application to service both the web application and the host 5250 application.

4.1 Overview of the Order Entry Application

This section provides an overview of the application and a description of how the application database is used.

4.1.1 The ABC Company

The ABC Company is a wholesale supplier with one warehouse and 10 sales districts. Each district serves 3000 customers (30000 total customers for the company). The warehouse maintains stock for the 100000 items sold by the Company.

Figure 64 illustrates the company structure (warehouse, district, and customer).

![Figure 64. The Company Structure](image-url)
4.1.2 The ABC Company Database

The company runs its business with a database. This database is used in a mission critical, OLTP (online transaction processing) environment. The database includes tables with the following data:

- District information (next available order number, tax rate, and so on)
- Customer information (name, address, telephone number, and so on)
- Order information (date, time, shipper, and so on)
- Order line information (quantity, delivery date, and so on)
- Item information (name, price, item ID, and so on)
- Stock information (quantity in stock, warehouse ID, and so on)

4.1.3 A Customer Transaction

A customer transaction occurs based on the following series of events:

1. Customers telephone one of the 10 district centers to place an order.
2. The district customer service representative answers the telephone, obtains the following information, and enters it into the application:
   - Customer number
   - Item numbers of the items the customer wants to order
   - The quantity required for each item
3. The customer service representative may prompt for a list of customers or a list of parts.
4. The application then performs the following actions:
   a. Reads the customer last name, customer discount rate, and customer credit status from the Customer Table (CSTMR).
   b. Reads the District Table for the next available district order number. The next available district order number increases by one and is updated.
   c. Reads the item names, item prices, and item data for each item ordered by the customer from the Item Table (ITEM).
   d. Checks if the quantity of ordered items is in stock by reading the quantity in the Stock Table (STOCK).
5. When the order is accepted, the following actions occur:
   a. Inserts a new row into the Order Table to reflect the creation of the new order (ORDERS).
   b. A new row is inserted into the Order Line Table to reflect each item in the order.
   c. The quantity is reduced by the quantity ordered.
   d. A message is written to a data queue to initiate order printing.
4.1.4 Application Flow

The RPG Order Entry Application consists of the following components:

- **ORDENTD (Parts Order Entry)—Display File**
- **ORDENTR (Parts Order Entry)—Main RPG processing program**
- **PRTORDERP (Parts Order Entry)—Print File**
- **PRTORDERR (Print Orders)—RPG server job**
- **SLTCUSTD (Select Customer)—Display file**
- **SLTCUSTR (Select Customer)—RPG SQL stored procedure**
- **SLTPARTD (Select Part)—Display file**
- **SLTPARTR (Select Part)—RPG stored procedure**

To download the sample code used in this redbook, please refer to Appendix A., “Example Programs” on page 245, for more information.

ORDENTR is the main RPG program. It is responsible for the main line processing. It calls two supporting RPG programs that are used to prompt for and select end-user input. They are SLTCUSTR which handles selecting a customer, and SLTPARTR which handles selecting part numbers. PRTORDERR is an RPG program that handles printing customer orders. It reads order records that were placed on a data queue and prints them in a background job.
4.1.5 Customer Transaction Flow

The following scenario walks through a customer transaction showing the application flow. By understanding the flow of the AS/400 application, you can understand the changes made to this application to support a graphical client.

4.1.5.1 Starting the Application

To start the application, the customer calls the main program from an AS/400 command line:

```
CALL ORDENTR
```

When the Order Entry application is started, the display shown in Figure 66 appears.

![Figure 66. Parts Order Entry](image)

When the Parts Order Entry display appears, the user has two options:

- Type in a customer number and press the Enter key
- End the program by pressing either F3 or F12

If they do not know the customer number, the user can press F4 to view a window containing a list of available customers.
Figure 67. Select Customer

The user presses F12 to remove the window and return to the initial panel, or scrolls through the items in the list until they find the customer they want. By typing a 1 in the option field and pressing the Enter key, the user indicates their choice. The selected customer is then returned to the initial panel (Figure 68 on page 84).
After selecting a customer from the list, or typing a valid customer number and pressing the Enter key, the customer details are shown and an order number is assigned. An additional prompt is displayed allowing the user to type a part number and quantity.

If the user does not know the part number, they can press F4 to view a window containing a list of available parts (Figure 69 on page 85).
The user presses F12 to remove the window and return to the initial panel, or scrolls through the items in the list until they find the part they want. By typing a 1 in the option field and pressing the Enter key, they indicate their choice. The selected part is returned to the initial panel (Figure 70).
After selecting a customer from the list, or typing a valid customer number and pressing the Enter key, the part and quantity ordered are added to the list section below the part entry fields (Figure 71).

![Figure 71. Parts Order Entry](image1)

The user may type a 2 beside an entry in the list to change the order. When the Enter key is pressed, a window appears that allows the order line to be changed (Figure 72).

![Figure 72. Change Selected Order](image2)

The user chooses to press F12 to cancel the change, press F4 to list the parts, or type a new part identifier or different quantity. Pressing the Enter key validates
the part identifier and quantity. If valid, the order line is changed in the list and the window is closed.

![Figure 73. Completed Order](image)

In Figure 73, you see the quantity for Zoo Season Pass is changed to 3. When the order is complete, the user presses F6 to update the database. Then, an order is placed on the data queue for printing.

![Figure 74. Printed Order](image)
The printed order (Figure 74 on page 87) is created by a batch process. It shows the customer details and the items, quantities, and cost of the order.

### 4.1.6 Database Table Structure

The ABC Company database has eight tables:

- District
- Customer
- Order
- Order Line
- Item
- Stock
- Warehouse
- History

The relationship among these tables are shown in Figure 75.

![Diagram of Table Relationships](image)

*Figure 75. Table Relationships*

### 4.1.7 Order Entry Application Database Layout

The sample application uses the following tables of the database:

- District
- Customer
- Order
- Order line
- Stock
- Item (catalog)

The following sections describe, in detail, the layout of the database.
### 4.1.7.1 Tables

#### Table 6. District Table Layout (Dstrct)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>DID</td>
<td>District ID</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>DWID</td>
<td>Warehouse ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>DNAME</td>
<td>District Name</td>
<td>Character</td>
<td>10</td>
</tr>
<tr>
<td>DADDR1</td>
<td>Address Line 1</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>DADDR2</td>
<td>Address Line 2</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>DCITY</td>
<td>City</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>DSTATE</td>
<td>State</td>
<td>Character</td>
<td>2</td>
</tr>
<tr>
<td>DZIP</td>
<td>Zip Code</td>
<td>Character</td>
<td>10</td>
</tr>
<tr>
<td>DTAX</td>
<td>Tax</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>DYTD</td>
<td>Year to Date Balance</td>
<td>Decimal</td>
<td>13</td>
</tr>
<tr>
<td>DNXTOR</td>
<td>Next Order Number</td>
<td>Decimal</td>
<td>9</td>
</tr>
</tbody>
</table>

Primary Key: DID and DWID

#### Table 7. Customer Table Layout (CSTMR)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>CID</td>
<td>Customer ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>CDID</td>
<td>District ID</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>CWID</td>
<td>Warehouse ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>CFIRST</td>
<td>First Name</td>
<td>Character</td>
<td>16</td>
</tr>
<tr>
<td>CINIT</td>
<td>Middle Initials</td>
<td>Character</td>
<td>2</td>
</tr>
<tr>
<td>CLAST</td>
<td>Last Name</td>
<td>Character</td>
<td>16</td>
</tr>
<tr>
<td>CADDR1</td>
<td>Address Line 1</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>CCREDT</td>
<td>Credit Status</td>
<td>Character</td>
<td>2</td>
</tr>
<tr>
<td>CADDR2</td>
<td>Address Line 2</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>CDCT</td>
<td>Discount</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>CCITY</td>
<td>City</td>
<td>Character</td>
<td>20</td>
</tr>
<tr>
<td>CSTATE</td>
<td>State</td>
<td>Character</td>
<td>2</td>
</tr>
<tr>
<td>CZIP</td>
<td>Zip Code</td>
<td>Character</td>
<td>10</td>
</tr>
<tr>
<td>CPHONE</td>
<td>Phone Number</td>
<td>Character</td>
<td>16</td>
</tr>
<tr>
<td>CBAL</td>
<td>Balance</td>
<td>Decimal</td>
<td>7</td>
</tr>
<tr>
<td>CCRDLM</td>
<td>Credit Limit</td>
<td>Decimal</td>
<td>7</td>
</tr>
<tr>
<td>CYTD</td>
<td>Year to Date</td>
<td>Decimal</td>
<td>13</td>
</tr>
<tr>
<td>Field Name</td>
<td>Real Name</td>
<td>Type</td>
<td>Length</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
<td>-----------</td>
<td>--------</td>
</tr>
<tr>
<td>CPAYCNT</td>
<td>Payment</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>CDELCNT</td>
<td>Delivery Qty</td>
<td>Decimal</td>
<td>5</td>
</tr>
<tr>
<td>CLTIME</td>
<td>Time of Last Order</td>
<td>Numeric</td>
<td>6</td>
</tr>
<tr>
<td>CDATA</td>
<td>Customer Information</td>
<td>Character</td>
<td>500</td>
</tr>
</tbody>
</table>

Primary Key: CID, CDID, and CWID

Table 8. Order Table Layout (ORDERS)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWID</td>
<td>Warehouse ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>ODID</td>
<td>District ID</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>OCID</td>
<td>Customer ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>OID</td>
<td>Order ID</td>
<td>Decimal</td>
<td>9</td>
</tr>
<tr>
<td>OENTDT</td>
<td>Order Date</td>
<td>Numeric</td>
<td>8</td>
</tr>
<tr>
<td>OENTTM</td>
<td>Order Time</td>
<td>Numeric</td>
<td>6</td>
</tr>
<tr>
<td>OCARID</td>
<td>Carrier Number</td>
<td>Character</td>
<td>2</td>
</tr>
<tr>
<td>Oelines</td>
<td>Number of Order Lines</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>OLOCAL</td>
<td>Local</td>
<td>Decimal</td>
<td>1</td>
</tr>
</tbody>
</table>

Primary Key: OWID, ODID, and OID

Table 9. Order Line Table Layout (ORDLIN)

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Real Name</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>Order ID</td>
<td>Decimal</td>
<td>9</td>
</tr>
<tr>
<td>ODID</td>
<td>District ID</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>OWID</td>
<td>Warehouse ID</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>OLNBR</td>
<td>Order Line Number</td>
<td>Decimal</td>
<td>3</td>
</tr>
<tr>
<td>OLSPWH</td>
<td>Supply Warehouse</td>
<td>Character</td>
<td>4</td>
</tr>
<tr>
<td>OLIID</td>
<td>Item ID</td>
<td>Character</td>
<td>6</td>
</tr>
<tr>
<td>OLOQTY</td>
<td>Quantity Ordered</td>
<td>Numeric</td>
<td>3</td>
</tr>
<tr>
<td>OLAMNT</td>
<td>Amount</td>
<td>Numeric</td>
<td>7</td>
</tr>
<tr>
<td>OLDLVD</td>
<td>Delivery Date</td>
<td>Numeric</td>
<td>6</td>
</tr>
<tr>
<td>OLDSI</td>
<td>District Information</td>
<td>Character</td>
<td>24</td>
</tr>
</tbody>
</table>

Primary Key: OLWID, OLDID, OLOID, and OLNBR
4.1.8 Database Terminology

This redbook concentrates on the use of the AS/400 system as a database server in a client/server environment. In some cases, we use SQL to access the AS/400 database. In other cases, we use native database access.
The terminology used for the database access is different in both cases. In Table 12, you find the correspondence between the different terms.

Table 12. Database Terminology

<table>
<thead>
<tr>
<th>AS/400 Native</th>
<th>SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>Collection,Schema</td>
</tr>
<tr>
<td>Physical File</td>
<td>Table</td>
</tr>
<tr>
<td>Field</td>
<td>Column</td>
</tr>
<tr>
<td>Record</td>
<td>Row</td>
</tr>
<tr>
<td>Logical File</td>
<td>View or Index</td>
</tr>
</tbody>
</table>
Chapter 5. Developing Enterprise JavaBeans

The objective of this chapter is to show how the integration of WebSphere Application Server Advanced Edition 3.0 and VisualAge for Java Enterprise Edition 3.0 makes the development of Enterprise JavaBeans easy and productive. The following application developments scenarios are discussed in detail:

- Generating a Container Managed Persistence (CMP) entity bean from an existing database file
- Developing a CMP entity bean and manually mapping it to an existing database file
- Developing a CMP entity bean and generating data for mapping
- Developing a Bean Managed Persistence (BMP) entity bean

For an introduction to Enterprise JavaBeans please refer to Chapter 3, “Introduction to Enterprise JavaBeans” on page 63.

5.1 Generating a CMP Entity Bean from an existing Database File

We start our discussion from the most common scenario for AS/400 users. When application developers move to the Enterprise JavaBeans technology, it is most probable that they want to map any new entity beans to legacy data.

As the simplest example of this scenario, we show how a Stock entity bean can be automatically generated by VisualAge for Java Enterprise Edition 3.0 from the existing Stock database file. As legacy data, we use the Stock table from the RPG Order Entry application discussed in Chapter 4, “Overview of the Order Entry Application” on page 79. For the structure of the Stock table, see section 4.1.6, “Database Table Structure” on page 88.

5.1.1 Importing a Schema

The first step is to create a database schema which is used to create the Enterprise JavaBean. We can create the schema based on an existing AS/400 table.

In the Enterprise Bean pane of the EJB tab of VisualAge for Java Enterprise Edition 3.0 click the mouse right button to display the pop-up menu. As shown in Figure 76 on page 94, select **Open To-->Database Schemas**.
As shown in Figure 77, the Schema Browser panel appears.
In the Schema browser menu bar click **Schema-->Import/Export** **Schema-->Import Schema from Database**. As shown in Figure 78, you now see the Information Required dialog.

![Figure 78. Information Required dialog](image)

Enter the name of the schema. This can be any name, since we create the schema from an AS/400 table. We enter DirectMap, in the dialog as shown in Figure 78 and click OK. The dialog shown in Figure 79 requests the connection information for the AS/400 database.

![Figure 79. Database Connection Info](image)

As shown in Figure 79, enter:

- The AS/400 Toolbox for Java JDBC driver name in the Connection type drop-down list
- The AS/400 database URL in the Data source drop-down list
- A valid AS/400 user id and password in the corresponding fields
- Click the OK button

As shown in Figure 80 on page 96, the Select Tables panel appears.
In the Qualifiers pane, the list of all the AS/400 libraries that the user profile can access is shown. Select the library, we use APILIB, and click the Build Table List button. The list of the database tables contained in library APILIB appears in the Tables pane. Select the STOCK table and click the OK button.

As shown in Figure 81 on page 96, in the Schema Browser, a new schema named DirectMap is created.

If you select the new schema, the STOCK table appears in the Tables pane and its fields are listed in the Columns pane. Figure 81 shows the STOCK table as "<broken>>", because the VisualAge wizard could not find a key in the STOCK
file. This wizard is able to automatically generate a key descriptor only for files with primary key constraints. STOCK is a keyed physical file, but it does not have any primary key constraints. To manually add a key descriptor for the STOCK table, click the mouse right button in the Tables pane and select Edit table...

As shown in Figure 82, you now see the Table Editor.

![Figure 82. Table Editor](image)

Select the fields making up the key, STWIID and STIID, in the table columns pane and add them to the Primary key pane using the >> button, then click the OK button.

Now the STOCK table is correct, as shown in Figure 83, and can be used.

![Figure 83. Schema browser](image)
5.1.2 Creating an EJB Group

Now that the schema is correct, we can use it to create an EJB Group. Go back to the Enterprise Bean pane of the EJB tab of VisualAge for Java Enterprise Edition 3.0. As shown in Figure 84 on page 98, click the mouse right button to get the pop-up menu and select **Add --> EJB Group from Schema or Model**.

![VisualAge for Java Workbench](image)

*Figure 84. VisualAge for Java Workbench*

The Create EJB Group SmartGuide shown in Figure 85 on page 99 appears.
In the SmartGuide dialog:

- Enter the name of the project and package where you want to place the generated enterprise bean.
- Enter the name of the EJB group you want to create. In this example, we name it DirectFromAS400.
- Select the **Create from schema** radio button and choose the name of the just created schema map in the Available Schemas list.
- Click the Finish button.

A new EJB group, named DirectFromAS400, containing a new CMP entity bean is automatically generated. This is shown in Figure 86 on page 100.
The Types pane of the EJB tab shows the classes (StockBean and StockKey) and Interfaces (Stock, StockBeanFinderHelper and StockHome) generated for the Stock bean.

Click the F icon on top of the Types pane of the EJB tab. As shown in Figure 87 on page 101, the Types pane switches to the Property pane, where all the fields generated for the entity bean are listed.
As you can see, all the fields are container managed and the fields `stiid` and `stwid` are correctly marked as key fields. Each field has the same name as the corresponding field in the AS/400 STOCK table which it is mapped to. This can be a problem, especially if the AS/400 field names are particularly synthetic and not very meaningful.

Click on the C (Classes) icon on top of the Properties pane to switch back to the Types pane and select the StockBean class. All the methods of the selected class are listed in the Members pane. As shown in Figure 88 on page 102, you can see, setter and getter methods have been automatically created for each container managed field and all the methods have been added to the EJB remote interface.
As key fields should not normally be modified by end users, you may want to remove the setter methods of the two key fields from the EJB remote interface. To do this, select the two setter methods in the Members pane, click the mouse right button and select **Remove From—>EJB Remote Interface**.

