

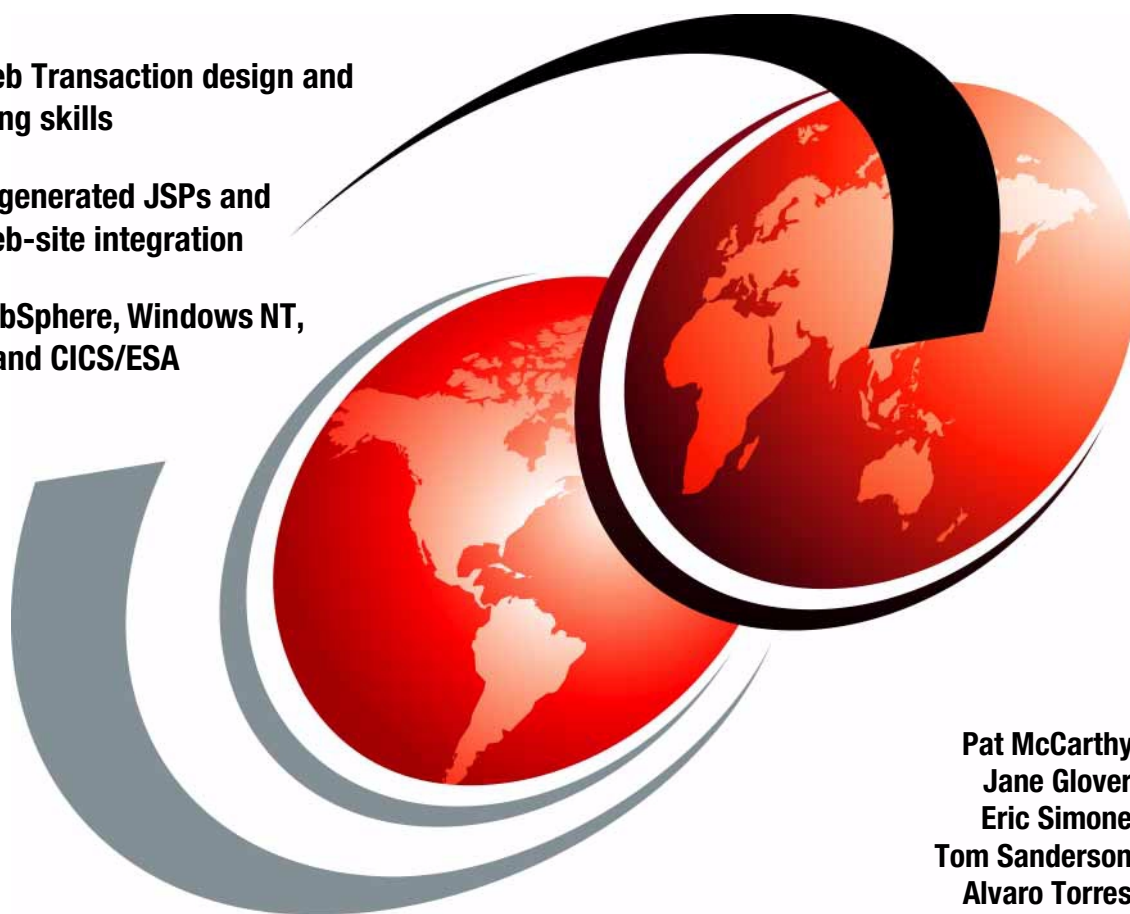
Building Enterprise Web Transactions

using VisualAge Generator JavaBeans and JSPs

Develop Web Transaction design and programming skills

Customize generated JSPs and perform Web-site integration

For IBM WebSphere, Windows NT, TX Series, and CICS/ESA



Pat McCarthy
Jane Glover
Eric Simone
Tom Sanderson
Alvaro Torres

ibm.com/redbooks

Redbooks



International Technical Support Organization

**Building Enterprise Web Transactions
using VisualAge Generator JavaBeans and JSPs**

May 2000

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix B, "Special notices" on page 369.

First Edition (May 2000)

This edition applies to VisualAge Generator V4:

- VisualAge Generator Developer for OS/2 and Windows NT Version 4.0, 5697-G98
- VisualAge Generator Server for OS/2, AIX, Windows NT, HP-UX and Sun Solaris Version 4.0, 5639-G97
- VisualAge Generator Templates for OS/2 and Windows NT, 5639-H09

Comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. OWR Building 80-E2
650 Harry Road
San Jose, California 95120-6099

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 2000. 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

Part 1. VisualAge Generator Web Transaction overview	1
Chapter 1. Introduction to VisualAge Generator Web Transactions	3
1.1 Rapid Web Transaction application development concepts	4
1.1.1 Problem statement	4
1.1.2 Architectural analysis	5
1.1.3 Solution	7
1.2 Web server-based transaction system implementation options	11
1.2.1 Native Java servlet and JSP programming	12
1.2.2 VisualAge Generator Web Transaction programming	13
1.2.3 Supported runtime configurations	14
1.2.4 Summary	16
Chapter 2. Web Transaction system implementation	17
2.1 Web Transaction development	18
2.1.1 UI Record definition	19
2.1.2 UI Record to HTML mapping	20
2.1.3 HTML forms in a UI Record	25
2.1.4 Web Transaction definition	26
2.1.5 Testing	27
2.2 Generation of Java components and runtime program	29
2.2.1 Java terminology	30
2.2.2 Programs	32
2.2.3 UI Records	33
2.2.4 UI Record interface bean (UIrecord Bean)	34
2.2.5 Java Server Page produced by VisualAge Generator	35
2.2.6 User edit tables	35
2.2.7 User message tables	36
2.3 Runtime system implementation	36
2.3.1 Basic processing concepts	37
2.3.2 Gateway Servlet	44
2.3.3 Session ID Manager (SIDM)	46
2.3.4 Web Transaction runtime scenario	47
Chapter 3. HTML and UI Record definition	55
3.1 An HTML document	56
3.2 General HTML tags	56
3.2.1 TITLE	56
3.2.2 General displayable text	57
3.3 FORMs	57
3.3.1 The HTML FORM tag	57

3.3.2	UI Record FORM support	61
3.3.3	Creating FORM fields in a UI Record	62
3.3.4	Match valid edit tables	65
3.3.5	Variable lists	65
3.3.6	UI type FORM	65
3.4	LINKs	68
3.4.1	LINKs in UI Record definition	68
3.5	HTML layout and look-and-feel	70
3.6	UI Record specific features	71
3.6.1	Record properties	71
3.6.2	Special VisualAge Generator UI types	71
3.6.3	VisualAge Generator features for UI Record data items	72
3.6.4	Data Item Edits	73
Chapter 4. Java Server Pages and the UI Record interface bean API		75
4.1	JSP syntax	75
4.1.1	Scriptlets	75
4.1.2	Expressions	76
4.1.3	Bean tag	77
4.1.4	Directives	79
4.2	The interface bean API	79
4.2.1	UI Record Bean Interface	80
4.2.2	VGDataElement Interface	81
<hr/>		
Part 2. Web Transaction design and development		83
Chapter 5. Web Transaction design concepts and considerations		85
5.1	Concepts	85
5.1.1	Main Transaction and Web Transaction program comparison	85
5.1.2	Web Transaction state saving options	87
5.2	Program structure options	89
5.2.1	Using CONVERSE UI Record (complete state)	89
5.2.2	Using XFER Program WSRecord, UI Record (controlled state)	91
5.2.3	Using XFER ' ', UI Record (stateless)	93
5.3	Implementing self-managed state support for XFER ' ' programs	94
5.3.1	Introduction	94
5.3.2	Global state	95
5.3.3	Conversation state	95
5.3.4	Implementation	96
5.4	Design considerations	98
5.4.1	Data transfer	98
5.4.2	UI Record edits	100
5.5	System architecture considerations	103

Chapter 6. Web Transaction Web site development	105
6.1 Development process overview	105
6.2 Roles and skills in the development process	106
6.3 Function and presentation	107
6.4 Level 1: the stranded Web Transaction developer	109
6.5 Level 2: Web Transaction developer and JSP developer	111
6.6 Level 3: Web Transaction developer, JSP developer, HTML designer	112
Chapter 7. Transforming TUIs into Web Transactions	115
7.1 Considerations	115
7.2 Phase 1 — analysis	116
7.3 Phase 2 — basic transformation	116
7.4 Phase 3 — make it more Web-like	120
7.5 Phase 4 — modify default JSP	123
<hr/>	
Part 3. Web Transaction programming and front end customization	125
Chapter 8. Developing Web Transaction programming skills	127
8.1 Program structure	127
8.1.1 Loading code base	128
8.1.2 Converse model programming	128
8.1.3 Single segment (XFER PGM) programming	134
8.1.4 Single segment (XFER ' ') programming	142
8.2 Implementing global state management	150
8.2.1 UI Record-based state management implementation	150
8.2.2 Working storage record-based state management	163
8.2.3 Self-managed state implementation (XFER ' ' model)	168
Chapter 9. VisualAge Generator Templates Web Transactions	173
9.1 Preparing the workspace	173
9.2 Relational table definition using database import	174
9.3 Business Object definition	175
9.4 Interface Unit definition	176
9.5 Generation Option definition and system generation	177
9.6 Test the generated system	179
9.7 Customization	179
Chapter 10. Demonstration system	181
10.1 Components	181
10.2 Processing overview	182
10.3 Transfer processing and data management	183
10.4 State management	184
10.5 Input validation	185

10.6 Testing path	186
Chapter 11. Front-end customization techniques	195
11.1 Level 1: What's a Web Transaction developer to do?	195
11.1.1 Correcting the generated default JSP	195
11.1.2 Easy elements	197
11.1.3 Implementing help	198
11.1.4 Protecting FORM fields	200
11.1.5 Making the default JSP look better	201
11.2 Level 2: Enter the JSP developer	205
11.2.1 Advanced JSP customization	206
11.2.2 WebSphere Studio	210
11.2.3 Modification Using WebSphere Studio	216
11.3 Level 3: Integrating Web Transactions into a Web site	226
11.3.1 Explanation of simultaneous development.	226
11.3.2 Web site planning issues.	227
11.3.3 Development steps	228
11.3.4 Front end Web site development.	228
11.3.5 Bringing the two sides (front and back) together	234
<hr/>	
Part 4. Environment configuration and system implementation	245
Chapter 12. Runtime environment scenario implementation.	247
12.1 Windows NT Web Transactions	247
12.1.1 Software requirements	247
12.1.2 Implementation tasks	250
12.2 CICS for NT Web Transactions	250
12.2.1 Software requirements	250
12.2.2 Implementation tasks	253
12.3 CICS/ESA Web Transactions	254
12.3.1 Software requirements	254
12.3.2 Implementation tasks	255
Chapter 13. VisualAge Generator Web Transaction runtime setup	257
13.1 Base software	257
13.1.1 DB2 Client Application Enabler	257
13.1.2 VisualAge for C++.	259
13.1.3 VisualAge Generator Server	259
13.1.4 Setting up FTP support for program preparation	259
13.2 Web Transaction gateway interface configuration (csogw.properties)	263
13.2.1 Control entries	264
13.2.2 Application entries.	264
13.2.3 serverLinkage entries	265