The icon indicating a method added to the EJB remote interface is removed from the two selected methods is added, as shown in Figure 89 on page 103.
As shown in Figure 90, we can select the Stock Interface in the Types pane and check that the two setter methods do not appear in the corresponding Members pane.
Only an ejbCreate() method is generated in the class StockBean, with its counterpart create() method in the home interface. The generated ejbCreate() method receives only two parameters, whose values are used to set the key fields. You can add additional ejbCreate() methods with different signatures, to set other fields besides the key fields at create time. As shown in Figure 91 on page 105, we add an ejbCreate() method with parameters for warehouse id, stock id and stock quantity.
When adding ejbCreate() method, you can let VisualAge for Java Enterprise Edition 3.0 generate the corresponding create() method in the home interface. To do this, as shown in Figure 92, select the new ejbCreate() method, click the mouse right button and select Add To-->EJB Home Interface.

Figure 91. Adding ejbCreate() methods

Figure 92. Adding create() method to home interface
As shown in Figure 93 on page 106, an icon indicating a method added to the EJB home interface is added beside the new ejbCreate() method.

Figure 93. Adding create() method to home interface

As shown in Figure 94, if you select StockHome in the Types pane, the new create() method is listed in the corresponding Members pane.

Figure 94. Displaying the home interface methods
This can be a good starting point for you to add business logic methods to enforce business rules in the Stock entity bean.

5.1.3 Deploying the Stock Entity Bean

Now the Stock entity bean is ready to be deployed. A Stock entity bean deployed jar file must be created by VisualAge for Java Enterprise Edition 3.0, because the Stock bean is a CMP entity bean that maps legacy data. WebSphere Application Server Advanced Edition 3.0 can automatically create a deployed jar file and the corresponding database table only for Container Managed Persistence (CMP) entity beans that do not rely on a particular database configuration. That is, they do not store data in existing database tables. For more information on WebSphere Application Server Advanced Edition 3.0 deployment refer to section 5.3.1, “Deployment in the WebSphere Environment” on page 149.

As shown in Figure 95, to create the other classes needed to build the Stock bean deployed jar file, select the Stock bean in the Enterprise Beans pane, click the right mouse button and select Generate-->Deployed Code.

![Figure 95. Generating the deployed code](image)

As shown in Figure 96 on page 108, in the Types pane you can now see the classes generated by VisualAge for Java Enterprise Edition 3.0 for the Stock bean deployment.
Before deployment, you may want to look at and change the deployment descriptor. To do so, as shown in Figure 97 on page 109, select the Stock bean in the Enterprise Beans pane. Click the right mouse button and choose Properties.
As shown in Figure 98 on page 109, the Properties panel appears.
The Bean, Environment and Method tabs contain information is put in the bean deployment descriptor.

In the first field of the Bean tab you can change the JNDI name of the bean home interface, it defaults to the bean name.

In Figure 98 on page 109, the possible values for the bean overall transaction attributes are shown. Select the more appropriate one for your application. For a definition of the Enterprise JavaBeans transaction attributes please refer to Section 6.4.2, “Transaction Attribute” on page 189.

*** Warning ***

Note that, unless you specify TX_NOT_SUPPORTED as your bean transaction attribute, the AS/400 file used for mapping must be journaled.

If the Reentrant check box is checked, the same transaction is allowed to re-enter an enterprise bean.

In Figure 99 on page 111 the possible values for the bean overall isolation level are shown. Select the most appropriate value for your application. The isolation levels of Enterprise JavaBeans exactly match the JDBC isolation levels.

Table 13 shows the correspondence between the Enterprise JavaBeans/JDBC isolation levels and the AS/400 commitment control lock levels.

<table>
<thead>
<tr>
<th>EJB/JDBC Isolation Levels</th>
<th>AS/400 Commitment Control Lock Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSACTION_READ_COMMITTED</td>
<td>*CS</td>
</tr>
<tr>
<td>TRANSACTION_READ_UNCOMMITTED</td>
<td>*CHG or *UR</td>
</tr>
<tr>
<td>TRANSACTION_REPEATABLE_READ</td>
<td>*ALL or *RS</td>
</tr>
<tr>
<td>TRANSACTION_SERIALIZABLE</td>
<td>*RR</td>
</tr>
</tbody>
</table>

Table 13. EJB isolation levels
In Figure 100 the possible values for the bean overall run-as-mode values are shown.

- **CLIENT_IDENTITY** — The method executes with the security identity of the client.
- **SYSTEM_IDENTITY** — The method executes with the security identity of the system. This means that any bean called by this bean thinks it is called by the system.
- **SPECIFIED_IDENTITY** — The method executes with the security identity specified in the `RunAsIdentity` attribute.
The transaction attribute, isolation level and run-as-mode specified in the Bean tab are the defaults for all the methods in the bean. If you want to specify different values at the method level, select the method tab.

As shown in Figure 101 on page 112, to select the method to change, click the Add button.

![Figure 101. Adding method control descriptors](image)

As shown in Figure 102, the list of the bean methods appears.

![Figure 102. Selecting the method to change](image)

Select the method you want to change and click the OK button. As shown in Figure 103 on page 113, the Add Control Descriptor for the selected method panel appears.
Change the values as desired and click the OK button.

The Environment tab of the Properties panel, shown in Figure 104, allows you to enter information specific to the bean. This is useful to externalize values which depend on the specific deployment environment.

To set environment properties, enter the variable name and its value in the corresponding fields and click the Set button.

For an example of using the deployment descriptor to externalize application information, refer to section 6.3.5, “Stateful Session Bean - OrderEntryClerk” on page 184.

5.1.4 Exporting to the AS/400 System

You are now ready to generate the deployed jar file and to export it in an AS/400 IFS directory. As shown in Figure 105 on page 114, select the Stock bean in the Enterprise beans pane, click the right mouse button and select Export-->Deployed JAR.
As shown in Figure 106 on page 115, the Export to a Deployed Jar File SmartGuide appears.
Figure 106. Export to a Deployed JAR file SmartGuide

Insert a valid AS/400 IFS path and the name of the jar file you want to create in the Jar file field. By default .java source files are not included in the jar file; you may optionally include them by checking the .java check box. Debuggable class files and a compressed jar file are produced by default. Click the Finish button.

Now the Stock bean deployed jar file is on the AS/400 system and you can install it in WebSphere Application Server Advanced Edition 3.0.

5.1.5 Installing a JAR file in the WebSphere Environment

In this section of the chapter, we cover installing the JAR file in the WebSphere Application Server Advanced Edition 3.0 environment. As shown in Figure 107 on page 116, in the Topology tab of the Administrative Console select the EJB Container in which you want to install the Stock bean. Click the mouse right button and choose Create -->EnterpriseBean.
The dialog shown in Figure 108 appears.

You can optionally insert a value in the Name field. Click the Browse button. As shown in Figure 109 on page 117, you are prompted to select the JAR file to be deployed.
Figure 109. Finding the JAR file

Find the directory to which you exported the deployed jar file. Select it and find the JAR file to be deployed. In this case select `StockBeanDep.jar`. Click the Select button.

As shown in Figure 110, a confirm dialog appears.

Figure 110. Confirm dialog

Click the Yes button. Because the selected jar file contains a deployed Enterprise JavaBean, WebSphere Application Server Advanced Edition 3.0 does not have to deploy it. The dialog shown in Figure 111 appears.
Click the Create button

The Information dialog displayed in Figure 112 appears. Click the OK button.

As shown in Figure 113 on page 119, the Stock entity bean appears in the Topology tree of WebSphere Application Server Advanced Edition 3.0 under the EJB Container.
As shown in Figure 113, if you select the new enterprise bean, its property pane appears in the right part of the administrative console.

As shown in Figure 114 on page 120, selecting the DataSource tab allows you to specify a DataSource at the enterprise bean level. If you do not, the DataSource specified at the EJB Container level is used.
Figure 114. Stock enterprise bean DataSource tab

WebSphere Application Server Advanced Edition 3.0 creates a database file to map entity bean properties to upon bean startup, if the Create Table check box is selected. In our case the Stock bean maps to an existing database file, so the automatic creation of a mapping table is not necessary. For this reason you have to uncheck the Create Table check box and click the Apply push button before starting the bean.

Before starting the bean, you may want to change its deployment descriptor, set in VisualAge for Java Enterprise Edition 3.0. Use the bean Properties editor to reflect specific characteristic of your particular WebSphere Application Server Advanced Edition 3.0 deployment environment.

To do this, select the enterprise bean to display its property pane in the right part of the administrative console. Select the General tab and click the Edit ... button.

As shown in Figure 115 on page 121, the Deployment Properties panel appears.
Figure 115. Deployment properties

In the General tab you can change the JNDI name of the bean home by clicking the Set button.

As shown in Figure 116, the Entity tab shows the entity descriptor attributes, such as the primary key class and the container managed fields.

Figure 116. Entity tab

We do not use the Security tab to control security. Instead, as shown in Figure 117 on page 122, we use the Configure resource security task in the Tasks tab of the WebSphere Application Server Advanced Edition 3.0 Administrative Console to change the bean’s Run As Mode value. For more information on configuring security, please refer to the IBM WebSphere Advanced Edition for AS/400 on-line documentation. See the Security topic in the Administration section.
Figure 117. Configure resource security

The Transactions tab, shown in Figure 118, and Isolation tab, shown in Figure 119 on page 123, show the Transaction Attribute and Isolation level of the bean. These values cannot be modified.
The Environment tab can be used to add or change bean-specific property names and values.

Now that the Stock bean is installed in the WebSphere Application Server Advanced Edition 3.0 environment, you can test it.

5.1.6 Starting the Stock Bean

As shown in Figure 121 on page 124, to start the enterprise bean, select the Stock bean in the Topology tree of the WebSphere Application Server Advanced Edition 3.0 Administrative Console. Click the right mouse button and select Start.
As shown in Figure 122, the Stock bean icon in the Topology tree changes from red to blue. The current state changes to Running, indicating that the bean start has completed successfully.
5.1.7 Creating a Test Client

One of the most useful functions of the VisualAge for Java Enterprise Edition 3.0 Enterprise JavaBeans development environment is the automatic creation of a test client for each enterprise bean. To create a test client for the Stock bean, as shown in Figure 123 on page 126, select the Stock bean in the Enterprise beans pane, click the right mouse button and select Generate --> Test Client.
Figure 123. Creating a test client

5.1.8 Testing the Stock Bean

To test the Stock bean, go back to VisualAge for Java Enterprise Edition 3.0. As shown in Figure 124 on page 127, in the Enterprise beans pane select the Stock bean, click the mouse right button and select Run Test Client.
As shown in Figure 125, the Connect page of the Stock bean test client appears.
The Provider URL field defaults to **IIOP://**, meaning that the client should look for the Stock home in the local JNDI name space. Change it to **IIOP://sysname:xxxx**. Replace sysname with the name of your AS/400 system and xxxx with the port number on which the server is running. The default port number is 900. Click on the Connect button.

As soon as the Stock home interface is found in the AS/400 WebSphere Application Server Advanced Edition 3.0 JNDI name space, the Home interface page of the Stock bean Test client appears. This is shown in Figure 126.

![Home interface page](image)

**Figure 126. Home interface page**

The Methods listbox shows the list of the Stock home interface methods. Select the **findByPrimaryKey** method. In the Parameters pane, one line with two push buttons is shown. This indicates that the **findByPrimaryKey** method of the Stock home interface receives only one parameter and that its type is not one of the Java elementary types (for instance int, java.lang.String). Click the New button.

As the type of the parameter of the **findByPrimaryKey** method is **StockKey**, the Constructor page (Figure 127 on page 129) of the StockKey class is shown.
Select new StockKey(String, String) in the Constructor list box. Two entry fields appear in the Parameters pane, allowing you to enter the values of the fields (stwid (stock warehouse id) and stiid (stock item id)) making up the Stock bean key. Enter 0001 for stwid (stock warehouse id) and 000001 for stiid (stock item id). This forms the key of an existing record in the STOCK file. Click the Send button first and then the Done button.

As shown in Figure 128, the Remote interface page of the Stock bean Test client is shown.
The Methods listbox shows all the methods of the Stock remote interface. Select the getStqty() method and click the Send button to read the stock quantity from the AS/400 STOCK table through the CMP Stock entity Enterprise JavaBean.

As shown in Figure 129, the value of stock quantity is displayed in the Result line.

![Figure 129. Displaying the results](image)

Select the setStqty() method in the Methods listbox to modify the stock quantity in the AS/400 STOCK table through the CMP Stock entity. In the Parameters pane, one line with two push buttons is shown. The setStqty() method receives one parameter of type BigDecimal. This class maps the DECIMAL SQL data type (PACKED data type in AS/400 terms) of the STOCK table STQTY field to the stqty property in the Stock bean. Click the New button.

As shown in Figure 130 on page 131, the Constructor page of the BigDecimal class is displayed.
Select new BigDecimal(double) in the Constructors list box. Enter a new value for the stock quantity in the entry field shown in the parameters pane. Click the Send and then the Done push buttons.

As shown in Figure 131, the Remote interface page of the Stock bean Test client appears again.
Select the getStqty() method. In the Result line the new value just set for the stock quantity is shown.

Now try to insert a new record in the STOCK file through the Stock entity bean. To do this, go back to the Home interface page of the Stock bean Test client by selecting it in the Page drop-down list. As shown in Figure 132, select the create(String, String, BigDecimal) method.

![Figure 132. Stock Test Home interface](image)

Enter warehouse id/item id values, forming the key of a record not existing in the STOCK file yet, in the two entry fields in the Parameters pane. Click on the New button.

As shown in Figure 133 on page 133, the constructors page for the BigDecimal class appears.
In the Constructor page of the BigDecimal class select new BigDecimal(double) as BigDecimal constructor. Enter a value for the stock quantity in the entry field shown in the parameters pane. Click the Send and then the Done pushbuttons.

As shown in Figure 134, the Remote interface page of the Stock bean Test client a new remote object is displayed marked as #2 in the Remote objects drop-down list, and a new record in the STOCK file has been added.
To complete the test of the Stock entity bean you may want to delete the record just added in the STOCK file through the Stock entity bean. To do this, select the remove() method in the Methods list box and click the Send button.

The STOCK record will be deleted and instance #2 of the Stock bean invalidated. Any attempt to use one of its methods will result in an exception, as shown in Figure 135.

![Image of Stock entity bean with remove() method highlighted and exception message]

Figure 135. Attempting to use a deleted bean

5.2 Developing a CMP Entity Bean with Mapping to an Existing Table

In the previous section of this chapter we saw how easy it is to generate the CMP entity bean directly from the AS/400 database table you want to map it to.

However this technique has some disadvantages:

- One property is generated in the entity bean for every field in the database file.
- The property names are exactly the same as the field names in the AS/400 table.

In this section of the chapter we discuss an alternative technique to map CMP entity beans to existing database files. Our objective is to create a Stock entity bean whose properties map to only some of the AS/400 STOCK table columns and use meaningful property names.

The first action to do is to import the VisualAge for Java Enterprise Edition 3.0 schema of the STOCK table from the AS/400 database table. This activity has already been illustrated in Section 5.1.1, “Importing a Schema” on page 93. Let’s assume you have already created a schema named DirectMap containing the STOCK table.
First create an EJB Group. In this example, we name it ManualMap.

As shown in Figure 136, in the Enterprise Bean pane of the EJB tab of VisualAge for Java Enterprise Edition 3.0, click the mouse right button and select **Add-->Enterprise Bean** ...

![Figure 136. Adding an enterprise bean](image)

As shown in Figure 137 on page 136, the Create Enterprise Bean SmartGuide is displayed.
Figure 137. Create Enterprise Bean SmartGuide

In the SmartGuide dialog:

- Enter Stock in the Bean name field.
- Select Entity bean with container-managed persistence (CMP) fields in the Bean type drop-down list.
- Enter the name of the project and of the package for the entity bean classes.
- Click on the Next button.

As shown in Figure 138 on page 137, the Define Bean Class Attributes and Interfaces SmartGuide appears.
The names for home interface, remote interface and key class are automatically generated and the Create finder interface check box is checked by default. Click the **Add...** button besides the Add CMP fields to the bean listbox.

The Create CMP field SmartGuide shown in Figure 139 is displayed.

Enter a meaningful name for the warehouse id field in the Field Name field.
As seen in Section 5.1, “Generating a CMP Entity Bean from an existing Database File” on page 93, the automatic generation of an entity bean from an AS/400 file takes care of the mapping between database types and Java types. In our new example, we control the mapping. Refer to the JDBC documentation for a correct mapping. Table 14 shows the mapping of the most common database types.

<table>
<thead>
<tr>
<th>AS/400 Type</th>
<th>SQL Type</th>
<th>Java type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>CHAR</td>
<td>java.sql.String</td>
</tr>
<tr>
<td>PACKED</td>
<td>DECIMAL</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>ZONED</td>
<td>NUMERIC</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>java.sql.Date</td>
</tr>
<tr>
<td>BINARY</td>
<td>INTEGER</td>
<td>int</td>
</tr>
<tr>
<td>FLTDBL</td>
<td>FLOAT</td>
<td>float</td>
</tr>
<tr>
<td>FLTDBL</td>
<td>DOUBLE</td>
<td>double</td>
</tr>
</tbody>
</table>

Table 14. AS/400 to Java mapping

As the warehouse id field type in the STOCK file is of type CHAR, select java.lang.String in the Field Type drop-down list. Because it is part of the STOCK file key, check the Key field check box. Click on the Finish button.

Repeat the operation for the other character key field, item id.

Next add the quantity and year to date fields. Use Figure 140 as a guide.
• For the quantity and year-to-date field, enter `java.math.BigDecimal` in the Field type drop-down list. You can use the **Browse...** button for help find the class you need. The mapping database fields are of type PACKED packed.
• Uncheck the **Key Field** check box.
• We recommend that you check the **Access with getter and setter methods** check box. Setter and getter methods will be automatically generated in the StockBean class.
• Optionally, you can check the **Promote getter and setter methods to remote interface** check box to have them in the Stock interface as well.

As shown in Figure 141, when you finish adding fields to the Stock entity bean, click the Finish button in the Define Bean Class Attributes and Interfaces SmartGuide.

![Define Bean Class Attributes and Interfaces SmartGuide](image)

**Figure 141. Define Bean Class Attributes and Interfaces SmartGuide**

The following figures (Figure 142 on page 140 to Figure 145 on page 143) show the generated classes and methods. Note the meaningful names of properties and methods.

This is the starting point for you to add new create() methods and business logic methods to enforce business rules in the Stock entity Enterprise JavaBean.
Figure 142. StockBean class
Figure 143. Stock interface
Figure 144. Stock properties

```java
import java.rmi.RemoteException;
import java.security.Identity;
import java.util.Properties;
import javax.ejb.*;

/**
 * This is an Entity Bean class with CMP fields
 */
public class StockBean implements EntityBean {
    private javax.ejb.EntityContext entityContext = null;
    public java.lang.String itemID;
    public java.math.BigDecimal quantity;
    final static long serialVersionUIDUID = 3206093459760846163L;
    public java.lang.String warehouseID;
    public java.math.BigDecimal yearToDate;
}
```
5.2.1 Mapping the Stock Bean

Before deploying the Stock bean, you must map it to the existing AS/400 STOCK table, already imported in VisualAge for Java Enterprise Edition 3.0 in the DirectMap database schema.

To do this, click the right mouse button in the Enterprise beans pane and select Open To-->Schema Maps.

As shown in Figure 146, in the Map Browser panel, click the right mouse button in the Datastore Maps pane and select New EJB Group Map ...
The New Datastore Map panel shown in Figure 147 appears.

- Enter the datastore map name, Manual, in the Name field.
- Select the EJB Group containing the Stock bean, ManualMap, in the EJB Group drop-down list.
- Select DirectMap, the database schema containing the STOCK table, in the Schema drop-down box.
- Click the OK button.

As shown in Figure 148 on page 145, in the Map Browser panel, select the map generated from the EJB Group in the Datastore map pane. The list of the CMP entity beans contained in the EJB Group (in our case only Stock) appears in the Persistent Classes pane. In the Table Maps pane, click the right mouse button and select New Table Map-->Add Table Map with No inheritance...
In the resulting dialog shown in Figure 149, select STOCK in the Table drop-down list and click the OK button.

As shown in Figure 150 on page 146, in the Table Maps pane the new table map STOCK appears.
Select STOCK from the Table Maps pane. Click the right mouse button and choose **Edit Property Maps**...

As shown in Figure 151, the Property Map Editor panel is displayed.

In the Class Attributes pane all the CMP fields of the Stock entity bean are listed. Select the first one and click the button of the corresponding drop-down list in the Map Type pane.

As shown in Figure 152 on page 147, select Simple in the Map Type drop-down list and click the button of the corresponding drop-down list in the Table column pane.
As shown in Figure 153, select the STOCK file field to which you want to map the Stock bean CMP field from the Table column drop-down list. In this case we choose STWID.

Repeat the operation described from Figure 151 to Figure 153 for all the CMP fields of the Stock bean to manually map them to the AS/400 STOCK table fields. The Property Map Editor attributes should look similar to Figure 154 on page 148.
Click the OK button.

The manual mapping is now complete. As shown in Figure 155, in the Property maps pane of the Map Browser Panel the list of the mapping fields is displayed.

Now the Stock Bean is ready to be deployed by VisualAge for Java Enterprise Edition 3.0 and to be installed in WebSphere Application Server Advanced Edition 3.0 environment. This activity has already been illustrated in the previous section of the chapter. See Section 5.1.3, “Deploying the Stock Entity Bean” on page 107 for details.
Be aware that AS/400 STOCK table fields not mapping any Stock bean properties must be null capable fields, otherwise inserts through the Stock bean create() method will fail.

5.3 Developing a CMP Entity Bean and Generating Data for Mapping

Next, let’s examine a different application development scenario. Suppose we use an object-oriented analysis tool to design a brand new Enterprise JavaBeans based application. Now we want to use VisualAge for Java Enterprise Edition 3.0 to develop CMP entity beans and then we want to run them in the WebSphere Application Server Advanced Edition 3.0 environment on the AS/400 system.

The activity to create the Stock CMP entity bean is much the same as described previously in section 5.2, “Developing a CMP Entity Bean with Mapping to an Existing Table” on page 134.

Once you have created the Stock Enterprise JavaBean, you can choose between:

1. Deploying in the WebSphere Application Server Advanced Edition 3.0 environment and letting it create the database table for the bean mapping.


5.3.1 Deployment in the WebSphere Environment

First you must create an EJB (that is, not deployed) jar file for the Stock entity bean.

As shown in Figure 156 on page 150, in the Enterprise beans pane of the EJB tab of VisualAge for Java Enterprise Edition 3.0, select the Stock bean, click the right mouse button and choose Export--> EJB JAR...
As shown in Figure 157, the Export to an EJB Jar file dialog appears.
Insert a valid AS/400 IFS path and the name of the jar file you want to create in the Jar file field. By default, .java source files are not included in the jar file; you may optionally include them by checking the .java check box. Debuggable class files and a compressed jar file are produced by default. Click the Finish button.

5.3.2 Installing in the WebSphere Environment

Now the Stock bean EJB jar file is on the AS/400 system and you can install it in the WebSphere Application Server Advanced Edition 3.0 environment.

As shown in Figure 158, in the Topology tab of the Administrative Console, select the EJB Container in which you want to install the Stock bean. Click the mouse right button and choose Create ...-->Enterprise Bean.

![Figure 158. Creating an enterprise bean](image)

The dialog shown in Figure 159 appears.
Optionally insert a name in the Name field and click the **Browse**... button. From the select dialog, as shown in Figure 160, find the directory where you exported the EJB jar file.

Select the JAR file and click the **Select** button. As shown in Figure 161 on page 153, a confirm dialog appears. Click the **Yes** button.
As the selected JAR file contains a non-deployed enterprise bean, the confirm dialog shown in Figure 162 appears. Click the Yes button.

The message box shown in Figure 163 is displayed during the deployment process.

The dialog shown in Figure 164 on page 154 appears as soon as the entity bean is deployed. Click the OK button.
Next, we create the enterprise bean in WebSphere Application Server Advanced Edition 3.0. You are returned to the deployment dialog shown in Figure 165.

Click the Create button. The Information dialog displayed in Figure 166 appears. Click the OK button.

As shown in Figure 167, the Stock entity bean appears in the Topology tree of WebSphere Application Server Advanced Edition 3.0 under the EJB Container.
If you select the new enterprise bean, its property pane appears in the right part of the administrative console. As shown in Figure 168, select the DataSource tab.

You can specify a DataSource at enterprise bean level, but if you do not, the DataSource specified at EJB Container level is used.

WebSphere Application Server Advanced Edition 3.0 automatically creates a database file to map entity beans properties upon bean startup if the Create table check box is selected. This is the default. In this case, we want to create the table. Click the right mouse button on the Stock entry in the Topology tree and select Start.

Figure 167. Stock enterprise bean in the Topology pane
The Stock bean icon in the Topology tree changes from red to blue, indicating that the bean start has completed successfully. Also the Create table check box in the Data source tag of the enterprise bean properties is unchecked now, so that the database file for mapping is not recreated.

The database files used to map entity beans are always created by WebSphere Application Server Advanced Edition 3.0 in library EJB. The file name always follows the convention `<beanname>BEANTBL`. In this example a file named STOCKBEANTBL is created in library EJB.

As shown in Figure 169, if we display the object created, the name STOCK00001 is shown in because STOCKBEANTBL is longer than ten characters. We can
verify that the name STOCKBEANTBL is used with interactive SQL. As shown in Figure 170, we start interactive SQL (STRSQL) and enter the following statement:

```
Select table_name, table_schema FROM systables WHERE table_schema = 'EJB' and table_name like 'STOCK%'
```

As shown in Figure 171, the name of the table is **STOCKBEANTBL**.

Figure 172 shows the output of the display file field description (DSPFDD) command for the STOCKBEANTBL table.

You can see that the two BigDecimal properties of the Stock entity bean are mapped to fields of data type BLOB. Other tests show that java.sql.Date is
mapped to AS/400 data type DATE and java.util.date is mapped to BLOB as well. This will definitely cause problems. You would not want BigDecimal fields mapped as BLOBS (Binary Large Objects). If you use BigDecimal and Date properties, you should use VisualAge for Java for deployment to insure that the table mapping is correct. We discuss this option in the next section.

5.3.3 Table Creation and Deployment with VisualAge for Java

To deploy with VisualAge for Java Enterprise Edition 3.0, the entity bean must be mapped to a database table. In section 5.2, “Developing a CMP Entity Bean with Mapping to an Existing Table” on page 134 we discuss how to map an entity bean to an existing database file. We now show how to build Database Maps and Schema Maps from an entity bean and how to create the mapping file.

As shown in Figure 173, in the Enterprise Beans pane of the EJB tab of VisualAge for Java Enterprise Edition 3.0, select the Stock bean, click the right mouse button and choose **Add-->Schema and Map from EJB Group.**

![Figure 173. Adding a schema and map from an EJB](image)

If you open the Schema Browser, as shown in Figure 174 on page 159, you see that a Database Schema containing a Stock Table has been automatically created. Its fields, generated from the Stock bean properties information, are listed in the Columns pane.
Figure 174. Schema browser

As shown in Figure 175, select Stock in the Tables pane, click the right mouse button and choose Edit Table...

Figure 175. Editing the Stock table

As shown in Figure 176 on page 160, the Table Editor panel appears.
The Qualifier field is blank by default. In this case, VisualAge for Java Enterprise Edition 3.0 attempts to create the table in a library with the same name as the specified database user id. To specify the AS/400 library where the database table should be created, enter the library name in the Qualifier field and click the OK button.

As shown in Figure 177 on page 161, to create an AS/400 database file, select the EJB Group containing the Stock bean in the Enterprise bean pane and choose Add To--->Server Configuration.
Figure 177. Creating a database file

As shown in Figure 178 on page 162, The EJB Server configuration panel appears.
Select the added EJB server, click the right mouse button and choose Properties.

As shown in Figure 179, the Properties for EJB Server panel is displayed.

- Enter the AS/400 JDBC database URL in the Data Source field.
- Select the AS/400 Toolbox for Java JDBC Driver in the connection type drop-down list.
- Enter a valid AS/400 user id and password in the corresponding fields.
- Click the OK button.
As shown in Figure 180, select the added EJB server, click the right mouse button and choose **Create Database Table**.

![Create Database Table](image)

**Figure 180. EJB Server Configuration**

A message box like the one displayed in Figure 181 may be displayed. In this case the library specified in the Table Editor already exists and in this case it is not an SQL collection. The created table can not be journaled.

![Create Table - Problems Occurred](image)

**Figure 181. Create Table - Problems Occurred dialog**

As shown in Figure 182 on page 164, if you display the generated file field description, you will see that the BigDecimal properties of the Stock bean have been correctly mapped to PACKED fields.
If you go back to VisualAge for Java Enterprise Edition 3.0 and you open the Map Browser, you will see that:

- A Datastore map has been automatically created for the EJB Group containing the Stock bean.
- The Stock Persistent class has been linked with the automatically generated Table map Stock.
- Property maps have been automatically created.

All this is all shown in Figure 183.

---

**Figure 182. Display File Field Description output**

**Figure 183. VisualAge for Java - Map Browser**

Now you can generate the deployed code for the Stock bean in VisualAge for Java Enterprise Edition 3.0, export the deployed jar file to the AS/400 system and create the enterprise bean in WebSphere Application Server Advanced Edition.
3.0. This is described in section 5.1.3, “Deploying the Stock Entity Bean” on page 107 from Figure 95 to Figure 113.

5.4 Developing a BMP Entity Bean

After the discussion of all the possible alternatives of developing and deploying a Container Managed Persistence (CMP) entity bean, let’s look at Bean Managed Persistence (BMP) entity beans.

You may want to develop BMP, instead of CMP, entity beans for two main reasons:

- Performance
- Complex entity/database mapping

BMP application development is more complicated. However, VisualAge for Java Enterprise Edition 3.0 can still help you.

To create a BMP entity bean, from the VisualAge for Java Enterprise Edition 3.0 WorkBench, highlight the project to hold the new bean. Right click and select Add-->Enterprise Bean from the pop-up menu.

To create a BMP entity bean, select Entity bean with bean-managed (BMP) persistence fields in the Bean type drop-down list, as shown in Figure 184.

![Figure 184. Create Enterprise Bean SmartGuide](image)

Enter the information for the new bean as shown in Figure 184 and click the Next button. In the Define Bean Attributes and Interface SmartGuide, the check box to create a finder helper class and the button to add CMP fields are automatically disabled, as shown in Figure 185 on page 166.
Click the Finish button to generate the bean.

As shown in Figure 186 on page 167, two classes (\(<\text{beanname}\>\)Bean and \(<\text{beanname}\>\)Key) and two interfaces (\(<\text{beanname}\>\) and \(<\text{beanname}\>\)Home are automatically created with all the methods required by the Enterprise JavaBeans specifications.
You still have the ability to add methods to the remote and home interfaces. However no fields, no key fields, no field setter and getter methods are generated. They should be manually created by the developer. Not only is creating methods to enforce business rules a developer responsibility, so is entity data persistence.

The bean developer must:

- Insert a new record in the mapping AS/400 file in the `ejbCreate()` method of the `<beanname>`Bean class.
- Retrieve data from a record of the mapping AS/400 file and place it in the bean properties in the `ejbLoad()` method.
- Update the AS/400 file record mapping the bean in the `ejbStore()` method.
- Delete the AS/400 file record mapping the bean in the `ejbRemove()` method.

### 5.5 Deployment

The same entity bean can be deployed in two different EJB Containers, provided each deployed instance has a different JNDI Home Name. To change the JNDI Home Name set in VisualAge for Java Enterprise Edition 3.0, edit the general tab...
of the deployment property panel in WebSphere Application Server Advanced Edition 3.0 before starting the entity bean.

The two deployed instances of the same bean can access data in different libraries, provided:

- VisualAge for Java Enterprise Edition 3.0 made the deployment
- Before VisualAge for Java Enterprise Edition 3.0 deployment, you blank out the Qualifier text field in the Table editor panel for the Table in the Database schema used for the entity bean mapping.
- The two instances of the entity bean use different DataSources. The two DataSources may use the same JDBC driver, but must have different database names, for instance AS20/LIB1 and AS20/LIB2.
Chapter 6. AS/400 EJB Application Development Scenario

In this chapter, we discuss how the example RPG application discussed in Chapter 4, “Overview of the Order Entry Application” on page 79 can be implemented using EJB components. The application architecture is detailed by identifying the objects, using Unified Modeling Language (UML) diagrams. These objects are identified as entity and session beans. We discuss the reasons behind these selections, the implementation details of these choices, and how these EJB components are used to create various types of applications.

6.1 Order Entry Application Architecture with Objects

This architecture example assumes that the database tables already exist, and that an object architecture is built surrounding them. Initially, it is important to name the identifiable objects in the application. A variety of methods exist for creating an object-oriented application. One common way is to describe the process and highlight the nouns, as a starting point for identifying the objects.

In our example, a customer resides in a particular district. The customer contacts an order entry clerk to place an order for one or more items. Each item appears as an order detail line in the order. Before the order can be placed, the clerk checks to see if there is enough of the item in stock.

This list of objects corresponds to the tables identified in Chapter 4, “Overview of the Order Entry Application” on page 79, which also identify the fields contained in the tables. An object diagram corresponding to these tables is shown in Figure 187 on page 170.
In Figure 187, we make one further refinement. We separate out an address object to hold the common address information that is required by the customer and the district.

This list of objects represents all of the tables. However, the description does include another noun: the order entry clerk (OrderEntryClerk). This object becomes important in the next step of identifying objects; identifying the relationships between the objects. The OrderEntryClerk is an object with the primary purpose of acting on other objects. The relationships between the identified objects are either containment or uses relationships. Figure 188 on page 171 shows all of the objects that were previously identified and the relationships between them.
Both the customer and the district class contain an address. This provides reuse, one of the key tenets of object-oriented programming. As shown later in this chapter, reuse as a goal, may need to be overlooked to gain other advantages, such as making use of some of the programming facilities. The order contains a collection of order details. Because the order entry clerk initially takes down the order information, it also needs to contain a collection of order details (OrderDetails). This collection of order details is used as a parameter when calling the method necessary to create a new order.

The uses relationships primarily involve the OrderEntryClerk, as well. The OrderEntryClerk acts on several different objects. It retrieves a list of customers when the order is initiated. Later, when an order is being placed, it verifies the existence of a specific customer. It also retrieves a list of items to be ordered. As requests for creating order details are received by the OrderEntryClerk, it checks the stock availability of the item being ordered. After all of the order details are created, the order is placed through the OrderEntryClerk.

To this point, all of the actions that have been described reflect only as reads of the database tables. Placing the order is the activity that involves writing to the tables represented by the objects. Because this is a complex task and represents a business transaction, OrderPlacement is represented as a separate object. The OrderPlacement object also acts on several different objects. It creates a new order object. Because it receives a collection of order details as a parameter, it merely passes them on to the order for them to be created. The OrderPlacement object also retrieves the next order number from the district, updates the
customer balances, and reduces the stock quantity. All of these updates are completed as a single unit of activity and therefore, are encapsulated in a transaction, as shown later in this chapter.