13.2.4	Protocol specific entries	266
13.2.5	Overriding serverLinkage entries.	266
13.3	Windows NT Web Transactions	268
13.3.1	VisualAge Generator control settings	268
13.3.2	Configure TCP/IP listener support.	269
13.3.3	Communications configuration	270
13.4	CICS for NT Web Transactions	270
13.4.1	Base software for CICS system.	270
13.4.2	Region definition	271
13.4.3	CICS DB2 attachment	271
13.4.4	Add CICS system listeners	275
13.4.5	Define CICS user	277
13.4.6	Add VisualAge Generator runtime and debug transactions	277
13.4.7	VisualAge Generator control settings	278
13.4.8	Communications configuration	279
13.5	CICS/ESA Web Transactions	279
13.5.1	Install the PCOMM software	280
13.5.2	CICS connection definition	296
13.5.3	CICS security	299
13.5.4	Set up VisualAge Generator Host Services	299
13.5.5	Communications configuration	299
Chapter 14. WebSphere Application Server setup		301
14.1	Installed software base.	301
14.2	IBM WebSphere Application Server for Windows NT	302
14.2.1	Install IBM WebSphere Application Server software	302
14.2.2	Startup WebSphere Application Server	303
14.2.3	Configure a new Application Server	304
14.2.4	Define VisualAge Generator Gateway Servlet.	312
14.2.5	Customize JSPs (as required).	317
14.2.6	Deploy JSPs and GIFs	317
14.2.7	Configure the vgj.properties file.	318
14.2.8	Set up VisualAge Generator session ID manager	318
14.2.9	Start application server	318
14.3	Adding CICS support	319
14.3.1	CICS Transaction Gateway	319
14.3.2	Customization for TX Series (CICS NT) access.	320
14.3.3	Customization for CICS/ESA access	321
14.3.4	CICSTERM behavior and signon capable terminals	322
14.4	VisualAge for Java WebSphere test environment	324
14.4.1	Setup	324
14.4.2	Configure WebSphere Test Environment	327
14.4.3	Configure GatewayServlet	328

14.4.4	Add generated components for Web Transaction	331
14.4.5	Test generated Web Transaction in VisualAge for Java	332
Chapter 15. Web Transaction generation 335		
15.1	Windows NT Web Transactions — base system deployment	336
15.1.1	Generation	336
15.1.2	Configure Gateway Servlet access	337
15.2	CICS NT Web Transactions — base system deployment	337
15.2.1	Generation	337
15.2.2	Define your generated Web Transaction(s) to CICS	338
15.2.3	Configure Gateway Servlet access	339
15.3	CICS/ESA Web Transactions — base system deployment	339
15.3.1	Generation	339
15.3.2	Define your generated Web Transaction(s) to CICS	339
15.3.3	Configure Gateway Servlet access	340
Chapter 16. Running Web Transactions 341		
16.1	Deploy generated code	341
16.1.1	JSPs, JavaBeans, and tables used in a UI Record	341
16.1.2	Web Transaction program materials	342
16.1.3	Invoking Web Transaction from default entry point JSP.	342
16.2	Runtime processing	344
16.2.1	System Startup	344
16.2.2	Gateway Servlet invocation.	345
16.2.3	Gateway Servlet processing	347
16.2.4	Windows NT Web Transaction processing	350
16.2.5	CICS Web Transaction processing	350
16.2.6	Debugging Web Transactions at runtime with CEDF	351
16.3	Security	352
16.3.1	Logon technique	352
16.3.2	Transparent login to CICS.	360
16.3.3	Other options	361
16.3.4	Secure HTTP	361
<hr/>		
Part 5. Appendices		363
Appendix A. Sample code and other materials 365		
A.1	VisualAge Generator code.	365
A.2	WebSphere Studio.	366
A.3	Database	367
A.4	Additional materials	367

Appendix B. Special notices	369
Appendix C. Related publications	373
C.1 IBM Redbooks publications	373
C.2 IBM Redbooks collections	373
C.3 Other resources	373

x Building Enterprise Web Transactions using VisualAge Generator JavaBeans and JSPs

Figures

1. VisualAge Generator V4 overview	3
2. Problem: Implementation and communication (tier-1, tier-2, and tier-3).	4
3. Architectural affinity: 3270 and Web systems.	6
4. Web Transaction definition	8
5. Web Transaction implementation overview	9
6. Web Transaction support for automated tier-to-tier communication.	11
7. Anatomy of a Web system	12
8. Web Transaction runtime	14
9. Web Transaction runtime configurations using WebSphere Application Server. . .	15
10. Implementation process for VisualAge Generator V4 Web transactions	17
11. Process life cycle	18
12. The UI Record.	19
13. UI Record properties.	20
14. UI Record mapping to HTML	22
15. HTML Forms.	26
16. Web Transaction processing structures	27
17. Testing a Web-based system	28
18. Generation of a Web-based system	30
19. Web-based system development	36
20. Servlets.	39
21. Java Server Pages	41
22. The HttpSession object.	42
23. CONVERSE and XFER program Web Transaction structure options	47
24. CONVERSE and XFER program Web Transaction runtime processing	48
25. XFER ' ' , UIRecord Web Transaction structure design option.	51
26. Main and Web Transaction program CONVERSE model comparison.	86
27. Web Transaction processing for each program type	88
28. CONVERSE (complete state) program design	90
29. Data lost after program link during CONVERSE.	91
30. XFER Program WSRecord, UI Record (controlled state) program design	92
31. XFER ' ' , UI Record (stateless) program design	93
32. DB2 table definition for self-managed state	97
33. Web Transaction development process overview	105
34. Generated default JSP	107
35. Default JSP after HTML presentation enhancements.	108
36. Level 1 development process diagram	109
37. Sample UI Record definition	110
38. Generated default JSP for sample UI Record.	110
39. Level 2 development process diagram	111
40. Process Diagram for Level 3 Development	113

41. Setting submit values before CONVERSE	118
42. Conversed UI Record	119
43. Adding program link	121
44. UI Record with program link	122
45. CSTCNV Web Transaction program structure	129
46. Customer Info Web page	130
47. CUSTUI UI Record definition	131
48. CSTCNV-MAIN processing logic	133
49. CSTXP Web Transaction program structure	134
50. CUSTUI UI Record definition	136
51. CSTXP-MAIN processing logic.	137
52. CSTXP1 and CSTXP2 Web Transaction programs structure.	138
53. Input Customer Info Web page.	138
54. Output Customer Info Web page	139
55. CUSTUI1 UI Record definition	139
56. CUSTUI2 UI Record definition	140
57. CSTXP1-MAIN processing logic.	141
58. CSTXP2-MAIN processing logic.	141
59. CSTXB1 Web Transaction program structure	142
60. Customer Info Web page	143
61. CUSTUI_IO UI Record definition	144
62. CSTXB1-MAIN processing logic.	145
63. CSTXB2 Web Transaction program structure	145
64. Input Customer Info Web page.	146
65. Output Customer Info Web page	146
66. CUSTUI_I UI Record definition.	147
67. CUSTUI_O UI Record definition.	147
68. CUSTUI_IN UI Record definition	148
69. CSTXB2-MAIN processing logic.	149
70. CSTCNS Web Transaction program structure	151
71. User Name inquiry Web page.	151
72. Customer Info Web page without the Update button	152
73. Customer Info Web page with the Update button.	152
74. CUSTUIS UI Record definition	153
75. CSTIDUS-MAIN processing logic (1)	154
76. CSTCNS-MAIN processing logic (1).	155
77. CSTXPS Web Transaction program structure	156
78. CSTIDUS-MAIN processing logic (2)	156
79. CSTXPS-MAIN processing logic.	158
80. CSTXBS Web Transaction program structure	159
81. CSTIDUS-MAIN processing logic (3)	159
82. CUSTUI_IOS UI Record definition	160
83. CSTXBS-MAIN processing logic.	162

84. CSTIDUS-MAIN processing logic (4)	163
85. CSTCNS2-MAIN processing logic	165
86. CSTIDUS-MAIN processing logic (5)	166
87. CSTXPS2-MAIN processing logic.	167
88. CSTIDUS-MAIN processing logic (6)	169
89. CUSTUI_IOS2 UI Record definition	170
90. CSTXBS2-MAIN processing logic.	172
91. Program link settings for BANKID.	180
92. Demonstration system: Web Transactions and UI Records	181
93. Demonstration system: transfer control	182
94. Demonstration system: transfer processing and data management.	183
95. FRST_FRM_UI_RECORD defined form properties	187
96. CONV_UI_RECORD defined form properties	188
97. CONV_UI_RECORD program link definition (1).	190
98. CONV_UI_RECORD program link definition (2).	191
99. Enhancing JSP rendering: Before.	196
100. Enhancing JSP rendering: After.	197
101. Implementing help — JavaScript	199
102. Implementing help — JSP tags	199
103. Customized JSP with help button and help window.	199
104. Default Customer Info JSP.	201
105. Default code for HTML table	202
106. Modified code for HTML table	203
107. JSP with TABLE modifications.	203
108. Highlighting the FORM element to move	204
109. JSP source after modification.	204
110. Modified JSP rendered in browser.	205
111. TABLE code with rows and cells	207
112. TABLE display after rows and cells	207
113. FONT attributes given to FORM element.	209
114. FORM field with modified FONT properties	210
115. Color information for FORM error message	210
116. WebSphere Studio, VisualAge for Java, and WebSphere Application Server.	211
117. Default publishing stages and servers after WebSphere Studio installation	212
118. Default properties for target server.	213
119. Development with WebSphere Studio, VisualAge for Java, and WebSphere Application Server.	214
120. Creating the StyleDemo project in WebSphere Studio	216
121. Publishing targets properties for project.	217
122. Publishing targets properties for VAGenFiles folder	218
123. Publishing targets properties for VAGenFiles folder	219
124. Publishing targets properties for VAGenFiles folder	220
125. Publishing targets properties for VAGenFiles folder	221

126.Base rendering of CSTUI.jsp in Navigator (l) and Internet Explorer (r)	222
127.Rendering with default Master.CSS in Navigator (l) and Internet Explorer (r) . . .	223
128.Master.css font modification	224
129.Rendering with modified fonts in Navigator (l) and Internet Explorer (r)	224
130.Master.css font modification	225
131.Rendering with modified fonts in Navigator (l) and Internet Explorer (r)	225
132.Editing CUSTUI.jsp in WebSphere Page Designer	226
133.Opening WebSphere Studio and creating a new project	229
134.Selecting the Corporate1 template.	229
135.Web site from Corporate1 template in WebSphere Studio	230
136.Entry page for Corporate1 template Web site	231
137.WebSphere Page Designer WYSIWYG view for top.html	231
138.WebSphere Page Designer HTML view for top.html	232
139.WebSphere Page Designer Frame HTML source view for index.html.	233
140.Enhanced entry page for Corporate1 template Web site	234
141.Source for HTML navigation to Gateway Servlet	236
142.Runtime view of HTML navigation to Gateway Servlet	236
143.Gateway Servlet as target of HTML navigation	237
144.Customer Info Web Transaction running in HTML content frame	238
145.Source for HTML navigation to defined Web Transactions	239
146.Direct invocation of Customer Info Web Transaction in content frame	239
147.Response failure when using index.html as Gateway Servlet entry page	240
148.Recursive failure when using index.jsp as Gateway Servlet entry page	241
149.Correcting the recursive loading for index.jsp as Gateway Servlet entry page .	242
150.Dynamic Gateway Servlet resolution in Vagen1EntryPage.jsp	243
151.Revised index.jsp for dynamic Gateway Servlet resolution	244
152.Revised top.jsp for dynamic Gateway Servlet resolution	244
153.Native NT and UDB scenario	247
154.Native NT and UDB implementation	247
155.NT CICS and UDB scenario	250
156.NT CICS and UDB implementation	251
157.CICS/ESA and DB2 scenario	254
158.CICS/ESA and DB2 implementation	254
159.Attaching DB2 CAE to a database on a remote database server — part 1	257
160.Attaching DB2 CAE to a database on a remote database server — part 2	258
161.Configuring FTP — part 1	260
162.Configuring FTP — part 2	260
163.Configuring FTP — part 3	261
164.Configuring FTP — part 4	262
165.Configuring FTP — part 5	263
166.Overriding CSOGW.properties file entries	267
167.TCP/IP catcher program start command	269
168.CSOGW.properties file entries: Windows NT system	270