The objects described in this section serve one of two purposes. They either represent data that is maintained in a table, or they represent actions on the data or business tasks. These two types of objects correspond to entity and session beans, respectively.

6.2 Business Data - Entity Enterprise JavaBeans

In this section, we discuss the entity enterprise beans we develop for our example application and highlight some of the key topics we address in writing them.

Entity beans are persistent objects that represent data stored in some persistent fashion, such as a database or an existing legacy application. When you modify the variables for the bean, you modify the data in the database. In our case, the persistent store is the relational database on the AS/400 system.

Our analysis identified the following five entity beans:

- Stock
- Item
- Customer
- Order
- District

6.2.1 Database Access - Using a Connection Pool

In our example application, we use connection pools to optimize performance. Connection pools are not part of the EJB specification. However, they are a common feature of application servers. The implementation described here is specific to the WebSphere Application Server Advanced Edition 3.0 environment.

Connection pools are commonly used because creating a new JDBC Connection is time consuming and uses system resources. With a pool, WebSphere Application Server Advanced Edition 3.0 creates connection objects when the server starts. When a bean needs a connection to the database, it gets one from the pool. When the bean is done with the connection, it returns it to the pool.

The way you return a WebSphere Application Server Advanced Edition 3.0 connection to the pool is to call the connection close method, for example:

```java
con.close();
```

This works because a connection is actually a class that wraps a JDBC Connection.

6.2.1.1 Using WebSphere Connection Pools

To optimize performance in the enterprise bean data access methods, the developer should not use the java.sql DriverManager getConnection(String url) method to get a JDBC connection to the database. The developer should take advantage of the WebSphere Application Server Advanced Edition 3.0 connection pools defined in the DataSource Configuration object. For information about creating DataSources, see Section 1.9 "DataSources" on page 24.

The code in Figure 189 shows an example of this technique.
Fig. 189. Using aDataSource

DataSources can be found by looking them up in the WebSphere Application Server Advanced Edition 3.0 JNDI naming space. To use a DataSource:

1. You must get an instance of the local AS/400 javax.naming.InitialContext object.
3. Use the DataSourceBean getConnection() method to get a JDBC connection out of the DataSource connection pool.
4. When you are done with it, use the java.sql.Connection close() method to put the connection back in the pool.

The name of the DataSource should be externalized, as it is not probable that you will know it during bean development. A good technique can be putting the DataSource name in an environment variable of the bean deployment descriptor. It can be changed at deployment time.

Figure 190 on page 174 shows how to set the environment variable in VisualAge for Java Enterprise Edition 3.0.
Figure 190. Setting an environment in VisualAge for Java

Figure 191 shows how to change the environment variable in WebSphere Application Server Advanced Edition 3.0.

Figure 191. Changing deployment properties in WebSphere

Figure 192 shows the code to get the environment variable value from the entity bean context.

```java
......
DataSource ds = (DataSource) initCtx.lookup("jdbc/" +
    getEntityContext().getEnvironment().getProperty("dataSourceName");
```

Figure 192. Code to get an environment variable
6.2.2 Persistence - Container or Bean Managed

After you have identified your entity beans, you must decide whether to use container managed persistence (CMP) or bean managed persistence (BMP). Use CMP, unless you want to do something it does not support, because CMP is much simpler to implement.

Common reasons for using BMP are:

- Performance
- Complex entity/database mapping

In our example, we use BMP in the Order Bean to achieve maximum performance. The underlying data is stored in two tables: order (ORDERS) and order line (ORDLIN). It may also be used many times during the placement of an order. For a complete description of the applications tables, see Section 4.1.6 "Database Table Structure" on page 88.

Another reason for rejecting CMP is that the bean data is stored in a legacy application rather than a database.

In our example, we also use BMP for the Customer Bean because the Customer Bean contains an Address object and the Address object contains fields which need persisting.

---

**Note**

As a bean developer, you must choose which form of persistence to use. It is not possible to write an entity bean that can be switched from BMP to CMP at deployment time. Changing from one form of persistence to the other requires changes in your source code for the bean.

6.2.3 Container Managed Persistence

In our example, the Stock, Item and District beans use container managed persistence (CMP). This means that we rely on WebSphere Application Server Advanced Edition 3.0 to update the Stock, Item, and District database tables any time we change the data in our entity bean. In section 5.1 "Generating a CMP Entity Bean from an existing Database File" on page 93, we use the Stock bean to describe how to implement CMP.

6.2.4 Bean Managed Persistence

When you create an entity bean that uses bean managed persistence, you take responsibility for the code that inserts, selects, updates and deletes data from the underlying database or other persistent storage mechanism by implementing the following four life cycle methods:

-.ejbCreate
-.ejbLoad
-.ejbStore
-.ejbRemove

In the simplest case where a bean instance represents one row in a database table, the responsibilities of the methods are summarized in the following list:
• **ejbCreate** — Uses the client supplied arguments to perform the following tasks:
  – Insert a new record in the database.
  – Set the instance variables for the bean.
  – Return an instance of the primary key for the bean.

• **ejbLoad** — Uses the primary key associated with the instance to perform the following tasks:
  – Retrieve the record from the database.
  – Update the bean instance variables from the database.

• **ejbStore** — Updates the database record using the values in the instance variables of the bean.

• **ejbRemove** — Deletes the database record.

The container is responsible for calling these methods at the appropriate time to maintain the synchronization between the bean and the underlying database.

In addition to the life cycle methods, you must implement the **ejbFindByPrimaryKey** method and any other finder methods that you want to define. The container calls this method when the client calls **findByPrimaryKey** on the home interface.

When you use bean managed persistence, you are not limited to storing a bean as a single row in a table. For example, the Order bean in our example application gets its data from two tables and several rows. You are not even limited to storing your data in a database. Regardless of the how the data is stored, the basic function of the four methods remains the same.

---

**6.3 Business Processes - Session Enterprise JavaBeans**

In this section, we discuss the session enterprise beans that we develop for our example application and highlight some of the key topics addressed in writing them.

Session beans are non-persistent objects that run on the server and implement business logic.

Our example uses two session beans:

• OrderEntryClerk
• OrderPlacement

**6.3.1 Three-Tier Versus Two-Tier Architecture**

Session beans allow you to easily implement a three-tier architecture. The traditional client/server architecture is a two-tiered architecture in which there are many clients and a single database. The clients often implement the business logic as well as the user interface. In a three-tiered architecture, business logic is moved out of the many clients and into a session bean running in an EJB server. The clients talk to the session bean and the bean talks to the databases.

This architecture makes the client code smaller and easier to maintain. You can change the business logic without redistributing any client code.
More importantly, this architecture lets you share business logic across different types of clients. If you implement your business logic in a bean, you can use that same bean in a Java application, an applet, a servlet or a non-Java application.

6.3.2 Stateless or Stateful

A stateless session bean allows you to write code which runs on the server and can be reused by many clients. However, the stateless session bean does not remember anything about the client between method invocations. In fact, if your client calls two methods of a stateless session bean, there is no guarantee that both methods are called on the same object in the server.

Conversely, a stateful session bean remembers information between method calls. To use a trivial example, a stateful session bean can have two methods: setName and getName. A client can pass a name to the bean using setName and retrieve that name in another call using getName.

6.3.3 Order Entry Example

In our example application, the OrderPlacement bean is a stateless session bean and the OrderEntryClerk bean is a stateful session bean. The OrderPlacement bean encapsulates the business logic to place an order. The OrderEntryClerk encapsulates the ordering process. The OrderEntryClerk bean is used by GUI programs such as Java applications or applets. The OrderPlacement bean is used by the OrderEntryClerk bean to actually place the order.

6.3.4 Stateless Session Bean - OrderPlacement

The OrderPlacement bean encapsulates the business logic to place an order in a method named placeOrder().

The source code for the OrderPlacement bean consists of the following four components:

- **Deployment Descriptor** — A serialized file that contains attribute and environment settings that define how the container invokes enterprise bean functions.
- **OrderPlacementHome** — The home interface for the bean
- **OrderPlacement** — The remote interface for the bean
- **OrderPlacementBean** — The implementation of the bean

6.3.4.1 Deployment Descriptor

Every enterprise bean must have a deployment descriptor that contains settings for the bean attributes. There are two different types of attributes, bean-level and method-level. A bean-level attribute is an attribute that applies to the entire bean, and it has a single value for the enterprise bean. A method-level attribute can be set on a single bean method. If a method-level value is specified, that will be used. If there is no method-level value, the bean-level value is used.

To see the deployment descriptor for the OrderPlacement bean, select the OrderPlacement bean in the Enterprise beans pane, click the right mouse button and choose Properties.

Figure 193 on page 178 shows the settings used in our example beans.
Figure 193. OrderPlacementBean properties

We define the OrderPlacement bean as a stateless session bean using the stateManagement attribute value in the deployment descriptor as shown in Figure 193. This tells our container that it does not need to assign one instance of the OrderPlacement bean to each client. Instead, it can create a pool. For example, there may be one hundred clients connected to the server, but only five call placeOrder at any one time. In this case, the EJB container only creates five instances of the OrderPlacement bean.

The Environment tab of the Properties panel, shown in Figure 194, allows you to enter information specific to the bean. This is useful to externalize values which depend on the specific deployment environment. The placeOrder method of the OrderPlacementBean class uses a data queue to pass order information to other applications. We use the deployment descriptor to externalize the name of the data queue.

Figure 194. OrderPlacementBean properties
6.3.4.2 Home Interface (OrderPlacementHome)
As shown in Figure 195, the home interface for a session bean does not have any finder methods because a session bean does not represent an entity in a persistent datastore that you can locate.

![Figure 195. OrderPlacement home interface](image)

6.3.4.3 Remote Interface (OrderPlacement)
We define the business methods of the bean in the remote interface. In the case of the OrderPlacement bean, there is only the placeOrder method. As shown in Figure 197 on page 181, there are two versions of the placeOrder method available, depending upon whether the client defines district as a String or an int. Internally, the String is converted to an int.
6.3.4.4 Bean Implementation (OrderPlacementBean)

We use the OrderPlacementBean class, shown in Figure 197 on page 181, to implement the EJB SessionBean interface, the create method from the home interface, and the business logic.
Figure 197. OrderPlacementBean

The SessionBean interface, together with the create method from the home interface, defines the bean instance life-cycle. As you can see in Figure 198 on page 181, very little goes on in these methods. In fact, the only work done is in the setSessionContext method.

```java
// private variables
private transient SessionContext ctx;
private transient Properties props;

// Implementation of methods required by the SessionBean interface.
public void ejbActivate() {
}

public void ejbRemove() {
}

public void ejbPassivate() {

public void setSessionContext(SessionContext ctx) {
    this.ctx = ctx;
    props = ctx.getEnvironment();
}

// Implementation of create methods defined in home interface OrderPlacementHome
public void ejbCreate() {
}
```

Figure 198. Session Bean Life-Cycle Methods
The business logic is implemented in the placeOrder method. As shown in Figure 199 on page 183, placeOrder uses the District, Stock, Order and Customer entity beans to perform much of its processing. It also uses the private method, writeDataQueue, to write the order to the data queue. We use an AS/400 batch program to read the data queue entries and print order information.
public float placeOrder(String wID, int dID, String cID, Vector orderLines) throws RemoteException{
    float oID = 0;

    try{
        // The InitialContext will let us retrieve references to the entity beans we need.
        InitialContext initCtx = new InitialContext();

        // Get the Order Number from the District entity bean.
        DistrictHome dHome = (DistrictHome) initCtx.lookup("District");
        DistrictKey districtID = new DistrictKey(dID, wID);
        District district = (District)dHome.findByPrimaryKey(districtID);
        int oIDInt = district.getNextOrderId(true); // 'true' tells the District to increment the order id.
        oID = oIDInt;

        // Update the Stock level for each item in an order line using the Stock entity bean
        StockHome sHome = (StockHome) initCtx.lookup("Stock");
        Enumeration lines = orderLines.elements();
        float orderTotal = 0;
        while(lines.hasMoreElements()){
            OrderDetail od = (OrderDetail)lines.nextElement();
            String itemID = od.getItemID();
            int itemQty = od.getItemQty();
            orderTotal += od.getItemAmount(); // Calculate the order total while
            // we are going through the orders.
            StockKey stockID = new StockKey(wID, itemID);
            Stock stock = (Stock)sHome.findByPrimaryKey(stockID);
            stock.decreaseStockQuantity(itemQty);
        }

        // Save the Order to the database by creating an Order entity bean
        OrderHome oHome = (OrderHome) initCtx.lookup("Order");
        oHome.create(wID, dID, cID, oID, orderLines);

        // Update the Customer records using the Customer entity bean
        CustomerHome cHome = (CustomerHome) initCtx.lookup("Customer");
        CustomerKey customerID = new CustomerKey(cID);
        Customer customer = (Customer)cHome.findByPrimaryKey(customerID);
        customer.updateBalance(orderTotal);

        // Write the order to the data queue.
        try{
            writeDataQueue(wID, dID, cID, oID);
        } catch(Exception e){
            System.out.println("WriteDataQueue error: " + e.getMessage());
            throw new RemoteException(e.getMessage());
        }
    } catch (Exception e) {
        throw new RemoteException(e.getMessage());
    }
    return oID;
}

Figure 199. The placeOrder Method Processing
6.3.5  Stateful Session Bean - OrderEntryClerk

The OrderEntryClerk encapsulates the ordering process. We define the OrderEntryClerk bean as a stateful session bean to use it as an order clerk. This bean keeps a record of who the customer is and what items they want to order.

The source code for the OrderClerk bean consists of the following four components:

- **Deployment Descriptor** — A serialized file that contains attribute and environment settings that define how the container invokes enterprise bean functions.
- **OrderEntryClerkHome** — The interface for the bean
- **OrderEntryClerk** — The remote interface for the bean
- **OrderEntryClerkBean** — The implementation of the bean

6.3.5.1  Deployment Descriptor

To see the deployment descriptor for the OrderEntryClerk bean, select the OrderEntryClerk bean in the Enterprise beans pane, click the right mouse button and choose Properties.

Figure 200 shows the settings used in our bean.

![Properties](image)

*Figure 200. OrderEntryClerkBean properties*

We define the OrderPlacement bean as a stateful session bean using the stateManagement attribute value in the deployment descriptor as shown in Figure 200. This tells our container that it needs to assign an instance to each client and that it needs to store the non-transient variables of the bean across method calls.

The Environment tab of the Properties panel, shown in Figure 201 on page 185, allows you to enter information specific to the bean. This is useful to externalize values which depend on the specific deployment environment. We use the deployment descriptor to externalize the name of the DataSource.
6.3.5.2 OrderEntryClerkBean

We use the OrderEntryClerkBean class, shown in Figure 202, to implement the EJB SessionBean interface, the create method from the home interface, and the business logic.

The SessionBean interface, together with the create method from the home interface, defines the bean instance life-cycle.

6.3.5.3 Implementing a State

As a bean developer, use the following steps to implement a state:

1. Define non-transient variables in your bean.
2. Initialize these variables in the ejbCreate methods.
3. Use these variables in the business logic.

The bean state consists of any variables we define that are not transient. As shown in Figure 203, our OrderEntryClerk bean has two variables that maintain the beans state.

```
// Following is the state in this stateful session bean
public String custID;
public Vector items;
```

Figure 203. Defining State Variables

As shown in Figure 204, we initialize these variables in the `ejbCreate` method. In our example, `ejbCreate` does not take any arguments. If we write the example differently, where the client passes the customer ID in as an argument when creating the bean, then we set `custID` to the provided argument.

```
public void ejbCreate() throws javax.ejb.CreateException, java.rmi.RemoteException {
  try {
    InitialContext initCtx = new InitialContext();
    ds = (DataSource) initCtx.lookup("jdbc/" +
        mySessionCtx.getEnvironment().getProperty("dataSourceName"));
  }
  catch (Exception e) {
    throw new RemoteException("Error in ejbCreate: " + e.getMessage());
  }
  items = new Vector();
  custID = null;
  isDirty = true;
}
```

Figure 204. Initializing Values

We use these variables in the business methods. The three methods related to the bean state are: `setCustomer` (Figure 205 on page 186), `addOrderLine` (Figure 206 on page 187) and `placeOrder` (Figure 207 on page 187).

In the `setCustomer` method, we set the value of `custID`.

```
public void setCustomer(String cid) throws RemoteException {
  if ( verifyCustomer(cid) ) {
    custID = cid;
  } else {
    throw new CpwejbException("Customer id " + cid + " not valid");
  }
}
```

Figure 205. The setCustomer Method

In the `addOrderLine` method, we add another OrderDetail object to the items Vector. These are the items that we want to order.
public void addOrderLine(String iid, int quantity) throws RemoteException {
    if (custID == null) {
        throw new RemoteException("OrderEntryClerkBean: Customer ID must be set first");
    }
    Item item = null;
    try {
        InitialContext initCtx = new InitialContext();
        Object homeObject = initCtx.lookup("Item");
        ItemHome home = (ItemHome)
                javax.rmi.PortableRemoteObject.narrow((org.omg.CORBA.Object) homeObject,
                        com.ibm.itso.roch.wasaejb.ItemHome.class);
        ItemKey itemK = new ItemKey(iid);
        item = home.findByPrimaryKey(itemK);
    } catch (Exception e) {
        throw new RemoteException(e.getMessage());
    }
    OrderDetail ol = new OrderDetail(iid, item.getItemPrice() * quantity, quantity);
    items.addElement(ol);
}

Figure 206. The addOrderLine Method

The two variables are used in the placeOrder method. The placeOrder method passes the custID and items variables as arguments to an instance of the OrderPlacement bean.

public String placeOrder() throws RemoteException {
    String orderNumber = null;
    if (custID == null) {
        throw new CpwejbException("OrderEntryClerkBean: Hey buddy!
                                    Customer ID must be set first");
    }
    if (items.size() == 0) {
        throw new CpwejbException("OrderEntryClerkBean:
                                    What are you thinking! No order lines!");
    }
    try {
        InitialContext initCtx = new InitialContext();
        OrderPlacementHome home = (OrderPlacementHome) initCtx.lookup("OrderPlacement");
        OrderPlacement placement = home.create();
        float number = placement.placeOrder(whid, did, custID, items);
        orderNumber = Float.toString(number);
        orderNumber = orderNumber.substring(0, orderNumber.length() - 2);
        // Clear out the state of the session bean at this point
        items = new Vector();
        custID = null;
    } catch (Exception e) {
        throw new RemoteException(e.getMessage());
    }
    return orderNumber;
}

Figure 207. The placeOrder Method

In the following list, the clerk bean also implements several methods that pertain to activities that a clerk performs, but does not use its state:

1. The findAllCustomers method retrieves a list of customers from the server.
2. The findAllItems method retrieves a list of items from the server.
3. The `findRangeOfItems` method retrieves a subset of items from the server.
4. The `verifyCustomer` method verifies that a customer number is valid.
5. The `getCustomerForOrder` method retrieves the customer number for an order.
6. The `getItemsForOrder` method retrieves the order lines for an order.