169.Attaching CICS to DB2 — part1	273
170.Attaching CICS to DB2 — part2	274
171.Adding a listener to CICS	276
172.CSOGW.properties file entries: CICS NT system	279
173.Add a node	280
174.Basic node details	281
175.Advanced node details	282
176.DLU requester details	283
177.Configure devices	284
178.Defining a LAN device — basic	285
179.Defining a LAN device — activation	286
180.Defining a LAN device — performance	287
181.Configure new connection	288
182.Defining a LAN connection — basic	289
183.Defining a LAN connection — advanced	290
184.Defining a LAN connection — Adjacent Node	291
185.Configure partner LU 6.2	292
186.Defining a Partner LU — basic	293
187.Defining a partner LU — advanced	293
188.Configure mode	294
189.Defining a mode — basic	295
190.Defining a mode — advanced	296
191.Starting the link	297
192.Start the node	298
193.Initialize the session	299
194.CSOGW.properties file entries: Windows NT system	300
195.WebSphere Administration Console	303
196.Application Server configuration: starting the task	304
197.Application Server configuration: defining parameters — part 1	305
198.Application Server configuration: defining parameters — part 2	306
199.Application Server configuration: choosing a node	307
200.Application Server configuration: virtual host	307
201.Application Server configuration: servlet engine	308
202.Application Server configuration: defining Web application	308
203.Application Server configuration: Web application properties	309
204.Application Server configuration: system servlets	310
205.Application Server configuration: defining error page	311
206.Application Server configuration: defining Gateway Servlet	312
207.Gateway Servlet properties — part 1	313
208.Gateway Servlet properties — part 2	313
209.Gateway Servlet properties — part 3	314
210.Gateway Servlet properties — using an external parameter file	316
211.Gateway Servlet properties — external parameter file contents	316

212.Starting WebSphere Application Server	319
213.CICS client settings for Windows NT-based CICS system	321
214.CICS client settings for MVS-based CICS/ESA system	321
215.Gateway Servlet problems when CICS support is not loaded	326
216.Gateway Servlet problems when CICS support is loaded	327
217.Gateway Servlet definition and initialization	329
218.Defining JSP 1.0 support	330
219.JSP execution monitor	333
220.Generation command: Windows NT system	337
221.Generation command: CICS NT system	337
222.Adding a new program to CICS	338
223.Generation command: CICS/ESA system	339
224.Gateway Servlet entry page definition	343
225.Gateway Servlet Logon page	345
226.Gateway Servlet Entry page	346
227.Gateway Servlet runtime system processing	347
228.Windows NT runtime system processing	350
229.CICS runtime system processing	351
230.Default Gateway Servlet logon page	353
231.Default entry page	354
232.No userid or password	355
233.Invalid userid or password	356
234.Default error page	358

Tables

1. Web Transaction state saving options	87
2. Comparison of Web Transaction program types.	89
3. Data management by UI Type	100
4. UI Record editing options	102
5. Web Transaction architecture issues	103
6. State management in the demonstration system	184
7. UI Record data item edits in demonstration system	185
8. CSOGW property file: application definition	264
9. CSOGW property file: serverLinkage definition	265
10. Valid locations and serverids for a given comtype.	266
11. Initialization parameters for the Gateway servlet	315

Preface

This redbook explains how VisualAge Generator V4 can be used to implement Web-based transaction systems that access enterprise server platforms and resources.

This redbook demonstrates the new Web Transaction capabilities provided in VisualAge Generator V4 for the implementation of a Web-based enterprise access system. The techniques used in this demonstration system will help you design, build, and implement the Web Transaction programs required to support the real-world business processing necessary for such systems. Software configuration examples, working VisualAge Generator Web Transaction systems, and advanced design discussions are included.

After reading the redbook you will fully understand how to use VisualAge Generator to build and implement an IBM WebSphere Application Server-based system that accesses enterprise transactions and data.

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 San Jose Center.

Pat McCarthy is a Consulting Application Development Specialist at the International Technical Support Organization, San Jose Center. He writes extensively and teaches IBM classes worldwide on all areas of application development, with a specific emphasis on VisualAge Generator and related technologies. Before joining the ITSO in 1990, Pat worked in an IBM internal information systems organization in Poughkeepsie, New York, as a programmer, database administrator, and development center leader. Pat received a B.S. in Business Administration from Indiana University of Pennsylvania (Indiana, PA) and an M.S. in Computer Science from Marist College (Poughkeepsie, NY).

Jane Glover is a software consultant in the UK. She works for Bloomsbury Software, a company that specializes in training and consultancy in the VisualAge product family and e-business technologies such as WebSphere and JavaScript. She has 8 years of experience with VisualAge Generator and its predecessor CSP, much of it with John Lewis, a well known UK retail company, where she worked in its Technical Support section. She holds an M.Eng. degree in Chemical Engineering from Christ's College, Cambridge.

Eric Simone is the founder and President of Compete Incorporated, a VisualAge and WebSphere consulting company. Compete Incorporated was recently purchased by Perficient, where Eric is now the Senior Managing Director of Emerging Practices. He has 11 years of development, design, and sales experience in VisualAge Generator. Before founding Compete, Eric worked for IBM for 5 years as an Application Development Specialist and a Systems Engineer. Eric received a B.S. in Computer Science from Purdue University (West Lafayette, IN).

Tom Sanderson is a Consultant with Perficient (previously known as Compete Incorporated), a VisualAge Generator and VisualAge for Java consulting company. He is a recent graduate of Cedarville College (Cedarville, OH), where he received a B.A. in Multimedia Technologies. Tom specializes in Web site design and development for Web applications.

Alvaro Torres has been working for IBM in Spain as an IT Specialist since 1996. He has concentrated on System Integration and Application Development projects for Telco & Media customers. He is an AIX Certified Specialist, and his areas of expertise include AIX, RS/6000, SP/2, and client/server development.

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

Phil Wakelin
International Technical Support Organization, San Jose Center

Tim Wilson
IBM Raleigh

Kristine Heaton
IBM Raleigh

Rob Swofford
IBM Raleigh

Guy Slade
IBM Raleigh

Stefano Sergi
IBM Raleigh

Denise Hendriks
Perficient (previously known as Compete Incorporated)

Jennifer Pearcey
IBM South Africa

Teresa Smit
IBM Raleigh

Chris McCarthy
IBM Santa Teresa

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 “IBM Redbooks review” on page 393 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

Part 1. VisualAge Generator Web Transaction overview

2 Building Enterprise Web Transactions using VisualAge Generator JavaBeans and JSPs

Chapter 1. Introduction to VisualAge Generator Web Transactions

The Web Transaction support provided in VisualAge Generator V4 is introduced in this chapter. VisualAge Generator V4 introduces both functional enhancements and new development and runtime platforms (see Figure 1).

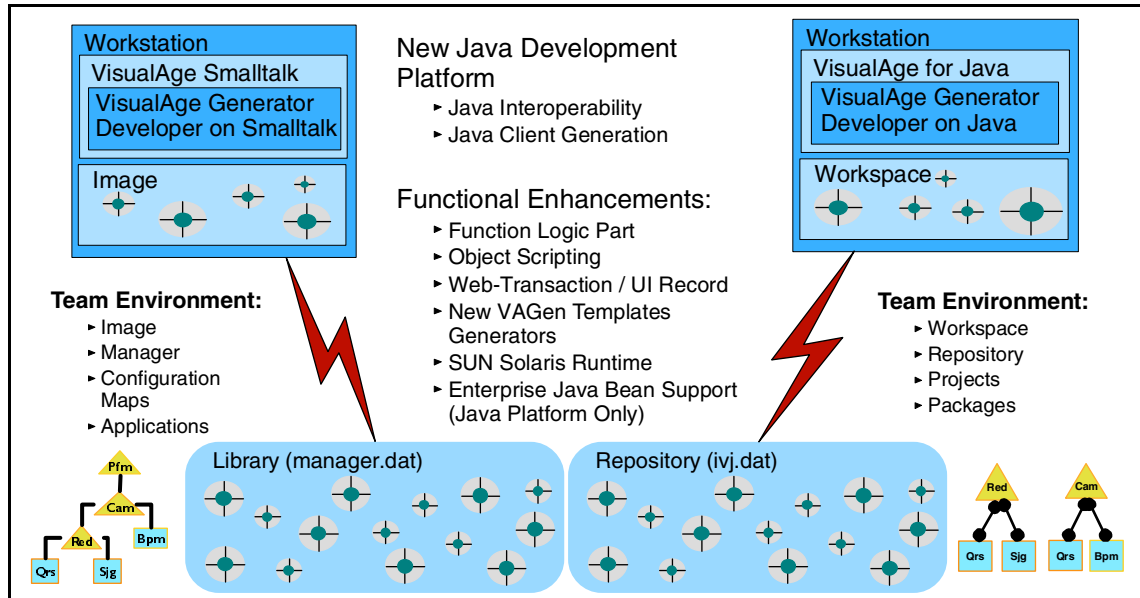


Figure 1. VisualAge Generator V4 overview

With VisualAge Generator V4 you get:

- VisualAge for Java Interoperability — a new development platform
- Functions — true reusable coding capability
- Object Scripting — tight integration between 4GL and object logic
- VisualAge Generator Templates Enhancements — a better RAD engine
- SUN Solaris Runtime Support — a new runtime platform
- Development Support for Web Transactions — Web systems made easy
- Enterprise Java Bean Support — integrated support for advanced Java-based distributed object systems

In this redbook we explore the new Web Transaction development capability provided in VisualAge Generator V4 (Java or Smalltalk based development).

Additional details on VisualAge Generator V4 are available in the *VisualAge Generator V4 System Development Guide*, SG24-5467. Only the new Web Transaction programming model will be discussed in this redbook.

1.1 Rapid Web Transaction application development concepts

In this section we discuss how the power of the new 4GL-based Web Transaction programming model and use of VisualAge Generator generation technology reduces the complexity and skill requirements, and therefore the price of entry, for Web-based transactional system development.

1.1.1 Problem statement

Basic (native) development for a Web-based transaction system requires a mix of skills, all of which are hard to master and typically require the coordinated effort of multiple individuals (see Figure 2).