These methods are used by client applications. See Chapter 7, “Building Java Applications with Enterprise JavaBeans” on page 191 for details about how we use these methods.

### 6.4 Transactions

A major benefit to implementing business logic as enterprise beans is that the EJB container can automatically provide transaction support. Essentially, transaction support means that you can group related database actions into a single transaction such that all the actions are performed or none are performed.

You can see an example of this in the `placeOrder` method of our `OrderPlacement` bean (Figure 199 on page 183). Placing an order involves updating a number of tables within our database: District, Stock, Order, Order Line and Customer. Because we perform these updates from within a method of a bean, we automatically get transaction support. If an exception is thrown at any point within the `placeOrder` method, our container rolls back all the database transactions that occurred up to that point.

To get transaction support, we set the `isolationLevel` and `transactionAttribute` values in our deployment descriptors. To see the deployment descriptor for the `OrderPlacement` bean, select the `OrderPlacement` bean in the Enterprise beans pane, click the right mouse button and choose Properties.

Figure 208 on page 188 shows the settings used in our example beans.
6.4.1 Isolation Level

The isolation level corresponds to JDBC isolation levels. The value is typically passed on to the underlying database. The safest isolation method to use is TRANSACTION_SERIALIZABLE. This avoids dirty reads, phantom reads, and unrepeatable reads.

6.4.2 Transaction Attribute

The transaction attribute tells the EJB server whether to create a new transaction when running a bean method. The TX_REQUIRED value tells the EJB server to create a transaction if the client did not already create one. If the client creates a transaction, then the bean uses that transaction.

In our OrderPlacement example, the client is a program that calls the placeOrder method (Figure 207 on page 187) of the OrderEntryClerk bean. The placeOrder method in OrderEntryClerk bean calls the placeOrder method of the OrderPlacement bean (Figure 199 on page 183). The OrderPlacement bean then calls methods on the District, Stock, Order and Customer entity beans.

Our EJB server starts a new transaction when the client calls placeOrder in OrderEntryClerk bean because the OrderEntryClerk bean has TX_REQUIRED set. The OrderEntryClerk bean then calls the OrderPlacement.placeOrder method inside of this transaction. This transaction context is then passed on to each of the entity beans.

The result is that all of the database activity triggered when the client calls placeOrder in OrderEntryClerk bean gets treated as one transaction: either everything happens or nothing happens.

The EJB specification defines five other values for the transaction attribute. Their effect is summarized in the following list:

- **TX_NOT_SUPPORTED** — The bean is invoked outside of any transaction scope. For example, if we define the Stock bean with this attribute, our EJB server does not include updates performed by the Stock bean in our transaction. If the transaction fails, the changes to the Stock table still occur.
- **TX_BEAN_MANAGED** — This allows beans to start their own transactions using the javax.jts.UserTransaction interface. Use this if you want to create a transaction smaller than an entire method. The drawback of this attribute is that EJB server does not create a new transaction if the client does not specify one.
- **TX_SUPPORTS** — If the client provides a transaction, then the EJB server uses it. If the client does not provide one, the server does not supply one.
- **TXQUIRES_NEW** — This tells the server to always create a new transaction regardless of whether the client supplies one or not.
- **TX_MANDATORY** — This tells the server to throw an exception if the client does not provide a transaction.

6.5 Conclusion

In this chapter, we covered the architecture and design of the Enterprise JavaBeans that we use in this redbook.
Our analysis identified the following five entity beans:

- Stock
- Item
- Customer
- Order
- District

We also designed the following two session beans:

- OrderEntryClerk
- OrderPlacement

We covered the following design decisions:

- Whether to use container managed or bean managed persistence for the entity beans
- Whether to use stateful or stateless session beans
- How to handle transactions

We covered key code snippets and control file settings that we used in implementing our beans. The entire source of all the beans is available for download from our Web site.

Now that our Enterprise JavaBeans are designed and written, we are ready to use them in applications. These applications are demonstrated in Chapter 7, “Building Java Applications with Enterprise JavaBeans” on page 191. This chapter demonstrates the power of using Enterprise JavaBeans. After they are written and deployed on a server, such as the AS/400 system, client programmers can use them with very little knowledge of how the beans actually work. The client programmer only has to know what methods the Enterprise JavaBeans support and how to call them. All the difficult work is done by the bean writer or the Java server. Another key advantage is that whether you are writing a Java application, a Java applet, a Java servlet or even a Visual Basic program, it always works the same way. You only need to call the methods provided by the Enterprise JavaBeans to handle the application processing.
Chapter 7. Building Java Applications with Enterprise JavaBeans

This chapter discusses building AS/400 Java applications using Enterprise JavaBeans. We use the Enterprise JavaBeans described in Chapter 5, “Developing Enterprise JavaBeans” on page 93 to provide access to AS/400 resources. The following types of application development scenarios are discussed in this chapter:

- Java applications
- Java servlets

7.1 Building Java Applications with VisualAge for Java

In this section, we use VisualAge for Java to build a Java application similar to the RPG order entry application discussed in Section 4.1, “Overview of the Order Entry Application” on page 79. To make it easy to reuse the classes that we build for our application, we separate the access to the AS/400 system from the graphical user interface. We create a package named Support, which contains a number of support classes.

Figure 209 shows the classes contained in the Support package. The ItemsDb class is responsible for all access to the AS/400 system. It uses the Enterprise JavaBeans described in Chapter 5, “Developing Enterprise JavaBeans” on page 93 to access the AS/400 system.

Figure 209. Support package

The following list shows the other Support package classes that are supporting classes for applications:

- **Customer** — An object oriented representation of a Customer table row
- **Item** — An object-oriented representation of an Item table row
- **Order** — An object-oriented representation of an Order table row
- **OrderDetail** — An object-oriented representation of an OrderLine table row
7.1.1 The ItemsDb Class

In this section, we investigate the ItemsDb class. We look at how it accesses the AS/400 resources. The following key methods are provided:

- The `getInitialContext` method creates an InitialContext object.
- The `connect` method establishes a connection to the Java server.
- The `getAllCustomers` method retrieves a list of customers from the server.
- The `getItems` method retrieves a list of items from the server.
- The `findRangeOfItems` method retrieves a subset of items from the server.
- The `verifyCustomer` method verifies that a customer number is valid.
- The `confirmOrder` method places and confirms an order.
- The `submitOrder` method places and confirms an order using a shopping cart.
- The `connectStateless` method establishes a stateless connection to the Java server for servlets.
- The `submitOrderStateless` method places an order given a customer ID and a shopping cart.

All the methods in the ItemsDb class, except connectStateless and submitOrderStateless, use the OrderEntryClerk session Enterprise JavaBean to access AS/400 resources. The "stateless" methods use the OrderPlacement stateless session EJB.

Our intention is to create an access class that is capable of running on its own. That is, you can use it even without any user interfaces. As shown in Figure 210 on page 192, the advantage of this approach is that you can use the access class in any context. For example, you can use a graphical user interface (GUI) on top of the access class when you want to create a stand-alone Java application. You also use exactly the same class when you want to develop a Java servlet. In this case, replace the GUI front end with a layer that is capable of running as a servlet. You can also implement a distributed application where some parts of your application run on a server. Our implementation using Enterprise JavaBeans is an example of this. When we call the methods of the Enterprise JavaBeans, we are actually running code on the server.

Separating the layers properly and using a clean and well-defined interface between the layers is important if you want to take advantage of object-orientation.

![Diagram](image-url)
To use Enterprise JavaBeans in an application, you must be able to access them. The way a client program accesses enterprise beans is defined by two interfaces:

- Home interface
- Remote (EJBObject) interface

The home interface contains methods that describe how you can instantiate an enterprise bean object. The remote interface, by comparison, defines the methods of an enterprise bean that can be accessed by your user program. To access an enterprise bean, your user program goes through the following steps:

1. Obtains a context to the name server (name service context).
2. Looks up the home of the enterprise bean using the name service context.
3. Creates an enterprise bean instance from the enterprise bean home, which returns an enterprise bean proxy object.
4. Accesses the remote methods of the enterprise bean instance through an enterprise bean proxy object. Each call to the enterprise bean proxy object is a remote call that can throw an exception, such as java.rmi.RemoteException.

### 7.1.1.1 Import Statements

Figure 211 shows the class description for the ItemsDb class. We must import classes from the javax.naming, javax.ejb and java.rmi packages to use Enterprise JavaBeans. We also import the interfaces for the Enterprise JavaBeans we use. They are found in the `com.ibm.itso.roch.wasejb` package.

```java
import java.math.*;
import java.util.*;
import com.ibm.itso.roch.wasejb.*;
import javax.ejb.*;
import javax.naming.*;
import java.rmi.RemoteException;
import java.rmi.Remote;
import java.util.*;
import java.io.*;
public class ItemsDb extends java.lang.Object {
```

*Figure 211. The ItemsDb Class Description*

### 7.1.1.2 The getInitialContext Method

To use the Java server and the Enterprise JavaBeans, we need to access the Java server’s Java Naming and Directory Interfaces (JNDI) naming service. This is done by instantiating an InitialContext object. We use the getInitialContext method to create the InitialContext object. Here is the code:
Figure 212. The getInitialContext Method

The InitialContext object is actually the naming service. When it is instantiated, it uses the name of the service provider and the URL where it is located. In this example, we use the IBM WebSphere Advanced Edition 3.0 JNDI naming service. We pass in the name of the server and the TCP/IP port number in the URL. Optionally, you can use a user ID and password as well. It is important to notice that this is a user ID and password for the naming service provider, not for the system. This is not the same as the AS/400 user profile security.

7.1.1.3 The connect Method

The connect method is used to connect to the Java server. We show the connect method in Figure 213.

```java
public static Context getInitialContext() throws Exception {
    Properties p = new Properties();
    try {
        p.put(Context.INITIAL_CONTEXT_FACTORY, 
            "com.ibm.ejs.ns.jndi.CNInitialContextFactory");
        p.put(Context.PROVIDER_URL, "iiop://" + getSystemName() + ":" + getPort());
        p.put(Context.SECURITY_PRINCIPAL, getUserId());
        p.put(Context.SECURITY_CREDENTIALS, getPassword());
        InitialContext cx = new InitialContext(p);
        return cx;
    } catch (Exception e) {
        System.out.println("error creating Context");
        throw (e);
    }
}
```

```java
public String connect() throws Exception {
    try{
        Context ctx = getInitialContext();
        java.lang.Object tempObject= ctx.lookup("OrderEntryClerk");
        OrderEntryClerkHome home = (OrderEntryClerkHome) javax.rmi.PortableRemoteObject.narrow(
            (org.omg.CORBA.Object) tempObject, com.ibm.itso.roch.wasaejb.OrderEntryClerkHome.class);
        clerk = (OrderEntryClerk) home.create();
    } catch (FinderException fe) {
        System.out.println ("Could not find order clerk " + fe.getMessage());
        return ("Could not find order clerk " + fe.getMessage());
    } catch (Throwable t) {
        System.out.println ("Could not connect " + t.getMessage());
        return ("Could not connect " + t.getMessage());
    }
    System.out.println("Connected to " + getSystemName() + ":" + getPort());
    return "Connected to " + getSystemName() + ":" + getPort();
}
```

Figure 213. The connect Method

We call the getInitialContext method to create an InitialContext object. After the InitialContext object is created, the home interface for the bean must be located.
We accomplished this through a method call to the lookup method of the InitialContext object. The home interface for the bean is located by name. The application programmer only needs to know the name of the interface and the class type that is returned. Because the InitialContext object returns any type that is registered with it, the application programmer must cast to the correct type. We use the static method narrow() to cast the java.lang.Object returned by the lookup method to the correct home interface type. The narrow() method takes the object to be narrowed and the class of the EJB home object to be returned as parameters. We use the home interface to create an instance of the OrderEntryClerk EJB which we name clerk. The clerk object knows how to do many things associated with the order entry system. For example, the clerk can perform the following tasks:

- Return a list of items from the Item table
- Return a list of customers from the Customer table
- Validate a customer number
- Create an order on the AS/400 system

We use the OrderEntryClerk object named clerk to handle the application processing.

### 7.1.1.4 The getAllCustomers Method

The getAllCustomers method retrieves all the rows from the AS/400 Customer table. Figure 214 shows the code.

```java
public java.util.Vector getAllCustomers() {  
    java.util.Vector tempCust = new java.util.Vector();  
    try {  
        tempCust = clerk.findAllCustomers();  
    } catch (Exception e) {  
        System.out.println("::::::::::::::: Unexpected Error :::::::::::::::");  
        e.printStackTrace();  
    }  
    return tempCust;  
}
```

**Figure 214. The getAllCustomers Method**

To retrieve all customers, we call the OrderEntryClerk’s findAllCustomers() method. It returns a Vector containing information for all customers. The Vector contains a String array element for each customer in the Customer table. The String array contains entries which correspond to the fields in the rows of the Customer table. We return the Vector to the caller of this method.

### 7.1.1.5 The getItems Method

The getItems method retrieves all rows from the AS/400 Item table. Figure 215 shows the code.
To retrieve all items, we call the OrderEntryClerk's `getItems()` method. It returns a Vector containing all items. The Vector contains a String array element for each item in the Item table. The String array contains entries which correspond to the fields in the rows of the Item table. We return the Vector to the caller of this method.

### 7.1.1.6 The `findRangeOfItems` Method

The `findRangeOfItems` method retrieves a subset of rows from the Items table. Figure 216 shows the code.

```java
public java.util.Vector findRangeOfItems(String itemnoMin, String itemnoMax) {
    try {
        ejbItems = clerk.findRangeOfItems(itemnoMin, itemnoMax);
    } catch (Exception e) {
        System.out.println("::::::::::::::: Unexpected Error :::::::::::::::");
        e.printStackTrace();
    }
    System.out.println("all items returned");
    return ejbItems;
}
```

To retrieve a subset of items, we call the OrderEntryClerk's `findRangeOfItems` method. We pass it parameters containing the value of the first item and the value of the last item to return. It returns a Vector containing all items within the range. We return the Vector to the caller of this method.

### 7.1.1.7 The `verifyCustomer` Method

The `verifyCustomer` method verifies that a customer number contains a valid row in the Customer table. Figure 217 shows the code.
Chapter 7. Building Java Applications with Enterprise JavaBeans

### 7.1.1.8 The submitOrder method

When we are ready to submit an order for processing on the AS/400 system, we use the `ItemsDb submitOrder` method. It, in turn, uses methods provided by the `OrderEntryClerk` bean. Figure 218 shows the code.

```java
public String submitOrder(Order anOrder) throws Exception {
    try {
        OrderDetail orderLine = null;
        // set the customer number
        clerk.setCustomer(anOrder.getCustomer().getId().toString());
        BigDecimal orderLineCount = new BigDecimal(anOrder.getOrderDetails().size());
        for (java.util.Enumeration itemE = anOrder.getOrderDetails().elements(); itemE.hasMoreElements();) {
            orderLine = (OrderDetail) itemE.nextElement();
            clerk.addOrderLine(orderLine.getItem().getId(), orderLine.getQuantity());
        }
        // place the order
        String orderNumber = clerk.placeOrder();
        return orderNumber;
    } catch (Exception e) {
        throw (e);
    }
}
```

**Figure 217. The verifyCustomer Method**

To verify that a customer exists in the Customer table, we call the `OrderEntryClerk`'s `verifyCustomer` method. We pass it a parameter containing the customer number. It returns a boolean which is true or false, depending on whether the customer is valid. We return the boolean to the caller of this method.

```java
public boolean verifyCustomer(String customerId) {
    boolean isValid = false;
    System.out.println("isValid = " + isValid);
    try {
        isValid = clerk.verifyCustomer(customerId);
        System.out.println("isValid = " + isValid);
        if (isValid)
            validCustomerId = customerId;
    }
    catch (Exception e) {
        return isValid;
    }
    return isValid;
}
```

**Figure 218. The submitOrder Method**
We use the following steps to submit an order:
1. Set the name of the customer using the `setCustomer` method.
2. Pass in as parameters, the item number and the quantity for each item we want to order using the `addOrderLine` method of the OrderEntryClerk.
3. Place the order using the `placeOrder` method of the OrderEntryClerk. It returns the order number of the order that is created.
4. Return the order number to the caller of this method.