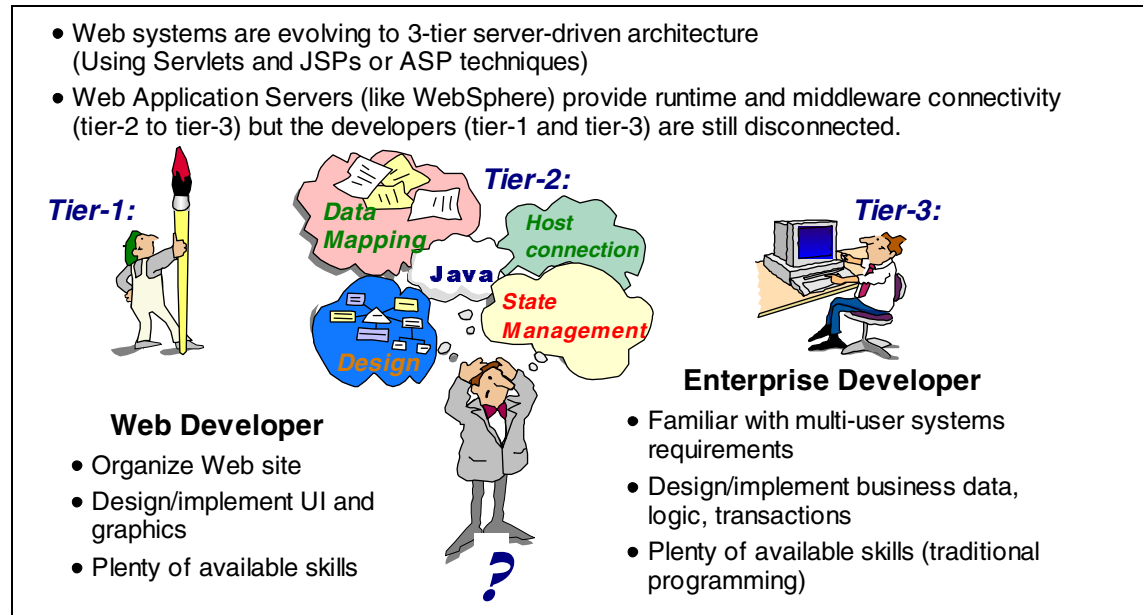


Figure 2. Problem: Implementation and communication (tier-1, tier-2, and tier-3)

It is widely recognized that Web systems are now evolving towards a 3-tier architecture, with server-driven content visualized with dynamic Web pages driven by technologies such as Microsoft ASPs or those used by the rest of the industry: JSPs and Servlets.

Web Application Servers, such as IBM WebSphere Application Server, are key to being able to implement such systems, providing the environment that supports the execution of these special programs and the connectivity to other enterprise IT assets; however, the development of tier-2 is the real challenge!

While the tier-1 developer (the Webmaster) now has a widespread skill in a job that is fairly well understood, and while tier-3 development is easily accomplished with traditional business application programmers, tier-2 requires dealing with the most difficult issues, such as:

- Designing small modular servlets that are coordinated by a Java server driver in charge of managing state (data) between user think time screens, which of course requires Java skills to implement
- Understanding how to map data between Java objects and flat data structures
- Understanding API for host connectivity

These skills are rarely found in one person, and this may be the real hurdle in achieving rapid and successful development in a Web-server environment.

1.1.2 Architectural analysis

Figure 3 shows that if we think carefully about the architecture of a mainframe attached terminal pseudo-conversational system and a servlet-based Web system, we can see that they are similar:

- The user view sends a request to the runtime domain.
- The runtime domain recovers any stored information that may be required to continue the conversation.
- Edit and business logic is used to react to the request and formulate a response.
- Information required to continue the conversation is stored.
- The response is returned to the user view.

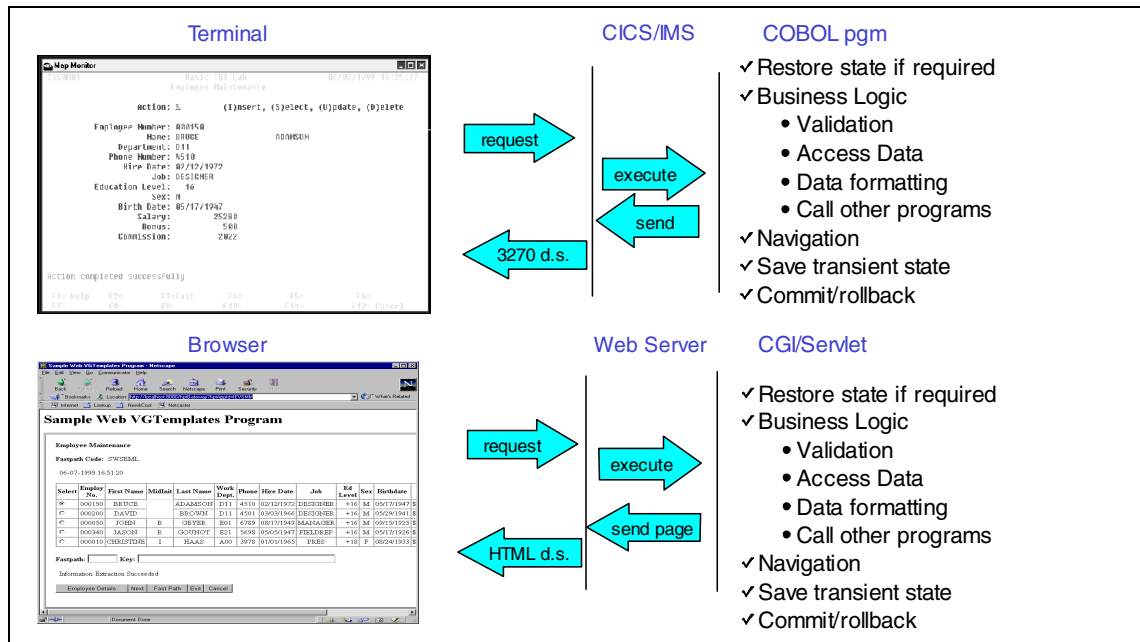


Figure 3. Architectural affinity: 3270 and Web systems

Web systems work very much the same way: instead of a 3270 data stream being sent to terminals, an HTML data stream is sent to browsers; instead of CICS or IMS middleware, a Web server software is in charge of running the program and interacting with the display device; and instead of mainframe COBOL programs, we have servlets, which essentially must do similar things.

A Web environment differs from traditional 3270 in that:

- The volume of users logging on to the system is hard to predict (in the case of the Internet).
- HTML features such as ANCHOR tags give us the option of opening several browser windows at once; so we are not restricted to a modal environment.
- It is quite likely that the users will not leave the system cleanly; they will probably not logoff, but will just close down their browsers or go to another URL.
- It is also likely, in the case of the Internet, that your system may be invoked from public shared machines.

These issues must be considered during the implementation and operation of a Web-based transaction system.

Pseudo-conversational execution is predicated on the fact that a program retrieves data, formats it, and sends it to the user; then goes away, saves the state of the program at that moment, and releases all resources. Once the user finishes interacting with the screen information, they submit another request (pressing a keyboard key) and the program gets control again, restores state, processes input, and so on, eventually interacting with the user sending out another screen; and the cycle repeats.

VisualAge Generator already has a lot of core technology and competence in handling code generation and runtime services to support this type of Web system. Therefore, we have exploited this know-how to implement what we call the WebSphere RAD facility: a simple innovative way of developing Web systems.

1.1.3 Solution

A new model of programming has been introduced with VisualAge Generator V4. This new programming model includes:

- A new main program type — Web Transaction
- A new record type — User Interface Record (UI Record).

These new programming constructs support 4GL-based development of business systems that use a Web-browser for the user interface and generated runtime programs for business logic and database processing.

The Web Transaction programming model is related to the model used for TUI programming in VisualAge Generator. Similar component roles exist, and the method of interacting with the end user is based on a similar use of either the 4GL CONVERSE processing option or XFER single segment transfer technique (see Figure 4).

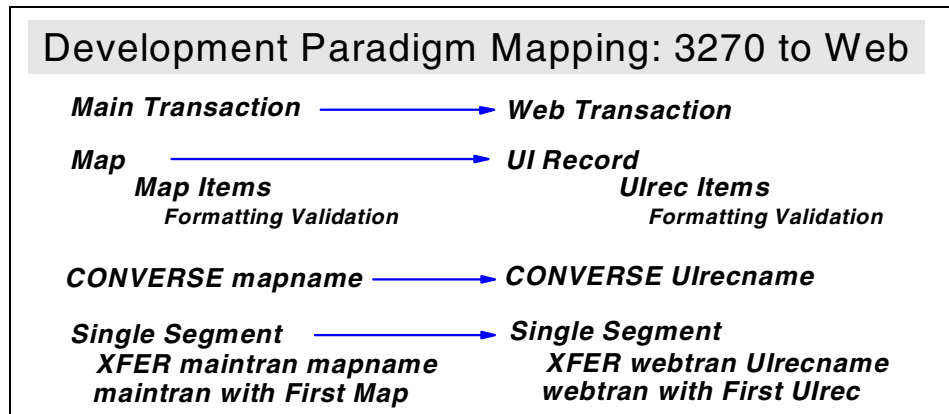


Figure 4. Web Transaction definition

This implementation approach for the Web Transaction programming model is a derivative of the existing VisualAge Generator approach for support of CICS pseudo-conversational programming.

This approach is based on recognizing that programs that are driven from the server (such as a CICS conversation or a Web application) have the same basic request/response form of communication. The inherent programming problems are very similar (accept input, remember the conversation state, process request) no matter what the technology.

This means that the CICS pseudo-conversational programming problems (state management between screens, transaction control, program flow) are the same as those experienced by Web application programmers.

Using a Web Transaction program, programmers can define a Web system as a single-threaded program which interacts with users by sending business data directly to a browser through the use of the UI Record data definition and either the CONVERSE or XFER language verb.

A traditionally skilled Enterprise Developer can easily master this technique and define a Web Transaction program that can have as many User Interactions and as much back-end processing as desired. This is far easier to conceptualize and design than conceiving separate servlets and a dispatcher program in charge of figuring out what to invoke next.

VisualAge Generator V4 provides definition, testing, generation, and runtime support for the new Web Transaction programming model (see Figure 5).

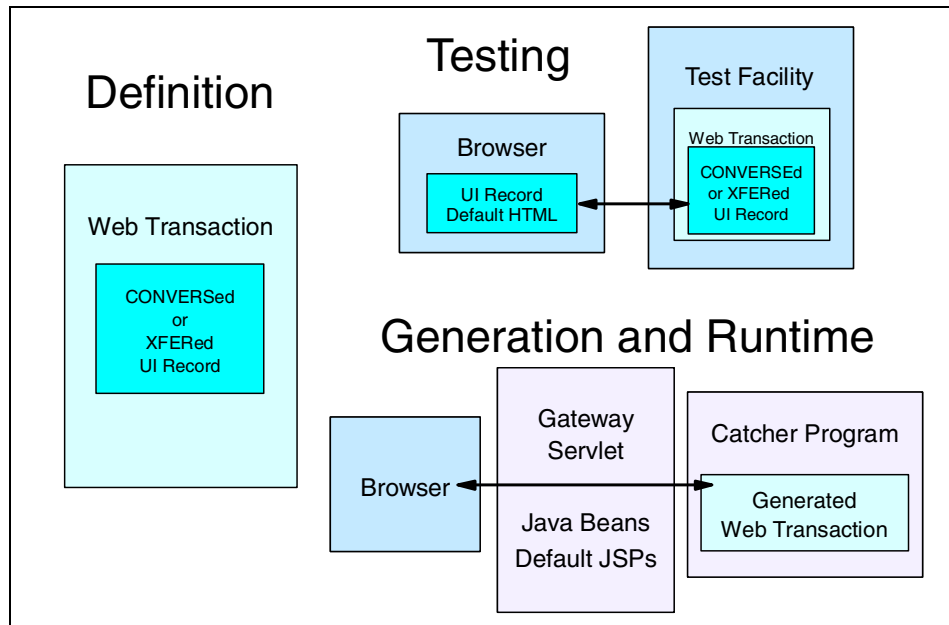


Figure 5. Web Transaction implementation overview

As shown in Figure 5, the support for this new programming model implemented in VisualAge Generator V4 permits:

Definition

- The definition of UI Records that control the default formatting of data in the Browser.

Input validation controls and logic are defined as part of the UI Record.

- The implementation of Web Transaction main programs, using 4GL programming techniques, that interact with end users in a Web browser.

These 4GL-based programs implement user interface, business logic, and database processing. This processing can be implemented in either a series of main programs or a set of main and called batch (server) programs.

Testing

- Testing of these new programs using the existing VisualAge Generator test facility and a Web browser.

A default view for the UI Record is used in the browser. This communicates directly with the main program being tested.

Generation and runtime

- UI Record generation creates JavaBeans and default Java Server Pages (JSPs).

These materials can be used by Web-based user interface developers (who use HTML, JavaScript, and so on) to customized the data access and how the browser display is rendered. The JavaBeans have methods that allow attribute access and a single action that puts all the inputs through the edits defined by the programmer.

- The program is generated using the current VisualAge Generator pseudo-conversational model for runtime execution.
CICS pseudo-conversational problems of state, transaction control, program flow, and so on, are similar to Web problems.
- Generation technology is matched with runtime components to automate the complexities of browser input processing and runtime communication.

The Java Gateway Servlet provided by VisualAge Generator uses the JavaBeans and JSPs and calls the Web Transaction program when required.

When required (depends on program structure), the implementation of state management is automated by the generated transaction program and the Gateway Servlet provided as part of VisualAge Generator runtime support. The Gateway Servlet drives the system by processing browser requests using the generated JavaBeans, formatting calls to tier-3 servers, and rendering the browser display using the associated JSP (default generated by VisualAge Generator).

The programmer thinks of the sequence of user interactions and back-end processing as a single logical unit (very much the way Enterprise Developers are used to developing programs today!). So, using VisualAge Generator Web Transaction support, all the tier-2 complexity is simplified, and the Enterprise Developer will program tier-2 without even being aware that he is doing so! (see Figure 6).

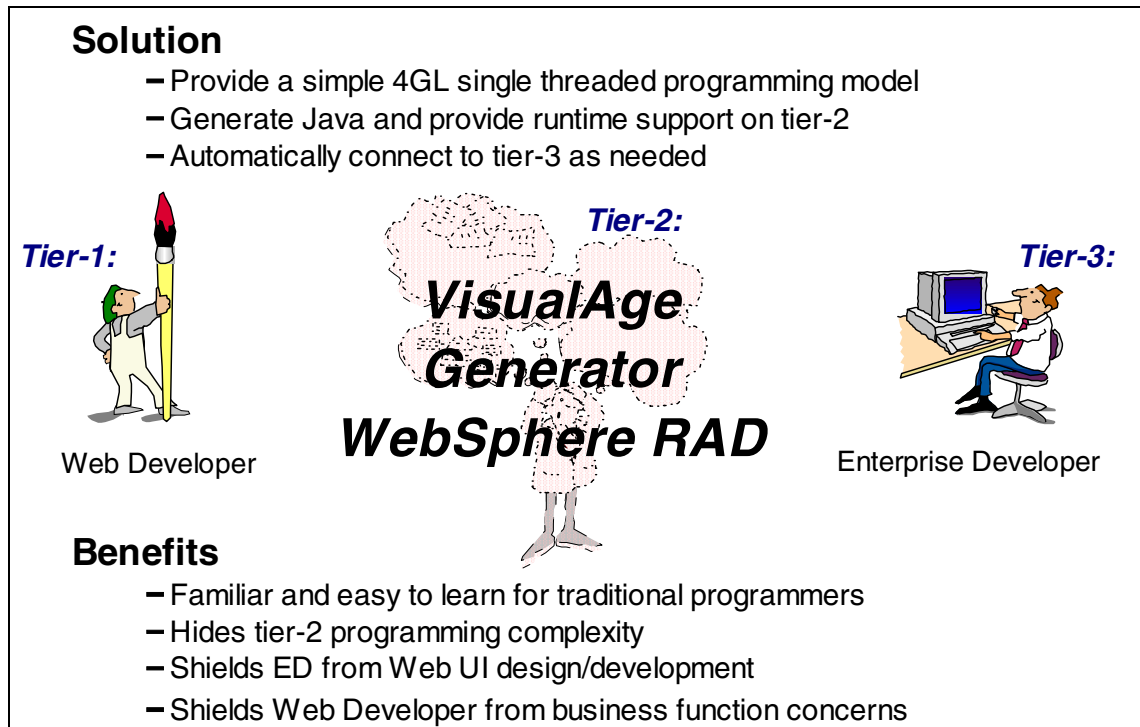


Figure 6. Web Transaction support for automated tier-to-tier communication

The output of the programming effort of the Enterprise Developer is translated by VisualAge Generator into JavaBeans and JSPs that are very familiar to the Web Developer, thus achieving an easy way of bridging the two development skills and realizing optimal separation of concerns: the Enterprise Developer does not have to be a Web expert, and the Web Developer need not be concerned with business function, but can focus on the design of the user interface as well as the appropriate use of the Gateway Servlet and user interface components supplied by the Enterprise Developer.

Just as with VisualAge Generator's support for client/server programming, the difficult task of connecting the client (Web browser) and the server (generated version of main program) is automated.

1.2 Web server-based transaction system implementation options

There are a variety of techniques available for the implementation of Web server-based transaction systems. In this section we will review two as a way of describing how a VisualAge Generator Web Transaction is implemented.

1.2.1 Native Java servlet and JSP programming

Typically a Web application programmer must string a series of servlets or CGI programs together to render the complete *Web-based business application*. The servlet will receive and process the HTTP request and send back an HTTP response.

The Web programmer is also likely to save conversation state for the user on the Web server during an ongoing transaction with that user over potentially many servlets (see Figure 7).

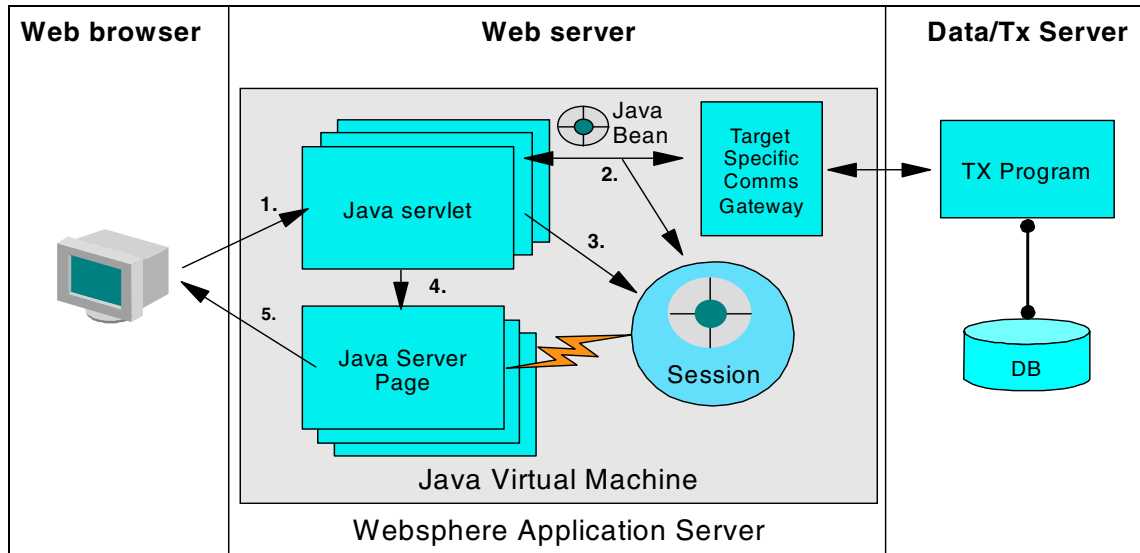


Figure 7. Anatomy of a Web system

The processing steps in Figure 7 include:

1. The browser invokes a servlet from an HTML FORM.
2. The servlet reads the data from the FORM INPUT fields using its `HttpServletRequest` object. It may create, or *instantiate*, new Java objects which hold this user data.

The servlet may also initiate some enterprise access, for example, look up information on a database. The results of the enterprise access may cause the instantiation of other objects.

3. The servlet now needs to create a new session object for the user, or else reference an existing one. There is a single method, `getSession()` of the `HttpServletRequest` to do this.

Once we have reference to a session object we can store references to objects for that user in it (typically the objects we referred to in step 2).

4. The servlet invokes a JSP, passing the `HttpServletRequest` object.
5. JSP syntax will allow you to effectively run a `getSession()` against the request and also to then query the session object to locate user objects, so they may be displayed to the user in the final HTML stream sent to the browser.

1.2.2 VisualAge Generator Web Transaction programming

The new model of programming provided with VisualAge Generator V4 supports 4GL-based development of business systems that use a Web-browser for the user interface and generated runtime programs for business logic and database processing.

To better understand how this new facility works, let's walk through the development steps necessary to define and test the system.

First the programmer defines the data structures that represent business information to be shown to the user on a browser. We call this a UI Record, and it contains data items, their validation and formatting rules, default labels, and a type that indicates whether the item represents output data, input/output data, or user actions.

Then the programmer defines a program (of type Web Transaction) which contains 4GL logic that fills UI Record items and sends the UI Record to the browser (CONVERSE). The same program can also contain logic to process the user actions and take new business logic steps (maybe gathering more business data, formatting it and sending out another UI Record), and so on.

When the program is ready to test, the VisualAge Generator Test Facility is able to animate the 4GL source definitions and simulate the program execution, including formatting data into HTML and invoking a browser to display it! All this is done in one seamless interactive facility without the need to install and deploy any Web server infrastructure, or to compile and deploy servlets or 3rd-tier server programs!

When generated, the VisualAge Generator Web Transaction program can be implemented in a Web server runtime domain such as IBM WebSphere Application Server. The actual runtime processing is similar to that used for a native Java implementation (see Figure 8), but this complexity is hidden from the development programmer.