The `submitOrder` methods demonstrates one of the key advantages of using Enterprise JavaBean technology. You only need to know what the bean does and how to interface to it. Do not be concerned with the details of how it accomplishes its task. If you want to implement the order processing logic in your application without using Enterprise JavaBeans, the application code is much more complex because you must provide the code which does the database access and order processing.

### 7.1.1.9 The “stateless” methods

In order to properly support servlet clients, we provide two “stateless” methods in the `ItemsDb` class that don’t use the `OrderEntryClerk` session bean at all, `connectStateless` and `submitOrderStateless`. The only constraint to using these methods is that the `connectStateless` method must be called before the `submitOrderStateless` method. These methods are similar to the `connect` and `submitOrder` methods, but use the `OrderPlacement` session bean (stateless) instead of the `OrderEntryClerk` session bean.

The `connectStateless` method provides a connection to the `OrderPlacement` session bean.

```java
public String connectStateless() throws Exception {
    try {
        Context ctx = getInitialContext();
        java.lang.Object tempObject= ctx.lookup("OrderPlacement");
        OrderPlacer = (OrderPlacement) home.create();
    } catch (FinderException fe) {
        System.out.println("Could not find order placer " + fe.getMessage());
        return ("Could not find order placer " + fe.getMessage());
    } catch (Throwable t){
        t.printStackTrace();
        return ("Could not connect " + t.getMessage());
    }
    return "Connected to "+ getSystemName() + ":" + getPort();
}
```

Figure 219. The `connectStateless` method

The `submitOrderStateless` method, shown in Figure 220, is used by a client application to place an order. It calls the `placeOrder` method of the `OrderPlacement` session bean. The input parameters for this method are a String
containing the customer ID and a Vector containing the items to order. It returns
the order number of the new order.

```java
public int submitOrderStateless(String CustID, Vector OrderLines) throws Exception {
    int returnvalue;
    try {
        returnvalue = (int)OrderPlacer.placeOrder("0001", "1", CustID, OrderLines);
    } catch (Exception e) {
        throw (e);
    }
    return returnvalue;
}
```

Figure 220. The submitOrderStateless method

### 7.1.2 Using the ItemsDb Class

Now we put it all together. First, we run our ItemsDb class just using scripts in the
VisualAge for Java Scrapbook. We create a new instance of the ItemsDb class
and call its connect method. The ItemsDb class also provides methods to set
values for the user ID, password, TCP/IP port, and system name.

```java
Support.ItemsDb aItemsDb = new Support.ItemsDb();
aItemsDb.setUserId("cpwejb");
aItemsDb.setPassword("cpwejb400");
aItemsDb.setPort("900");
aItemsDb.setSystemName("localhost");
aItemsDb.connect();
```

Figure 221. Testing the ItemsDb Class in the VisualAge for Java Scrapbook

To run the code (Figure 221) in the VisualAge for Java Scrapbook, we select and
highlight it, click the right mouse button, and select run. To help test the code, we
write messages to the Java console using the System.out.println method. In this
example, we try to connect to the Java server running on the local system and
listening on port 900. If the server is active and the security information is valid,
we see the message Figure 222 on page 199 on the Java console.

```
Connected to localhost:900
```

Figure 222. Java Console Messages

If this is successful, we try to retrieve information from the Customer table on the
server. To accomplish this, we use the getAllCustomers method from the ItemsDb
class. Figure 223 shows the code.
If this is successful, we retrieve the rows from the Customer table and display the first two fields for each row on the Java console. The getAllCustomers method returns a Vector. The Vector contains a String array element for each customer in the Customer table. The String array contains entries which correspond to the fields in the rows of the Customer table. For each String array element, we print the first two array elements. Figure 224 shows the Java console output.

In this section, we tested the ItemsDb class without using a graphical user interface. In the next section, we build an order entry application. We use the ItemsDb class to access the AS/400 system. Later, we build a servlet. Because the ItemDb class is independent of the user interface, we also use it for the servlet.

### 7.2 Building the Order Entry Application

In this section, we show the Java order entry application. It is an Enterprise JavaBeans version of the RPG order entry application discussed in Chapter 4, “Overview of the Order Entry Application” on page 79. Figure 225 shows the main order entry window that is displayed when the application is started.
Figure 225. Parts Order Entry - Initial Window

This window is built using VisualAge for Java. The name of the class that defines this window is OrderEntryWdw. It is the controlling class (or entry point) of the client application.

The following list contains the primary components contained in the window:

- A menu bar that contains a Connect menu item
- Six text fields for customer information
- Three text fields for the current item that is selected
- A multicolumn list box that displays all items in the current order
- A text field for status updates
- A button for listing all valid customers
- A button for listing all valid items that can be ordered
- A button for adding the current item to the order list box
- A button for submitting the order
- A button for exiting the application

When the window is first displayed, all the buttons except the Exit button are disabled. Initially, the only valid options are exiting or connecting to the remote database (the host AS/400 system).

7.2.1 Application Flow Through the Java Client Order Entry Window

The series of tasks that the client application supports is summarized in the following list:

- Connect to the remote database
- Retrieve a list of valid customers
- Select a customer
- Retrieve a list of valid items (parts)
- Select an item
- Verify the item
- Add the item to the order
• Submit the order

Each of these tasks are discussed in the following sections.

7.2.2 Connecting to the Database

We use the following steps to connect to the AS/400 database:

1. To start the application, enter the following command statement:
   java EjbApplications.OrderEntryWdw xxxx
   The xxxx is equal to the TCP/IP port number on which the Java server runs.

2. As shown in Figure 226, after the Parts Order Entry window is shown, click Connect to connect to the AS/400 database.

   ![Figure 226. The OrderEntry Connection Menu](image)

3. As shown in Figure 227, a sign-on dialog is shown. Type the system name, user ID, and password. The TCP/IP port used by the Java server is also required. The port is passed in as a parameter (see step 1) when starting the application.

   ![Figure 227. Sign On to the System](image)
After the security information is entered, click **OK**. A connection is made to the Java server. To see the action of the **OK** button, we show the connections in the VisualAge for Java Visual Composition Editor.

![Sign-On Dialog in the VisualAge for Java Visual Composition Editor]

Figure 228. The Sign-On Dialog in the VisualAge for Java Visual Composition Editor

Figure 228 shows the Sign-On dialog in the VisualAge for Java Visual Composition Editor. Notice that we place an ItemsDb bean on the free form surface. This has the effect of instantiating an ItemsDb object. We use this ItemsDb object for all of our AS/400 resource access.

Figure 229 on page 203 shows the connections of the **OK** button. The connections perform the following tasks:

1. Call the **setSystemName** method of the ItemsDb class passing in the value from the System Name text field as a parameter.
2. Call the **setUserId** method of the ItemsDb class passing in the value from the UserID text field as a parameter.
3. Call the **setPassword** method of the ItemsDb class passing in the value from the Password text field as a parameter.
4. Call the **connect** method of the ItemsDb class.

![OK Button Connections]

Figure 229. **OK** Button Connections

If the connection is successful, the Parts Order Entry window is shown with the List Customers button enabled (see Figure 230).
Figure 230. Order Entry Application with List Customers Button Enabled

7.2.3 Retrieving the Customer List

The action of the List Customers button is to instantiate an object based on the SltCustWdw class.

Figure 231 shows the SltCustWdw class in the VisualAge for Java Visual Composition Editor. The following three methods in this class are of interest to us:
• **setupCustBox** — Instantiates and formats the multicolumn list box which displays the customer records.

• **populateCustBox** — Retrieves the customer records from the Customer table.

• **custSelected** — Handles processing when a customer is selected from the list box.

The `WindowOpened` event of the SltCustWdw window is used to populate the list box. Figure 232 shows the connections tied to the `WindowOpened` event of the SltCustWdw window.

![Figure 232. WindowOpened Event Connections](image)

The purpose of the `populateCustBox` method is to retrieve the rows from the Customer table and place them in the list box. Figure 233 shows the code.

```java
private void populateCustBox(java.util.Vector customerList) {
    orderWindow.updateStatus("Retrieving customer list...");
    for (java.util.Enumeration e = customerList.elements(); e.hasMoreElements();) {
        String[] array = (String[]) e.nextElement();
        getIMulticolumnListbox1().addRow(array, array[0]);
    }
    return;
}
```

![Figure 233. The populateCustBox Method](image)

This method is very simple. It receives a Vector containing the customers as an input parameter. It reads each element from the Vector, formats it into a String array, and adds the String array to the list box. The question to ask is: **How does the Vector passed in as a parameter get populated?**

To answer this question, notice in Figure 231 on page 204 that the connection to the `populateCustBox` method is connected to the ItemDb object. It uses the ItemsDb class to set the required parameter.
Figure 234. The populateCustBox Parameter

Figure 234 shows how the parameter is set. We call the `getAllCustomers` method in the `ItemsDb` class. It returns a Vector containing the customers.

Figure 235 on page 206 shows the window displayed after the list of customers is retrieved.

Selecting a customer and clicking on the OK button sets the appropriate text fields in the main Order Entry Window. The action of the OK button is connected to the `custSelected` method. Figure 236 on page 207 shows the code.
private void custSelected() {
    int selectedIndex = getIMulticolumnListbox1().getSelectedIndex();
    if (selectedIndex < 0)
        return;
    java.util.Vector selected = new java.util.Vector();
    String[] array = new String[9];
    Object key = getIMulticolumnListbox1().getSelectedObject();
    selected.addElement(getIMulticolumnListbox1().getRowData(key));
    for (java.util.Enumeration e = selected.elements(); e.hasMoreElements();) {
        Object[] element = ((Object[]) e.nextElement());
        array[0] = element[0].toString();
        array[1] = element[1].toString();
        array[7] = element[7].toString();
        array[8] = element[8].toString();
        Customer aCustomer = new Customer(array);
        orderWindow.setSelectedCust(aCustomer);
    }
    this.dispose();
    return;
}

Figure 236. The custSelected Method

The custSelected method retrieves the row selected in the list box. It creates a Customer object based on the Customer class found in the Support package. Figure 236 shows the class description for the Customer class.

public class Customer implements java.io.Serializable {
    public String id;
    private String lastName;
    private String firstName;
    private String init;
    private String address1;
    private String address2;
    private String city;
    private String state;
    private String postCode;
}

Figure 237. The Customer Class

The Customer class is an object-oriented representation of a customer record. The constructor simply sets the data members based upon the String elements in the array that is passed in. Figure 238 on page 208 shows the constructor.
The Customer class also provides standard getter methods for retrieving the values of individual data members. These methods are basic and not discussed here.

The selected Customer object is passed as a parameter to the setSelectedCust method of the OrderEntryWdw object. Figure 239 shows the setSelectedCust method.

```java
public Customer (String[] custInfo) {
    id = custInfo[0];
    lastName = custInfo[1];
    firstName = custInfo[2];
    init = custInfo[3];
    address1 = custInfo[4];
    address2 = custInfo[5];
    city = custInfo[6];
    state = custInfo[7];
    postCode = custInfo[8];
}
```

**Figure 238. Customer Class Constructor**

```java
public void setSelectedCust(Customer selectedCust) {
    customer = selectedCust;
    order = new Order();
    order.setCustomer(selectedCust);
    getCustIDTF().setText(selectedCust.getId().toString());
    getCustNameTF().setText(selectedCust.getFullName());
    getStreetTF().setText(selectedCust.getAddress());
    getCityTF().setText(selectedCust.getCity());
    getStateTF().setText(selectedCust.getState());
    getPCodeTF().setText(selectedCust.getPostCode());
    getListItemBTN().setEnabled(true);
    updateStatus("Customer information set");
    return;
}
```

**Figure 239. The OrderEntryWdw setSelectedCust Method**

Because we are in the process of creating an order, this method creates a new Order object and sets the selected customer in the order object. The Order class is like the Customer class, an object-oriented representation of an order that we place. The Order class is also be found in the Support package. The setSelectedCust method also sets the appropriate text fields in the Order Entry window. After you have selected a customer, the current state of the main window is shown in Figure 240 on page 209.
After the information for the selected customer is set, the List Items button is enabled and we are ready to retrieve the list of valid items from the server.

### 7.2.4 Retrieving the Item List

The action of the List Items Button is to instantiate an object based on the SltItemWdw class.

Figure 241 shows the SltItemWdw class in the VisualAge for Java Visual Composition Editor. The following three methods in this class are of interest to us:
• **setupItemBox** — Instantiates and formats the multicolumn list box which displays the item records

• **populateItemBox** — Retrieves the customer records from the Item table

• **itemSelected** — Handles processing when an item is selected from the list box

We use the windowOpened event to populate the list box. Figure 242 shows the connections tied to the windowOpened event of the SltItemWdw window.

![Figure 242. WindowOpened Event Connections](image)

The purpose of the populateItemBox method is to retrieve the rows from the Item table and place them in the list box. Figure 243 shows the code.

```java
private void populateItemBox(java.util.Vector itemList) {
    orderWindow.updateStatus("Retrieving item list...");
    for (java.util.Enumeration e = itemList.elements(); e.hasMoreElements();)
        String[] array = (String[]) e.nextElement();
        getIMulticolumnListbox1().addRow(array, array[0]);
    orderWindow.updateStatus("Item list retrieved");
    return;
}
```

**Figure 243. The populateCustBox Method**

This method is very simple. It receives a Vector containing the items as an input parameter. It reads each element in the Vector, formats it into a String array and adds the String array to the list box. Again, the question to ask is: How does the Vector passed in as a parameter get populated?

It works the same as the sltCustWdw window. Notice in Figure 241 on page 209 that the populateItemBox method connection is connected to the ItemDb object. This is how the parameter for the populateItemBox is set.
Figure 244. The populateCustBox Parameter

Figure 244 shows how the parameter is set. We call the getItems method in the ItemsDb class. It returns a Vector containing the items.

Figure 245 shows the window that is displayed after the list of items is retrieved.

Selecting an item and clicking on the OK button sets the appropriate text fields in the Order Entry Window.

The action of the OK Button is connected to the itemSelected method. Figure 246 on page 212 shows the code.
The `itemSelected` method retrieves the row selected in the list box. It creates an `Item` object based on the `Item` class which is found in the Support package. Figure 247 shows the class description for the `Item` class.

```java
public class Item implements java.io.Serializable {
    private String id;
    private String name;
    private java.math.BigDecimal price;
    private int quantity;
    private String description;
}

public Item(String[] itemInf) {
    id = itemInf[0];
    name = itemInf[1];
    // remember to trim the `$` symbol before instantiating a BigDecimal
    price = new java.math.BigDecimal(itemInf[2].substring(1));
    quantity = new Integer(itemInf[3]).intValue();
    description = itemInf[4];
}
```

The `Item` class is an object-oriented representation of an item record. The constructor simply sets the data members based upon the String elements in the array that is passed in. Figure 248 shows the constructor.

```java
private void itemSelected() {
    int selectedIndex = getIMulticolumnListbox1().getSelectedIndex();
    if (selectedIndex < 0)
        return;
    java.util.Vector selected = new java.util.Vector();
    Object key = getIMulticolumnListbox1().getSelectedObject();
    selected.addElement(getIMulticolumnListbox1().getRowData(key));
    for (java.util.Enumeration e = selected.elements(); e.hasMoreElements()) {
        Object[] element = ((Object[]) e.nextElement());
        array[0] = element[0].toString();
        array[1] = element[1].toString();
        Item aItem = new Item(array);
        orderWindow.setSelectedItem(aItem);
    }
    return;
}
```

The `Item` class also provides standard getter methods for retrieving the values of individual data members. These methods are basic and not discussed here.
The selected Item object is passed as a parameter to the `setSelectedItem` method of the OrderEntryWdw object. Figure 249 shows the `setSelectedItem` method.

```java
public void setSelectedItem(Item selectedItem) {
    getItemTF().setText(selectedItem.getId());
    getDscTF().setText(selectedItem.getName());
    getQtyTF().requestFocus();
    updateStatus("Item information set: please enter quantity");
    item = selectedItem;
    return;
}
```

*Figure 249. The OrderEntryWdw setSelectedItem Method*

The `setSelectedItem` method then sets the appropriate text fields in the Order Entry window. After an item is selected, Figure 240 on page 209 shows the current state of the main window.

*Figure 250. The Order Entry Window with an Item Selected*

### 7.2.5 Verifying the Item and Adding It to the Order

After a quantity is entered, in the Quantity text field, the Add Item button is enabled. The Add Item button is connected to the `addOrderItem` method. Figure 251 on page 214 shows the code.
The item information from the text fields is added to the order multicolumn list box. A new OrderDetail object is created. It contains the item number and the quantity to order. The OrderDetail object is added to the Order object that we are in the process of creating.

After an item is added to the order list box, the Submit button is enabled. You can now add more items to the order or submit the order as shown in Figure 252.

7.2.6 Submitting the Order

The action of the Submit Button is connected to the submitOrder method of the OrderEntryWdw class. Figure 253 on page 215 shows the code.
As you can see, the `submitOrder` method is quite simple. It calls the `submitOrder` method of the ItemsDb class passing in the Order object which contains all the information necessary to create an order for the items selected.

```java
public void submitOrder() {
    String orderNumber = null;
    updateStatus("Processing order ...");
    try{
        orderNumber = getItemsDb().submitOrder(order);
    } catch(Exception e){
    }
    updateStatus("Order " + orderNumber + " successfully processed");
    return;
}
```

Clicking on the Submit button, causes the order to be submitted. If this is successful, the Status text field is updated to display the order number. This number is used to track the order. At this point, the user can end the application using the Exit button or start the start the order entry process over by clicking on the List Customers button to retrieve the customer list.
7.3 Building Servlets

Servlets are inherently multithreaded. Recall that stateful session beans are used to maintain state for one client. Even if multiple web users may be accessing the servlet, multithreading makes it appear as if the servlet is one client. State is not automatically maintained across method calls for each web client. This means that if we’re going to use a stateful session bean, we’re going to have to explicitly manage that connection.