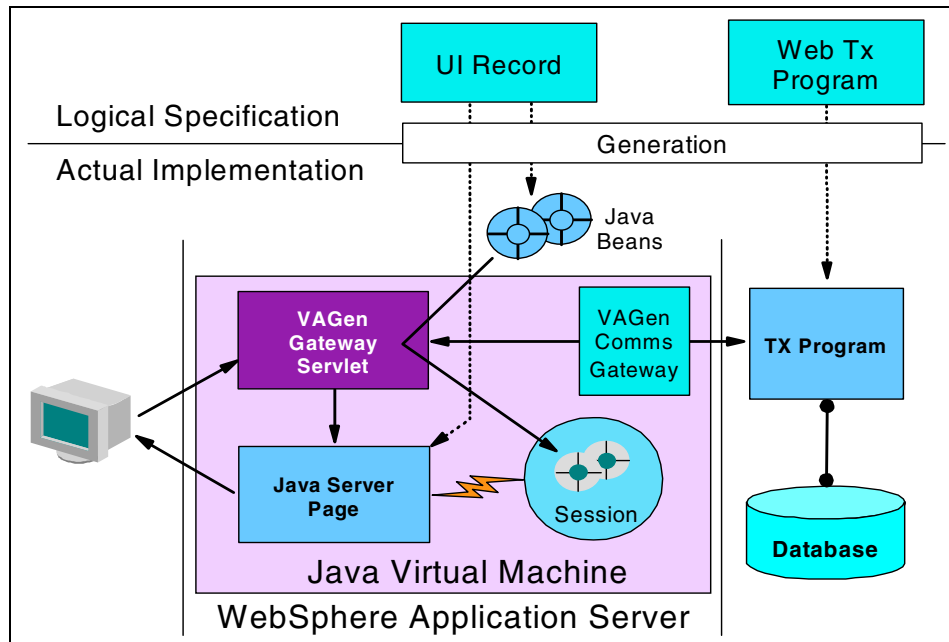


Figure 8. Web Transaction runtime

Runtime involves several tiers:

- A Web browser
- A Web server with servlet support, perhaps using the IBM WebSphere Application Server platform
- A VisualAge Generator server runtime environment

The Gateway Servlet is included with the VisualAge Generator runtime environment installed on the Web server. The Web Transaction programs could be implemented on the Web server or another remote platform.

1.2.3 Supported runtime configurations

Multiple VisualAge Generator Web Transaction program runtime configurations are supported when using IBM WebSphere Application Server, CICS, and DB2 (see Figure 9).

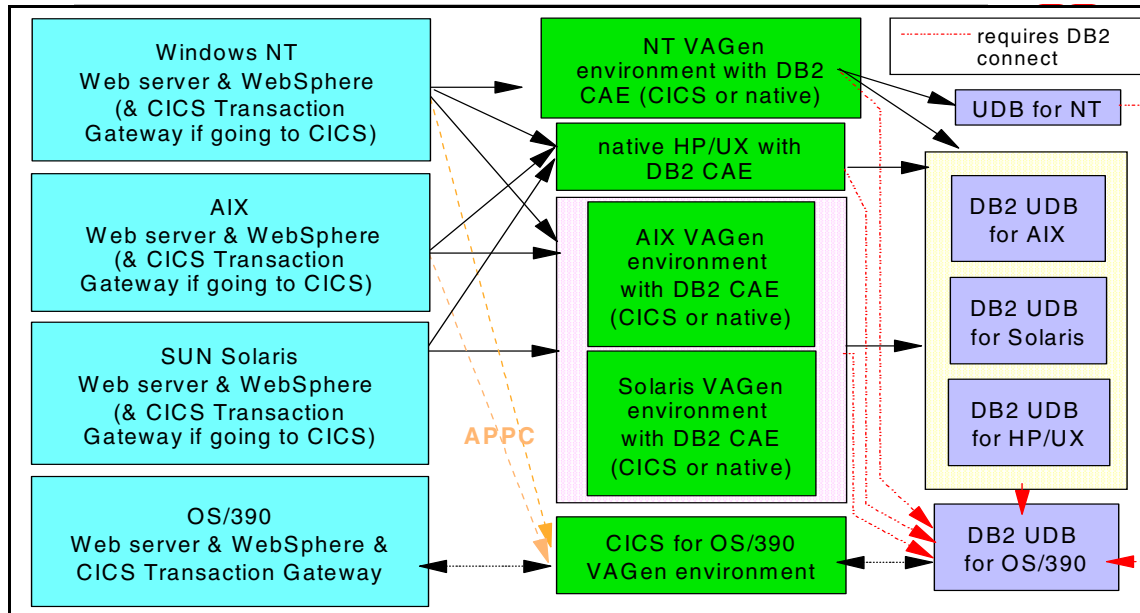


Figure 9. Web Transaction runtime configurations using WebSphere Application Server

The use of database products other than DB2 is possible. VisualAge Generator allows the generation of native Oracle access and access to other database platforms is available using ODBC.

Notes:

- While the IMS/TM transaction platform option is not illustrated in Figure 9, VisualAge Generator Web Transactions can be run in IMS/TM.
- The use of WebSphere on an MVS platform requires support for JSDK 2.0 and other product changes. This support, and support for converting the generated JSPs from the 1.0 to .91 syntax level is expected in a later release of VisualAge Generator.
- The OS/400 platform is not currently supported as a runtime platform for Web Transactions (expected in a later release of VisualAge Generator). You could run the Web Transactions on a Windows NT platform and have them call server programs that run (and access data) on the AS/400.

Chapter 12, “Runtime environment scenario implementation” on page 247 reviews the implementation of selected runtime configurations.

1.2.4 Summary

VisualAge Generator's approach to solving this problem is to:

- Use a UI Record, which contains data and control information, to communicate with the Web browser.
- Allow Web Transactions to invoke many of the field edits, such as input required, which we expect to have in traditional 3270 programs to speed program development.
- Use a similar model of programming for Web Transactions as has already been used for pseudo-conversational programs: a CONVERSE or Single Segment XFER programming model.

Note: There are significant performance and design implications to consider between these models. Both the models and implications are discussed in detail in Chapter 5, "Web Transaction design concepts and considerations" on page 85).

- Support testing of the Web Transaction program using a Web browser, a simulation of the runtime environment provided by VisualAge Generator, and the 4GL Test Facility.
- Generate JavaBeans, Java Server Pages (JSPs), a segmented implementation of the Web Transaction, and other materials to support implementation in a Web browser and target runtime system.
- Provide new runtime components, in the form of a Gateway Servlet and catcher program in the target runtime environment, combined with additional generation technology to link the Web Transaction, at the CONVERSE/ XFER point, to the Web browser.
- Allow Web Transactions to be implemented on a variety of platforms (Windows NT, AS/400, AIX, HP-UX, SUN Solaris, VSE, MVS) with support for both native operation and the use of transaction processing engines such as CICS and IMS/DC. (MVS support is expected in a fixpak, AS/400 support in a future version of VisualAge Generator).

The default system look-and-feel, as implemented by the generated JavaBeans and JSPs, can be enhanced by Web client-side specialists using tools such as IBM WebSphere Studio. This separates user interface design from the implementation of business logic and database access while providing a solid foundation for communication and cooperation between the two development domains or communities.

Chapter 2. Web Transaction system implementation

This chapter steps through the process of Web Transaction definition, testing, generation, and runtime implementation (see Figure 10).

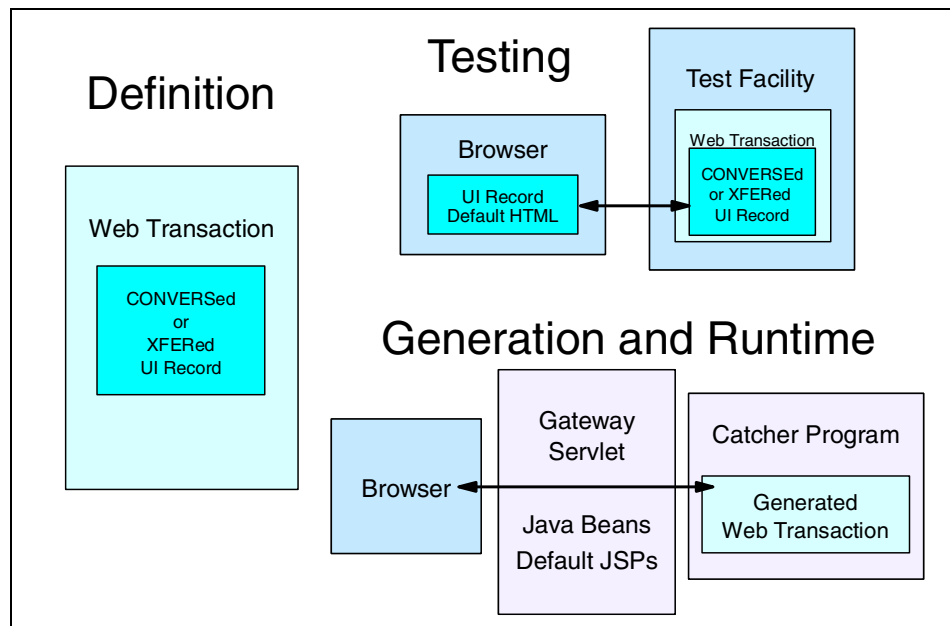


Figure 10. Implementation process for VisualAge Generator V4 Web transactions

This guided tour through the development and implementation process will help you understand these key points:

- Mapping of UI Record definitions to the HTML seen in a browser.
- Source level testing support (runtime emulation) as implemented through interaction between the browser and Web Transaction code running in the Test Facility.
- Generation processing that transforms the UI Record and Web Transaction definitions into the appropriate components (JavaBeans, JSPs, and executable programs).
- Configuration and interaction of the generated components and VisualAge Generator supplied components in a IBM WebSphere Application Server runtime environment.

2.1 Web Transaction development

An overview of the development process for a Web Transaction based system is shown in Figure 11.

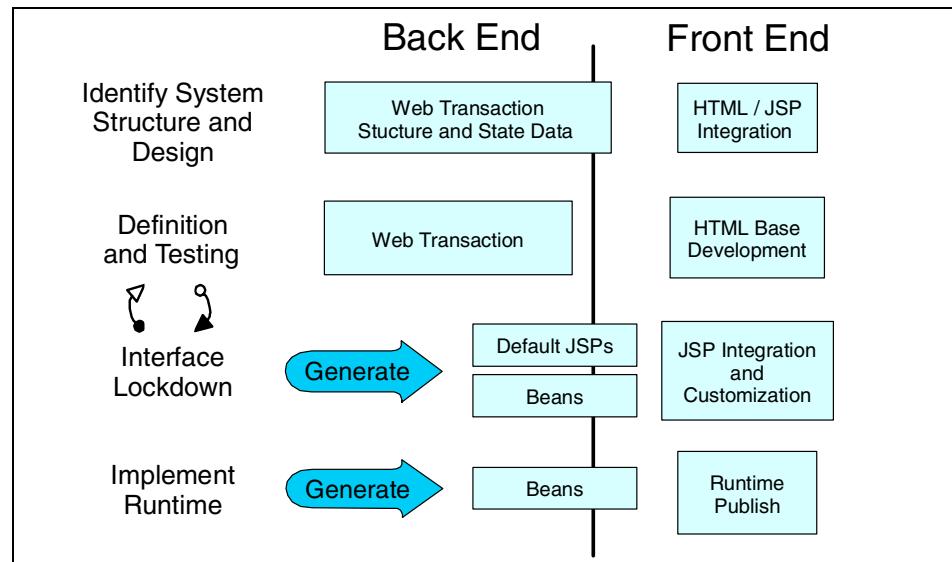


Figure 11. Process life cycle

From a Web Transaction viewpoint, this process includes:

- Define UI Record with data items required for the end user interaction:
 - Data item UI type (input, output, submit, and so on)
 - Data item edits (validation, input required, and so on)
- Define Program as Web Transaction:
 - Write program logic with references to UI Records using CONVERSE or XFER (single segment) approach.
 - Implement required file/database access in Web Transaction or called server programs (a better approach for reusability).
- Animate through ITF:
 - This provides simulation of business logic execution.
 - Test facility dynamically builds HTML for UI Record.
 - The browser submits return control to the test facility to handle browser submit request and link back to 4GL debugger.

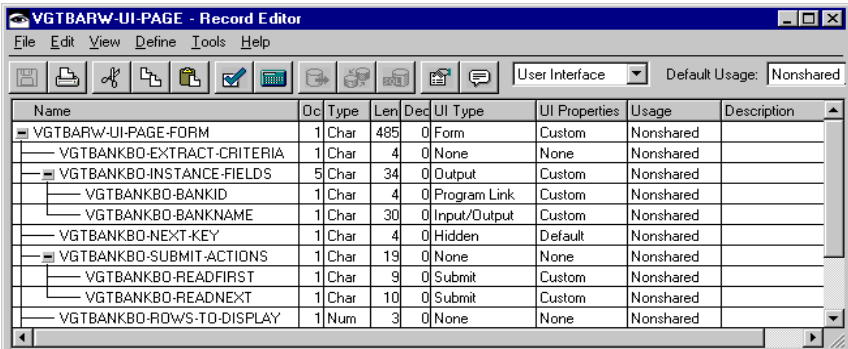
- Generation of runtime components for initial use in the front end development process:
 - Default JSPs that implement raw user interface function
 - JavaBeans that are used by Gateway Servlet to support interaction with runtime Web Transaction program

The front end development process includes both independent activities (Web site design and implementation) and the task of integrating the runtime Web Transactions, either by customizing the generated default JSPs, or by building customized interfaces that directly interact with the Gateway Servlet and the JavaBeans generated from the UI Record.

2.1.1 UI Record definition

A UI Record definition looks just about like any other record when viewed in the record editor. The UI Record includes a set of data items (shared or nonshared), with support for levels (03, 05, and so on), general characteristics (type, length), and other properties specific to the record type. See Figure 12.

- Links user interface and business processing domain
- UI Record is an organized set of data items and properties:
 - Determines what data is sent to, and received from, the browser
 - Contains user action indicators (what push button was clicked)
 - Can contain transfer control information (where to go next)
 - Contains UI functions (e.g. Help, Labels, Edits)
 - Enables default HTML rendering



Name	Oc	Type	Len	Dec	UI Type	UI Properties	Usage	Description
VGTBARW-UI-PAGE-FORM	1	Char	485	0	Form	Custom	Nonshared	
VGTBANKBO-EXTRACT-CRITERIA	1	Char	4	0	None	None	Nonshared	
VGTBANKBO-INSTANCE-FIELDS	5	Char	34	0	Output	Custom	Nonshared	
VGTBANKBO-BANKID	1	Char	4	0	Program Link	Custom	Nonshared	
VGTBANKBO-BANKNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
VGTBANKBO-NEXT-KEY	1	Char	4	0	Hidden	Default	Nonshared	
VGTBANKBO-SUBMIT-ACTIONS	1	Char	19	0	None	None	Nonshared	
VGTBANKBO-READFIRST	1	Char	9	0	Submit	Custom	Nonshared	
VGTBANKBO-READNEXT	1	Char	10	0	Submit	Custom	Nonshared	
VGTBANKBO-ROWS-TO-DISPLAY	1	Num	3	0	None	None	Nonshared	

Figure 12. The UI Record

The UI Record specific properties for both the record, and the data items in the record, make the UI Record special. The UI Type and UI Properties

columns in the record definition show options that support the use of a UI Record as a browser interface definition.

These properties determine the role of each UI Record data item, how it will be managed in a browser view, and what interaction is possible in both the browser and the Web Transaction program.

2.1.2 UI Record to HTML mapping

The test facility triggers default HTML generation when a UI Record is CONVERSEd.

Note: This default HTML generation used during testing emulates the functionality that will be provided when the UI Record is generated into JavaBeans and JSPs for use at runtime (see Figure 13).

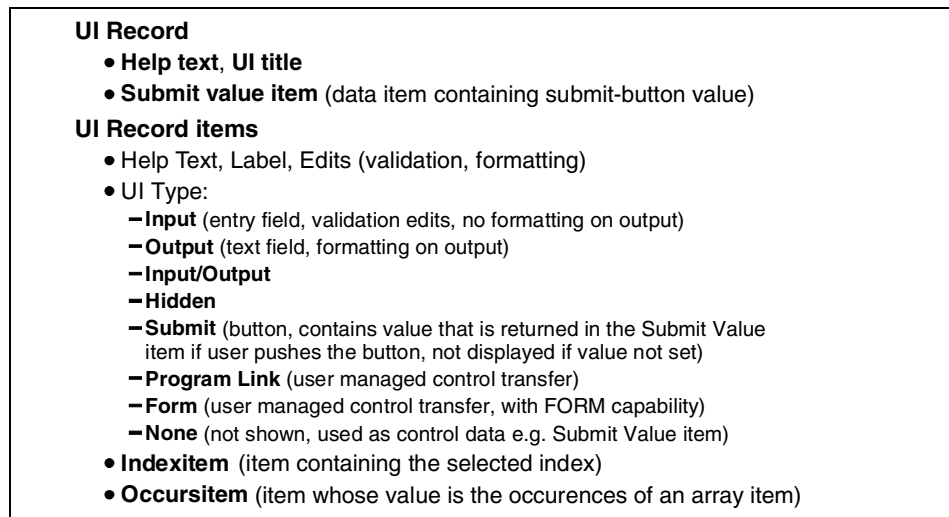


Figure 13. UI Record properties

Record attributes:

New attributes for a UI Record are:

- **Help Text** — Default help text for the entire UI Record.
- **Title** — Default title for the UI Record.
- **Submit Value Item** — Name of data item in UI Record that will contain the value of the actual submit button on the form presented to the end user.

Data item attributes:

New attributes for data item defined as part of a UI Record are:

- **User Interface Type** — The user interface type defines how the data item is used in the user interface.

The User Interface Type value, along with other data item attributes (such as occurs, substructuring, and so on), determine the content of:

- The default HTML that will be generated for the UI Record during testing (displayed in the Web browser)
- The JSPs that will be created for the UI Record during generation

The following User Interface Type values are supported:

Input	Defines that input can be entered by the end user and that edits will be run on the input data.
Output	Defines that output edits will be performed on data received from the server.
Input/Output	Specifies that INPUT and OUTPUT attributes are defined.
None	Defines that this field is not to show on the user interface and that no edits are to be defined for it. Items with this setting are typically used as control data for user-defined edits or as items such as the one defined as the Submit Value Item .
Submit	Defines an item to contain a value (or set of values if for an occurring data item) that can be received into the Submit Value Item when a user submits a form back to the server.
SubmitBypass	Same as Submit; however, when these buttons are pressed, all input edits are bypassed.
ProgramLink	Defines a data item, which implements a hyperlink in the visual display, that when selected (clicked on) by the user would implement a link to a defined program.
Hidden	Not shown in browser.
Form	Program switch with submit buttons.

- **OCCURSITEM** — Number of occurrences item

For an array item, another item in the record can be referenced as the **Number of occurrences Item**. This item must be a numeric item with no decimals. The values set into this item determine the number of occurs in the array item that actually get displayed in a list.

- **INDEXITEM** — Selected Index Item

For an array item, another item in the record can be referenced as the **Selected Index Item**. This item can be occurred. If occurred, the array will be a multiple select list. If not occurred (occurs=1), the list will be single select. The values set into this item are the indices of the elements that were selected by the user.

- **HELPTEXT** — Item Help Text

Help text defined for the item. This attribute is saved with global data items to facilitate sharing of help text between all records that use the item.

- **LABEL** — Item Label

Default Label for the item. If the item is occurred and is of type **Submit**, **SubmitBypass**, or **Link**; labels can be defined for each occurrence.

The UI type defined in the UI Record data item is used to determine the associated HTML form mapping (see Figure 14).

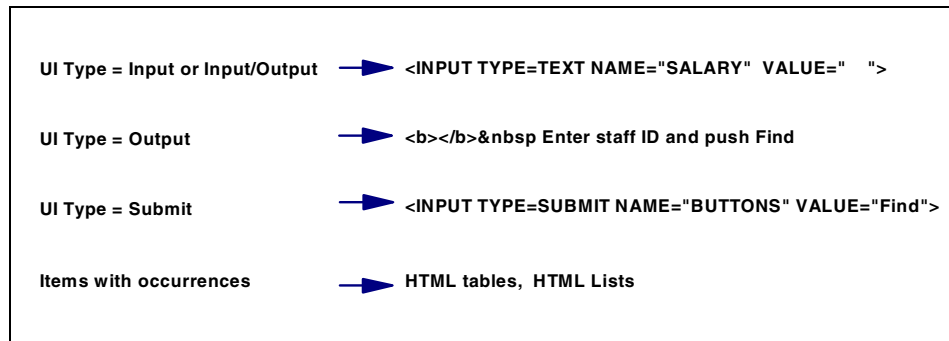


Figure 14. UI Record mapping to HTML

Rules are used to determine the appropriate HTML element to use for each data item in the UI Record.

The following rules determine if the data item should be included at all:

- Data items with a User Interface Type of *NONE* do not show on the generated page.
- Data items with a User Interface Type of *SubmitBypass* or *Submit* work the same.

Note: In the text below, the data item attribute User Interface Type (in a UI Record) will be referred to as **UIType**.

Single occurrence data item — no substructure:

UIType = Input or Input/Output

Data item length <= 80 Data item LABEL followed by Text input field.

Data item length > 80 Data item LABEL followed by Text Area (multi-line text) input field.

UIType = Output Data item LABEL followed by plain text of data item data

UIType = Submit A single submit button with the label of the item as the button text is used. All Submit buttons display at the end of the form.

UIType =ProgramLink A single hypertext link with the label of the item as the link text is used. If there is no label, then the value of the item is used. The value of the HREF attribute of the <A> HTML element will contain the program name and parameter name/value pairs as defined by Program Link properties.

Boolean

Data Item Edit The data item LABEL and a then a checkbox.

Multiple occurrence item with no substructure:

UIType = Submit Values will be displayed as a set of Submit Buttons. If a list of Labels is defined for this data item, the label values will be the text shown on the button.

UIType = ProgramLink Values will be displayed as a set of HTML hypertext links. If a list of Labels is defined for this item, the label values will be the text shown in the link. If there is no label, then the value of the item is used. The value of the HREF attribute of the <A> HTML element will contain the program name and parameter name/value pairs as defined by the Program Link properties. Since this is an occurred item, the index used to retrieve values from the value items will be the same as the given link item if those items are occurred or are defined in a substructure that is occurred.