The easiest and usually best way to connect servlets to session beans is to manage state information within the servlet, using stateless beans to perform business tasks. Our "stateless" methods in our ItemsDB class will do this.

ItemSessionServlet

The ItemSessionServlet servlet is fairly simple. It displays a list of the items available for ordering. It also allows the user to select items to add to their shopping cart. The connection occurs in exactly the same way as we saw previously in the application using the ItemsDB class.

CartServlet

The CartServlet servlet is a command-driven servlet that supports adding items to a shopping cart and placing an order. It uses the ItemsDB class to actually place the order.

Servlet Application Flow

The application flow happens like this:

1. When the ItemSessionServlet is started, it connects to the OrderEntryClerk session bean using the ItemsDB class. It maintains this object, so all threads use the same object.
2. When a user executes the ItemSessionServlet, it causes the "doGet()" method to be executed. The doGet() method executes the getAllItems() method of the ItemsDB class. It returns a Vector of items, which is used to populate the items table displayed on the form. This is shown in Figure 255.
Figure 255. The ItemSession servlet

3. The user selects the items they wish to order and clicks the "Add to Cart" button on the form, shown in Figure 256.

Figure 256. Add to Cart button

4. This posts the form to the CartServlet using the command "Add to Cart". The form shown in Figure 257 on page 219 is displayed. The CartServlet creates a shopping cart object in the servlet session object.

5. When the user is done shopping, they can click the "Check Out" button on the form to place an order. This action passes the command "Check Out" to the CartServlet servlet.
6. The CartServlet servlet posts a verification form, shown in Figure 258, and requests a customer number.

7. The user enters a customer number and clicks the “Place Order” button. This causes the placeOrder method, shown in Figure 259 on page 220, to be called. It calls the connectStateless and submitOrderStateless methods in the ItemsDb class, which in turn connect to the OrderPlacement session bean (which is stateless) and submit the order.
Another way to connect servlets to stateful session beans is to use the servlet's session object to maintain a handle to the bean. The servlet can then use the session bean to maintain state information about the web client.

This code does not use the ItemsDB class, but it would be possible to use it if the ItemsDB class were serializable and we could just put it in the session object. This would require overriding the serialization methods, so it would be arguably more complex than what we're doing here. The following code illustrates how a servlet can keep a handle to the bean.

```java
private void placeOrder(PrintWriter out, ShoppingCart cart, String custID) throws IOException {
    flexLog("CartServlet: placeOrder()...");
    Vector OrderLines = new Vector();

    Vector cartItems = cart.getItems();
    if (cartItems.size() > 0) {
        for (int i = 0; i < cartItems.size(); i++) {
            CartItem citem = (CartItem) cartItems.elementAt(i);
            com.ibm.itso.roch.cpwejb.interfaces.OrderDetail thisOrderDetail
                = new com.ibm.itso.roch.cpwejb.interfaces.OrderDetail(citem.getItemId(),
                Float.valueOf(citem.getPrice().replace('$','0')).floatValue(),
                1);
            OrderLines.addElement(thisOrderDetail);
        } // end for
        try{
            ItemsDB MyItemsDB = new ItemsDb();
            MyItemsDB.connectStateless();
            out.println("<H3>Your order has been processed.</H3> <BR>");
            out.println("Order number : ");
            out.println(MyItemsDB.submitOrderStateless(custID, OrderLines));
            out.println("<BR><BR>Thanks for your business");
            out.flush();
        } catch (Exception e) { e.printStackTrace(); out.println(e.getMessage()); } } // end try
}
```

Figure 259. The placeOrder method in the CartServlet

### 7.3.1 Guaranteeing One Web Client per Session Bean

Another way to connect servlets to stateful session beans is to use the servlet's session object to maintain a handle to the bean. The servlet can then use the session bean to maintain state information about the web client.

This code does not use the ItemsDB class, but it would be possible to use it if the ItemsDB class were serializable and we could just put it in the session object. This would require overriding the serialization methods, so it would be arguably more complex than what we're doing here. The following code illustrates how a servlet can keep a handle to the bean.
Notice that the getClerk() method is called to actually get the remote interface. The code for this is similar to the connect() and getInitialContext() methods of the ItemsDB class.
The advantage of doing things this way is that the servlet’s clients are the same as the EJB clients. This one-to-one mapping enables greater flexibility and code re-usability because the interface from an EJB perspective can be the same. The downside, however, is that the overhead is greater than it could be because of the larger number of RMI calls used. Just remember the trade-offs involved with each method presented here, and use whatever is appropriate to your application.
Chapter 8. AS/400 Unique Considerations

In this chapter, we discuss AS/400 unique considerations when developing, implementing, and deploying Enterprise Applications in WebSphere Application Server Advanced Edition 3.0.

8.1 Java on the AS/400 System

Java on the AS/400 system is compliant with the Sun specifications. This compliance with the standard means that Java code written to the Sun specification runs on the AS/400 system without modification or re-compilation, supporting the *write once, run anywhere* portability of Java. Currently, the AS/400 system supports JDK 1.1.4 in V4R2, JDK 1.1.6 in V4R3, and JDK 1.1.6, JDK 1.1.7, and JDK 1.2 in V4R4. Compliance with the standard also means that Java may not have the same characteristics as other languages on the AS/400 system.

Currently WebSphere Application Server Advanced Edition 3.0 can only be run with the JDK 1.1.7.

8.1.1 Java Files in the Integrated File System

Java source and class files are stored in the Integrated File System (IFS) in the root file system (/), QOPENSYS, or a user-defined file system. Java source files must be created on or copied to the IFS as ASCII files, not EBCDIC files. Notice that Java source and class file names are case-sensitive. The class files must be stored in a directory hierarchy, and both the package name and the CLASSPATH environment variable are used to find the classes in the same way classes are found on other platforms. The traditional AS/400 library, file, and member structure is not used in Java development or deployment.

8.1.2 Accessing AS/400 Resources from Java

Compliance with the Sun specifications also means that AS/400 specific resources cannot be accessed directly from Java using the standard APIs. Java currently supports stream file I/O and SQL database access, but currently has no mechanism for record-level I/O. Other AS/400 resources like user profiles, spool files, data queues, AS/400 messages, and CL commands also cannot be accessed using the standard Java APIs. A separate product, the AS/400 Toolbox for Java, provides access to these resources using Java classes that can run on the AS/400 system or any other Java-compliant platform.

8.1.3 Java Commands and Tools

Most of the Java tools and commands available on other platforms are available on the AS/400 system in the Qshell environment. The Qshell command interpreter is a command shell for the AS/400 system based on the POSIX and X/OPEN standards. The Qshell environment is started using the Start Qshell (STRQSH) command. The following Java tools are currently available under Qshell:

- *appletviewer* — Runs applets without using a Web browser.
- *jar* — Combines multiple files into a single Java ARchive (JAR) file.
- *java* — Runs Java programs.
- *javac* — Compiles Java programs.
- *javadoc* — Generates documentation.
• **javah** — Facilitates the implementation of Java native methods.
• **javakey** — Used for encryption, certificates, and digital signatures.
• **javap** — Disassembles compiled Java files.
• **native2ascii** — Converts native characters to Unicode.
• **rmic** — Generates the stub and class files for remote objects.
• **rmiregistry** — Starts a remote object registry on a server.

In addition to the Qshell Java tools, several Java CL commands are also available on the AS/400 platform. The **RUNJVA** or **JAVA** commands are used to run a Java class on the AS/400 system. The Create Java Program (CRTJVAPGM), Delete Java Program (DLTJVAPGM), Change Java Program (CHGJVAPGM) and Display Java Program (DSPJVAPGM) commands are used to create, delete, change and display optimized Java programs on the AS/400 system.

Java programs run in a slightly different environment than ILE programs on the AS/400 system. Because Java programs are inherently multi-threaded, they must run in **batch immediate** or **BCI** jobs on the AS/400 system. This has implications for both work management and debugging on the AS/400 system. These topics, as they relate to Enterprise JavaBeans, are explored later in this chapter.

More detailed information regarding developing Java applications on the AS/400 system is found in the redbook, *Building AS/400 Applications with Java*, SG24-2163. For additional information about the AS/400 Toolbox for Java, see the redbook *Building AS/400 Client/Server Applications with Java*, SG24-2152.

### 8.2 Java Performance Considerations

Java, by design, is an interpreted language. Java source code is compiled into an intermediate form which is stored in a class file. A Java class file must be interpreted at runtime by the Java Virtual Machine (JVM) which executes the interpreted code on a specific platform. This makes Java portable between different operating systems and hardware platforms. However, runtime performance is adversely affected by the interpreted nature of the Java language. For this reason, it is important to write your Java code to be as efficient as possible when runtime performance is critical.

#### 8.2.1 General Java Performance Techniques

There are a number of techniques that you can use to write efficient Java code. These techniques are not platform-specific and can apply to both client and server-side Java code. Some of the more useful performance techniques that may be applicable to Enterprise JavaBean development are shown in the following list:

- **Use** `javac -O` **to inline methods.**
- **Reuse objects** to reduce both object creation and garbage collection overhead.
- **Minimize the use of** `String` **objects, which are immutable.**
- **Use** `static final` **when creating constants.**
- **Store** character data in DB2/400 as Unicode, where possible, to eliminate conversions as the data moves into and out of the database file.
Store numeric data in DB2/400 as float or double, where possible, to eliminate conversions as the data moves into and out of the database file.

Use `get...(int columnIndex)` rather than `get...(String columnName)` with bean-managed persistence to retrieve fields from a database record.

8.2.2 AS/400-Specific Java Optimization

In addition to platform-neutral Java optimization techniques, most server platforms provide additional optimization that is specific to the platform. Starting with V4R4 on the AS/400 system, we have two options:

8.2.2.1 The Java Transformer

This is part of the JVM, which allows preprocessing of Java class, jar, or zip files and creates an optimized program object that is associated with the Java code. The optimized program object contains compiled 64-bit RISC machine instructions that execute without interpretation by the JVM.

8.2.2.2 Enhanced Just-In Time (JIT) support

This is a new V4R4 feature of the AS/400 Java support. When you specify the `jitc_de` JIT option, the JVM will perform just-in-time compilation for any java object that has not already been transformed. Once the object has been run through this level of JIT compilation the performance is very similar to that of transformed objects.

By default WebSphere Application Server Advanced Edition 3.0 utilizes the new JIT optimization in preference to the Java Transformer. You can use the Create Java Program (CRTJVAPGM) command to optimize Java classes at compile time instead of at runtime for classes specified in the CLASSPATH of the admin.properties file, making the initial runtime invocation of a class much faster.

8.2.2.3 Using the Java Transformer

When using the `CRTJVAPGM` command specify an optimization level of 30 or preferably 40. If you are debugging code, the optimization level should be left at 10. Use the Display Java Program (DSPJVAPGM) command to determine the optimization level of a class. Use the Delete Java Program (DLTJVAPGM) command to delete the AS/400 program object that is associated with a Java class, jar, or zip file.

For example, to create an optimized program object for the ItemBean class, enter the following command on the AS/400 command line:

```
CRTJVAPGM CLSF('ItemBean.class') OPTIMIZE(40)
```

Displaying the Java program associated with the ItemBean class results in the following display.
8.3 DB2/400 Considerations

Because much of the Enterprise JavaBean technology involves persistence, there are a number of DB2/400 considerations when using the WebSphere Application Server Advanced Edition 3.0. The standard for relational database access in Java is JDBC, which requires a JDBC driver for the underlying database implementation.

8.3.1 Native JDBC Driver versus AS/400 Toolbox JDBC Driver

Two JDBC drivers are available for DB2/400. One is part of the AS/400 Developer Kit for Java and is a native, type 2 JDBC driver. The other is part of the AS/400 Toolbox for Java and is a network, type 4 JDBC driver. The Native JDBC driver is the preferred driver for usage with WebSphere Application Server Advanced Edition 3.0. It is recommended that you use the native driver when accessing the local database. You must use the native driver when you require Java Transaction API (JTA) capabilities for two-phase commit functionality as this is not supported with the Toolbox driver. You can use the AS/400 Toolbox JDBC driver when you have simple remote database access without two-phase commit requirements.

The URL and the class names are different for each JDBC driver. For the native JDBC driver, the URL is \texttt{jdbc:db2://\text{SYSTEM}} where \text{SYSTEM} is an entry from the relational database directory on the AS/400 system. Use the Display Relational Database Directory Entries (DSPRDBDIRE) command to find the relational database entry for the *LOCAL AS/400 system. Typically on the AS/400 system, the relational database name is the same as the system name. The class name for the native JDBC driver is \texttt{com.ibm.db2.jdbc.app.DB2Driver}. The URL for AS/400 Toolbox JDBC driver is \texttt{jdbc:as400://\text{SYSTEM}} where \text{SYSTEM} is the name of the AS/400 system on which the database resides. The class name for the AS/400 Toolbox JDBC driver is \texttt{com.ibm.as400.access.AS400JDBCDriver}.

You use this information when defining the JDBC drivers and DataSources within the Administrative Client as shown in Figure 263 on page 227 and Figure 264 on page 227.
The JDBC driver used also affects where runtime information gets logged on the AS/400 system. If the AS/400 Toolbox JDBC driver is used, a server job, QZDASOINIT, is used to access the DB2/400 database. If the native JDBC driver is used, a server job, QSQRVR, is responsible for the JDBC access to the database.

When an Application Server is started a batch immediate job is started to run the Java code. The job, whose name consists of up to the first 10 characters of the Application Server name, is started in the QEJBSBS subsystem. As shown in Figure 265 on page 228, the Work with Active Jobs (WRKACTJOB) command is used to find the relevant job.
Figure 265. Finding the Application Server Job

The job log of the application server job indicates the SQL server job(s) used, as shown in Figure 266. The SQL Server (QSQSRVR) job logs are useful for debugging entity beans that use DB2/400.

Display Job Log

Job . . : DEFAULT_SE User . . : QEJB Number . . . : 082268
Public write authority on "<ustomerBeanDep.jar".
WebSphere server started with JDK 1.1.7.
Job 082270/QUSER/QSQSRVR used for SQL server mode processing

Press Enter to continue.
The QSQSRVR job log, shown in Figure 267, indicates the user profile being used by this server. Note that the user profile will be the user profile specified retrieving the JDBC connection. In this instance the job is one of the internal WebSphere Application Server Advanced Edition 3.0 database jobs.

Figure 267. QSQSRVR Job Log

These job logs show any SQL errors that occur in the application running under the control of this WebSphere Application Server.

8.3.2 Journaling

The isolation level set for transactions in the control property of the deployment descriptor may also affect your AS/400 system setup. The isolation level set in the deployment descriptor is passed to the underlying DB2/400 database. If the isolation level requires commitment control, journaling must be enabled for the database file. Journaling is enabled by creating a journal receiver using the Create Journal Receiver (CRTJRNRCV) command, creating a journal by using the Create Journal (CRTJRN) command, and then starting the journal against the database file using the Start Journaling Physical File (STRJRNPF). Journaling is automatically set up and started for any tables created as part of a collection.

8.4 Debugging WebSphere Applications on the AS/400 System

To debug an enterprise bean running on AS/400 using the AS/400 system debugger:

- Compile the EJB with option -g of javac.
- Place the Java source and debuggable class files in the same debug directory on the AS/400 system. The Java source files and the debuggable class files must be extracted to a debug directory on the AS/400 system from the jar file. Debugging directly from the jar file does not work.
Specify the directory in the classpath of the application server.

In the "command line arguments" field of the general panel of the EJB Container Application Server property panel, add the following:

-classpath <path_of_the_servlet_files>.

Do not include the sub-directories which make up the package qualification for the class.

- Run CRTJVAPGM, against the class file in the debug directory you want to debug, with OPTIMIZE(10). This is to override the OS400 optimization parameter in the admin.properties file, which is normally set to 30 (which gives you no access to the variables in debug mode).

- Start the application server. In this way, the class in the deployed jar file is not executed any longer. The class in the debug directory is executed instead.

- Determine the job number, user and name of the AS/400 job which represents the application server containing the EJB Container. This information is displayed in the Process ID property for the application server on the administrative console (normally nnnn/QEJB/FirstTenLettersOfTheAppServerName).

- Enter the command:

  STRSRVJOB <job_name>

  Supply the job number, user and name using the information from the previous step.

- Enter the Command

  STRDBG CLASS(package/classname)

  Insert the breakpoints you wish. You may also add other classes using F14 and insert further breakpoints.

- Run the client code which causes the enterprise bean to be invoked. When the EJB code at which you set a breakpoint is reached, the source view of the code at that point will be displayed in the AS/400 session from which you started your debug session. From here, you can use the system debugger commands and function keys to set more breakpoints and to step through your code, as if it were an RPG program.

### 8.4.1 Retrieving Java Source Files

There may be instances where you need to use the WebSphere Administrative Client to deploy the EJB jar file. This deployment process does not place the generated Java source files on your system. They are required for debugging purposes. In this case, there is a java application you can use to generate the Java source files.

Ensuring that you have the prerequisite classes and jar files in your classpath, add the following IBM jar files to the classpath:

- /QIBM/ProdData/WebAsAdv/lib/deployTool.jar
- /QIBM/ProdData/WebAsAdv/lib/ujc.jar
- /QIBM/ProdData/WebASAdv/lib/ejs.jar
- /QIBM/ProdData/WebASAdv/lib/iioptools.jar
Start a QSHELL session, using the STRQSH command. As shown in Figure 268, run the com.ibm.ivj.ejbtoolsdeployment.EJBDeploy class.

The important parameter is the -noclean option. This ensures that the java source files are placed in the working directory. All directories that you specify must exist, otherwise the tool will fail. Once you have generated the source files, you can follow the on-line debugging instructions for EJBs.

This java application is the same one used under the covers with WebSphere. You can also use this tool to deploy EJBs.
Appendix A. Special Notices

This publication is intended to help anyone who wants to use the IBM WebSphere Application Server Advanced Edition 3.0 for AS/400. The information in this publication is not intended as the specification of any programming interfaces that are provided by the IBM WebSphere Application Server Advanced Edition product. See the PUBLICATIONS section of the IBM Programming Announcement for the IBM WebSphere Application Server Standard Edition product for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM’s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM’s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (“vendor”) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer’s ability to evaluate and integrate them into the customer’s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.
Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:
IBM ®

The following terms are trademarks of other companies:
C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.
Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.
Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.
PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.
ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.
UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.
SET and the SET logo are trademarks owned by SET Secure Electronic Transaction LLC.
Other company, product, and service names may be trademarks or service marks of others.
Appendix B. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

B.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How to Get ITSO Redbooks” on page 237.

- Building AS/400 Client/Server Applications with Java, SG24-2152
- Building AS/400 Applications with Java, SG24-2163
- Building AS/400 Internet Based Applications with Java, SG24-5337
- Web Enabling AS/400 Applications with IBM WebSphere Studio, SG24-5634
- Building AS/400 Applications for IBM WebSphere Standard Edition 2.0, SG24-5635

B.2 Redbooks on CD-ROMs

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at http://www.redbooks.ibm.com/ for information about all the CD-ROMs offered, updates and formats.

<table>
<thead>
<tr>
<th>CD-ROM Title</th>
<th>Collection Kit Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>System/390 Redbooks Collection</td>
<td>SK2T-2177</td>
</tr>
<tr>
<td>Networking and Systems Management Redbooks Collection</td>
<td>SK2T-6022</td>
</tr>
<tr>
<td>Transaction Processing and Data Management Redbooks Collection</td>
<td>SK2T-8038</td>
</tr>
<tr>
<td>Lotus Redbooks Collection</td>
<td>SK2T-8039</td>
</tr>
<tr>
<td>Tivoli Redbooks Collection</td>
<td>SK2T-8044</td>
</tr>
<tr>
<td>AS/400 Redbooks Collection</td>
<td>SK2T-2849</td>
</tr>
<tr>
<td>Netfinity Hardware and Software Redbooks Collection</td>
<td>SK2T-8046</td>
</tr>
<tr>
<td>RS/6000 Redbooks Collection (BkMgr Format)</td>
<td>SK2T-8040</td>
</tr>
<tr>
<td>RS/6000 Redbooks Collection (PDF Format)</td>
<td>SK2T-8043</td>
</tr>
<tr>
<td>Application Development Redbooks Collection</td>
<td>SK2T-8037</td>
</tr>
</tbody>
</table>

B.3 Other Publications

These publications are also relevant as further information sources:

- Enterprise JavaBeans, ISBN 1-56592-605-6
- ILE RPG for AS/400 Programming Guide, SC09-2507
- OS/400 Work Management, SC41-5306
- Performance Tools/400, SC41-5340
- AS/400 Database Programming, SC41-5701
- Distributed Database Programming, SC41-5702
• Qshell Interpreter (See http://www.as400bks.com/ on the AS/400 Internet book server. **Note:** This book is only available on the Internet.)

• AS/400 Java Developer Kit for Java (See http://www.as400bks.com/ on the AS/400 Internet book server. **Note:** This book is only available on the Internet.)

• AS/400 Toolbox for Java (See http://www.as400bks.com/ on the AS/400 Internet book server. **Note:** This book is only available on the Internet.)


• Couthard, Phillip and Farr, George. **Java for RPG Programmers.** IBM Press, 1998 (GK2T-9890; available on diskette).

• Cooper, Alan. **About Face.** Foster City, CA: IDG Books Worldwide.


How to Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

  
  Search for, view, download, or order hardcopy/CD-ROM redbooks from the redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this redbooks site.

  Redpieces are redbooks in progress; not all redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**
  
  Send orders by e-mail including information from the redbooks fax order form to:
  
  **e-mail address** usib6ipl@ibmmail.com

  Contact information is in the “How to Order” section at this site:
  

- **Telephone Orders**
  
  Contact information is in the “How to Order” section at this site:
  

  **United States (toll free)** 1-800-879-2756

  **Canada (toll free)** 1-800-IBM-4YOU

  **Outside North America** Country coordinator phone number is in the “How to Order” section at this site:
  

- **Fax Orders**
  
  Contact information is in the “How to Order” section at this site:
  

  **United States (toll free)** 1-800-445-9269

  **Canada** 1-403-267-4455

  **Outside North America** Fax phone number is in the “How to Order” section at this site:
  

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the redbooks Web site.

---

**IBM Intranet for Employees**

# IBM Redbook Fax Order Form

Please send me the following:

<table>
<thead>
<tr>
<th>Title</th>
<th>Order Number</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

First name __________________________ Last name __________________________

Company __________________________

Address __________________________

City __________________________ Postal code __________________________ Country __________________________

Telephone number __________________________ Telefax number __________________________ VAT number __________________________

☐ Invoice to customer number

☐ Credit card number __________________________

Credit card expiration date __________________________ Card issued to __________________________ Signature __________________________

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFP</td>
<td>Advanced Function Printing</td>
<td>JAR</td>
<td>Java archive</td>
</tr>
<tr>
<td>ASP</td>
<td>Active Server Pages</td>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>APA</td>
<td>All Points Addressable</td>
<td>JDK</td>
<td>Java Development Toolkit</td>
</tr>
<tr>
<td>AWT</td>
<td>Abstract Windowing Toolkit</td>
<td>JFC</td>
<td>Java Foundation Classes</td>
</tr>
<tr>
<td>CA</td>
<td>Certificate Authority</td>
<td>JIT</td>
<td>Just in Time Compiler</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
<td>JPDC</td>
<td>Java Performance Data Converter</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
<td>JSP</td>
<td>Java Server Pages</td>
</tr>
<tr>
<td>CGI</td>
<td>Communications Gateway Interface</td>
<td>JVM</td>
<td>Java Virtual Machine</td>
</tr>
<tr>
<td>CPW</td>
<td>Commercial Processing Workload</td>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>DAX</td>
<td>Data Access Builder</td>
<td>MI</td>
<td>Machine Interface</td>
</tr>
<tr>
<td>DCM</td>
<td>Digital Certificate Manager</td>
<td>MIME</td>
<td>Multi-purpose Internet Mail Extensions</td>
</tr>
<tr>
<td>DDM</td>
<td>Distributed Data Management</td>
<td>MVC</td>
<td>Model View Controller</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic Link Library</td>
<td>NSAPI</td>
<td>Netscape API</td>
</tr>
<tr>
<td>DPC</td>
<td>Distributed Program Call</td>
<td>ODBC</td>
<td>Open Database Connectivity</td>
</tr>
<tr>
<td>EAB</td>
<td>Enterprise Access Builder</td>
<td>OOA</td>
<td>Object Oriented Analysis</td>
</tr>
<tr>
<td>EJB</td>
<td>Enterprise JavaBeans</td>
<td>OOD</td>
<td>Object Oriented Design</td>
</tr>
<tr>
<td>FFST</td>
<td>First Failure Support Technology</td>
<td>OOP</td>
<td>Object Oriented Programming</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
<td>ORB</td>
<td>Object Request Broker</td>
</tr>
<tr>
<td>GWAPI</td>
<td>Go Web Server API</td>
<td>PEX</td>
<td>Performance Explorer</td>
</tr>
<tr>
<td>HTML</td>
<td>Hypertext Markup Language</td>
<td>PCML</td>
<td>Program Call Mark-up Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transmission Protocol</td>
<td>PDML</td>
<td>Panel Definition Mark-up Language</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Hypertext Transmission Protocol</td>
<td>PTDV</td>
<td>Performance Trace Data Visualizer</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines Corporation</td>
<td>PTF</td>
<td>Program Temporary Fix</td>
</tr>
<tr>
<td>ICAPI</td>
<td>IBM Connection API</td>
<td>RAD</td>
<td>Rapid Application Development</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
<td>RAWT</td>
<td>Remote Abstract Windowing Toolkit</td>
</tr>
<tr>
<td>IDL</td>
<td>Interface Definition Language</td>
<td>RMI</td>
<td>Remote Method Invocation</td>
</tr>
<tr>
<td>IIOP</td>
<td>Internet inter-ORB protocol</td>
<td>SCS</td>
<td>SNA Character Set</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
<td>SLIC</td>
<td>System Licensed Internal Code</td>
</tr>
<tr>
<td>ITSO</td>
<td>International Technical Support Organization</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Abbreviation</strong></td>
<td><strong>Full Form</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>--------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>S-MIME</em></td>
<td>Secure Multi-purpose Internet Mail Extensions</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>SNA</em></td>
<td>System Network Architecture</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>SSL</em></td>
<td>Secure Sockets Layer</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>TCP</em></td>
<td>Transmission Control Protocol</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>TIMI</em></td>
<td>Technology Independent Machine Interface</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>UML</em></td>
<td>Unified Methodology Language</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>URL</em></td>
<td>Universal Resource Locator</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>VCE</em></td>
<td>Visual Composition Editor</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>WAS</em></td>
<td>WebSphere Application Server</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>WWW</em></td>
<td>World Wide Web</td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>XML</em></td>
<td>Extensible Mark-up Language</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Index

Numerics
5733-WA2 4
5733-WA3 4
5769-DG1 5

A
abbreviations 239
access package classes 46, 49
acronyms 239
ActiveX 77
administrative terminology 17
APGM 224
application servers 18
AS/400 Developer Kit for Java 75, 76, 226
AS/400 system
  EJB technology 76
  Java environment 75
  Java performance considerations 224
  specific Java optimization 225
  unique considerations 223
AS/400 to Java mapping 138
AS/400 Toolbox for Java 42, 48, 95, 224, 226

B
BEA WebLogic Application Server 70
Bean Managed Persistence (BMP) 93
bibliography 235
BlueStone Software 70
building Java applications 200
  using Enterprise JavaBeans 191

C
CLASSPATH 223
clustering 38
CNInitialContextFactory 44, 52
command
  Create Java Program (CRTJVAPGM) 224, 225
  Delete Java Program (DLTJVAPGM) 225
  Display Java Program (DSPJVAPGM) 224, 225
  Run Java (RUNJAVA) 224
  Start Journaling Physical File (STRJRNPF) 229
  STRQSH 223
commitment control 229
Common Object Request Broker Architecture (CORBA) 67
Component Broker 78
Component Broker Connector (CBConnector) 78
Component Broker Toolkit (CBToolkit) 78
Component Object Model (COM) 77
connection pools 172
customer table 172, 205
customer table
  Customer Table Layout (CSTMR) 89
Create Journal (CRTJRNR) command 229
Create Journal Receiver (CRTJRNRCV) command 229
creating a JDBC driver 24
creating a test client 125
creating multiple administrative servers 19
CRTJRNR command 229
CRTJRNRCV command 229
CRTJVAPGM 224
CRTJVAPGM command 224, 225
customer table 172, 205
Customer Table Layout (CSTMR) 89
creating a JDBC driver 24
creating a test client 125
creating multiple administrative servers 19
 CRTJRNR command 229
CRTJRNRCV command 229
CRTJVAPGM 224
CRTJVAPGM command 224, 225
 Delta JDBC Driver 44
 DataSource 24, 34, 119, 226
  adding to an EJB container 29
  creating 28
  migrating to 38
  servlet example 43
 DB2/400 224, 226, 228
deployment descriptor 71, 72, 110, 113, 229
 district table 172
deployment descriptor
  District Table Layout (Dstrct) 89
deployment descriptor domain 17
deployment descriptor
  domain object 192
  DSPJVAPGM command 224, 225, 231

E
EBCDIC 223
EJB
  container 67
  roles 72
  application assembler 73
  application deployer 74
  container provider 74
  Enterprise Bean provider 73
  server 67, 68
  server architecture 69
  specification 72
  technology on the AS/400 system 75
EJB Container 18
EJB Group 98
SmartGuide 99
EJB Roles
  server provider 74
EJB specification 63
 ejbCreate() method 104, 167, 176
enterprise beans 18
Enterprise JavaBean 63
  application development scenario 191
  architecture 64, 67
  components 67
  container 68
  contract 68
  services 67
  specification 63, 72

© Copyright IBM Corp. 1999

241
Entity beans  67, 68, 70, 93, 172
application development scenario  172
bean managed persistence  70, 175
container managed persistence  70, 149
container managed persistence  93
generating data for mapping  149
generating from existing data  93, 134

**H**
home interface  71, 193
HTML document  36
HTTP request  36

**I**
IBM Connection Manager APIs  34
IBM Connection Manager issues  35
IBM HTTP Server for AS/400  5
IIOP (Internet inter-ORB protocol)  77
InitialContext  193
integrated development environments (IDE)  64
integrated file system (IFS)  223
interface definition language (IDL)  77
Internet shopping application  191
isolation level  112
Item EJB  175, 225
item table  172
Item Table Layout (ITEM)  91
ItemsDb class  219
   connect method  194
findRangeOfItems method  196
getAllCustomer method  195
getAllCustomers method  206
getInitialContext method  193
getItems method  195, 211
submitOrder method  197
using without a GUI  199
verifyCustomer method  196

**J**
Java 1.1 Runtime Environment (JRE)  42
Java applications  191
Java Server Page (JSP)  56
JavaBean  63
   Enterprise JavaBeans  63
javax.ejb  193
javax.naming  193
JDBC  42, 206, 211, 226, 227
   AS/400 Toolbox for Java driver  226
   isolation levels  110
JDBCCatalogSupport class  50
JDBCPoolCatalog class  46, 51
JDK  223
J-Integra  77
JNDI  52, 193
journal receiver  229
Journaling  229
JSP
   porting  61
   specification  61
JSP Support in WebSphere Version 3.0  60
JSDK (JavaServer Web Development Kit)  33, 59
JVM  224

**K**
keyed physical file  97

**L**
life cycle methods  175
Linac  77

**M**
migrating WebSphere version 2.0 servlets  45

**N**
node  17, 18
Novera  70

**O**
object diagram  169
object model  192
Order Entry application  79
order entry application
   signon dialog  202
   SltCustWdw class  204
Order Line Table Layout (ORDLIN)  90
order table  172
Order Table Layout (ORDERS)  90
OrderEntryClerk EJB  176, 177, 184, 192, 195, 197
OrderEntryWdw Class  201
   setSelectedCust method  208
   submitOrder method  215
OrderPlacement EJB  176, 177
   deployment descriptor  178, 184
   home interface  179
   implementation  180, 185
   placeOrder method  182
   remote interface  179
ORDERS  175
ORDLIN  175

**P**
persistence  38, 70, 175, 226
   bean managed  175
   container managed  175
porting servlets  34
primary key constraints  97

**Q**
QOPENSYS  223
Qshell  223
QSQSRVR  227, 228, 229
QZDASOINIT  227
R
remote interface 71, 101, 193
Remote interface page 129
remote method invocation (RMI) 76
RISC 225
RPG application flow 81
RPG order entry application 79
application flow 81
components 81
customer transaction 80
customer transaction flow 82
database 80
starting the application 82
tables 89
run-as-mode 111, 112

S
SanFrancisco 77
Sapphire/Web 70
schema 93
Schema Browser 96
scrapbook 199
security 69
serializable 38
server 68
servlet 2.0 API 34
servlet API 2.1 34
servlet chaining 36
servlets 191, 217, 220
session bean
  CPWEJB application 176
session beans 67, 68, 70, 172, 176
  stateful 70, 177
  stateless 70, 177
SltCustWdw class 204
  custSelected method 207
  populateCustBox method 205
SltItemWdw Class
  itemSelected method 211
  populateItemBox method 210
stateful session bean 71
stateless session beans 70
stock EJB 175
stock table 172
Stock Table Layout (Stock) 91
STRHOSTSVR command 11
STRJRNPF command 229
STRQSH command 223
support package 191
  customer class 191, 207
  item class 191, 212
  order Class 208
  order class 191
  OrderDetail class 191, 214

T
topology 17
transaction 69, 110, 188, 229

U
unified modeling language (UML) 169
URL encoding 55

V
Visual Basic 77
VisualAge for Java 1, 201
VisualAge for Java Enterprise Edition 3.0 34, 93
  creating a test client 125
VisualAge for Java Scrapbook 199

W
WebSphere 1
  WebSphere Performance Pack 1
  WebSphere Site Analysis 1
  WebSphere Studio 1, 34
WebSphere Administrative Console 18
WebSphere Application Server 1, 2, 78
WebSphere Application Server Advanced Edition 3.0 1, 2, 33, 36, 70
  administration 17
  administrative client 5
  Base option 7
  configuration 17
  installation 6
  installing a jar file 115
  multi-instance support 5
  Server option 7
  servlet support 33
  starting an enterprise bean 123
  starting and stopping 20
  un-installing 17
ITSO Redbook Evaluation

Building AS/400 Applications for IBM WebSphere Application Server Advanced Edition 3.0
SG24-5691-00

Your feedback is very important to help us maintain the quality of ITSO redbooks. Please complete this questionnaire and return it using one of the following methods:

- Use the online evaluation form found at http://www.redbooks.ibm.com/
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Which of the following best describes you?
- Customer
- Business Partner
- Solution Developer
- IBM employee
- None of the above

Please rate your overall satisfaction with this book using the scale:
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction

Please answer the following questions:

Was this redbook published in time for your needs? Yes___ No___

If no, please explain:

What other redbooks would you like to see published?

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)