UIType = Output	<p>Values will be displayed as a select list.</p> <p>If the Selected Index Item associated with the array is a single occurrence item, then the list is single select.</p> <p>If the Selected Index Item is a multiple occurrence item, the list is a multiple select list.</p> <p>If there is NO Selected Index Item defined, the data will be displayed as a block of text; each line being the data at each index of the array.</p>
UIType = Input or Input/Output	<p>Values will be displayed as list of text entry fields.</p> <p>If there is a Selected Index Item defined, then there will be a Select column of either Radio or Checkbox fields.</p> <p>If the Selected Index Item associated with the array is a single occurrence item, then the Select column will contain Radio buttons — only one can be selected.</p> <p>If the Selected Index Item is a multiple occurrence item, the column will contain Checkbox fields in which multiple can be selected.</p>

Multiple occurrence item with substructures:

- Always displayed as an HTML Table where each leaf data item in the substructure is a column.
- Data items in the substructure defined as **UIType = None** will not show as a column.
- Data items in the substructure defined as **UIType = Input or Input/Output** will show as a column of text entry fields.
- If there is a **Selected Index Item** and the item is UIType Input or Input/Output, then there will be a **Select** column of either Radio Buttons (single select) or Checkboxes (multiple select) depending on whether the **Selected Index Item** has occurs > 1 or not.

This allows selection of a row from an HTML Table which has no inherent selection function.

Some level of field layout can be controlled in the default HTML as follows:

- Items of **UIType = Input or Input/Output** are rendered in the order they appear in the record.
- Top level items (items that are not substructured) cause paragraph breaks when they are rendered in HTML.
- To cause fields to be flowed from left to right, a superstructure filler item (named '*') can be added to the set of items that you want to flow horizontally.

This implies that if you have a single item at the top level that substructures all the record data, the fields will all flow horizontally.

2.1.3 HTML forms in a UI Record

The HTML created for the UI Record implements the processing required based on the UI Record rules for HTML associations.

- Input fields are defined with size information to limit the input string length.
- Literals are defined.
- Submit UI Types are implemented using push buttons.

Forms are used, which direct all responses back to the ***hptGateway*** process, the Gateway Servlet provided by VisualAge Generator which is used by all Web Transaction systems (Note: *hpt* is sometimes spoken as *Highpoint*). See Figure 15 for an example of forms, input fields, submit buttons, and the .

```

</HEAD>
<BODY>
<H1>Staff Info</H1>
<table border=4 width=100% cellspacing=0 cellpadding=20>
<tr><td><table border=0 align=left valign=middle>
<FORM METHOD="POST" ACTION="/hptGateway">
<P>
<b>Id</b>&nbsp;<INPUT TYPE=TEXT NAME="ID" SIZE=4 VALUE=" ">
<b>Name</b>&nbsp;<INPUT TYPE=TEXT NAME="NAME" SIZE=9 VALUE=" ">
<b>Department</b>&nbsp;<INPUT TYPE=TEXT NAME="DEPT" SIZE=4 VALUE=" ">
<b>Job</b>&nbsp;<INPUT TYPE=TEXT NAME="JOB" SIZE=5 VALUE=" ">
<b>Years</b>&nbsp;<INPUT TYPE=TEXT NAME="YEARS" SIZE=4 VALUE=" ">
<b>Salary</b>&nbsp;<INPUT TYPE=TEXT NAME="SALARY" SIZE=8 VALUE=" ">
<b>Commission</b>&nbsp;<INPUT TYPE=TEXT NAME="COMM" SIZE=8 VALUE=" ">
<INPUT TYPE=SUBMIT NAME="BUTTONS" VALUE="Find">
<INPUT TYPE=SUBMIT NAME="BUTTONS" VALUE="Exit">
<INPUT TYPE=HIDDEN NAME="hptPageId" VALUE="322964">
<INPUT TYPE=HIDDEN NAME="hptAppId" VALUE="WEBTRAN">
<INPUT TYPE=HIDDEN NAME="hptHandlerId" VALUE="24461">
</FORM>
</td></tr></table>
</BODY></HTML>

```

Figure 15. HTML Forms

What we are focusing on here is user entry screens which make use of the HTML FORM tag VisualAge Generator will generate for you to surround any HTML INPUT fields you specify when you define a UI Record.

It is also possible in VisualAge Generator to create your own HTML FORMs or HTML ANCHOR tags when defining UI Records. It is also possible to code an XFER to a program name of ' ' when passing a UI Record. This XFER statement behaves in a special way and we will discuss it separately later.

2.1.4 Web Transaction definition

There are multiple structure approaches for how a UI Record is used to display data in a browser and transfer of control is supported when defining a Web Transaction:

- CONVERSE UI Record program design
- First UI Record program design (single segment) with named program navigation (XFER Program WRec, UI Record)
- First UI Record program design (single segment) with form directed program navigation (XFER ' ', UI Record)

These structure options (see Figure 16) can be used exclusively or mixed in a single system.

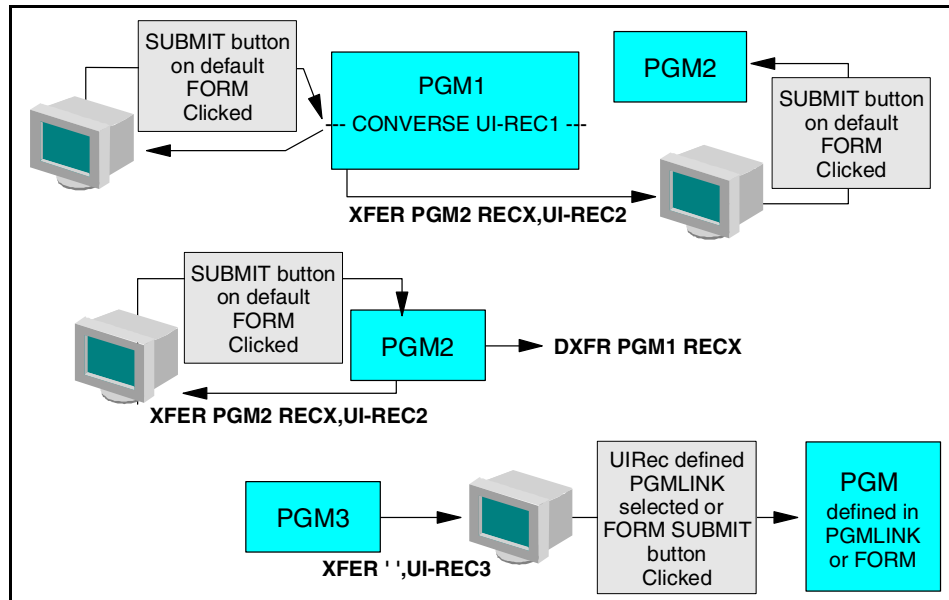


Figure 16. Web Transaction processing structures

The Web Transaction program structure chosen impacts how program logic is specified and the runtime processing of the generated system (see 5.2, “Program structure options” on page 89 for details).

2.1.5 Testing

A Web Transaction can be tested in the VisualAge Generator test facility. When you CONVERSE a UI Record or XFER with UI Record:

- Default HTML is generated.
- A Web browser is launched, if required.
- The Web Transaction pauses on the CONVERSE or on the XFER of the UI Record and waits for an interaction event to occur in the Web browser.

This is shown in Figure 17.

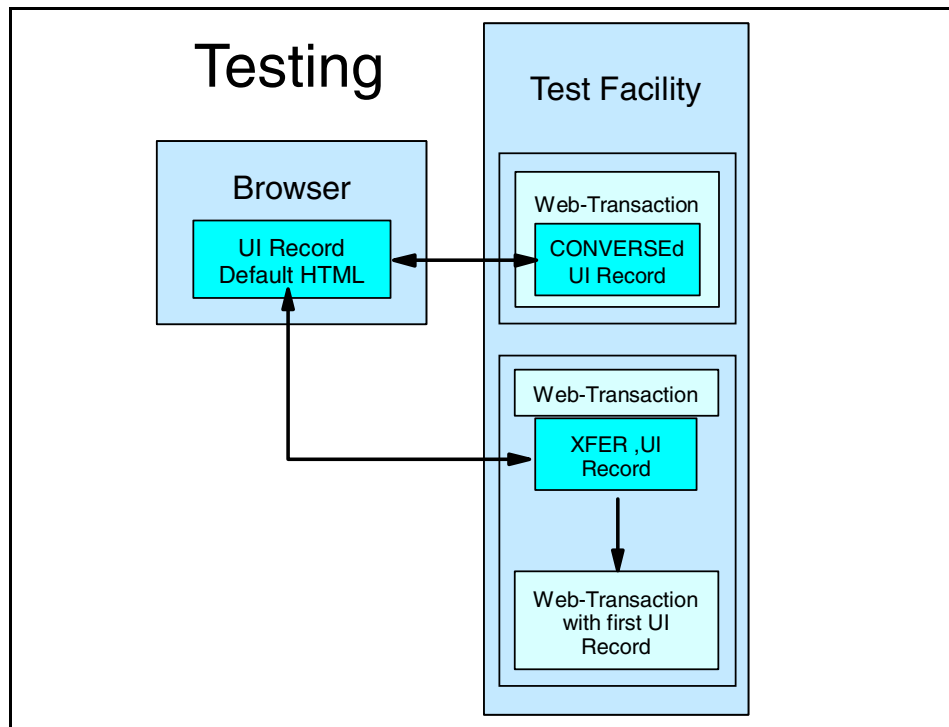


Figure 17. Testing a Web-based system

The test facility manages the processing required to support a CONVERSE or XFER of a UI Record:

- Default HTML is generated that renders a page with all the data in the UI Record (other than fields of UI type NONE). No Java Server Page is produced. The layout of the data is fairly rudimentary; you have a little control, but not to the extent that you will get once you have generated and produced a JSP. The rules for the generation of default HTML is discussed in Chapter 3, "HTML and UI Record definition" on page 55.
- The Web Browser that is registered to the operating system is invoked automatically and this generated page is sent to it.
- The Web Transaction pauses on the CONVERSE or XFER with UI Record and waits for an interaction event to occur in the Web browser.
- If there is an HTML FORM to submit (HTML INPUT fields and an HTML SUBMIT button), that data is sent back to the test facility when the button is pressed. If an HTML ANCHOR tag is used to invoke a Web Transaction, *no* changes input by the user are sent back to the test facility.

- All the edits defined in the UI Record are run.
 - If any of the edits fail, the HTML page is resent to the browser with error messages under each field that failed the defined edits.
 - If the edits succeed, all data is returned to ITF and the program continues on after the CONVERSE or the Web Transaction that was XFERed to starts.
- When the Web Transaction exits, a default entry point page is displayed that shows all Web Transactions currently loaded in the development image that can be run.

This simulates what occurs at runtime when a program terminates; the Gateway Servlet will serve the defined Entry Point Page.

Note: It is planned that the test facility could be used to test Web Transactions when the browser is using runtime code (Java Server Pages). The JSPs will communicate with the Gateway Servlet. There will be a COMMTYPE of the CSOGW properties file which will allow the Gateway Servlet to interact with Web Transactions in the test facility. This is similar to the current linkage table-based support for Callable ITF, except that control is returned to the Gateway Servlet during a CONVERSE/XFER with UI Record as well as at program termination. This is not available at present.

This is a good place to build some Web Transaction programming skills. If you have not already developed and tested a Web Transaction, consider following the scripted exercise in Chapter 6, “Developing Web Transaction programming skills” on page 121.

2.2 Generation of Java components and runtime program

New generation technology is used to produce the runtime implementation components required for a Web Transaction (see Figure 18).

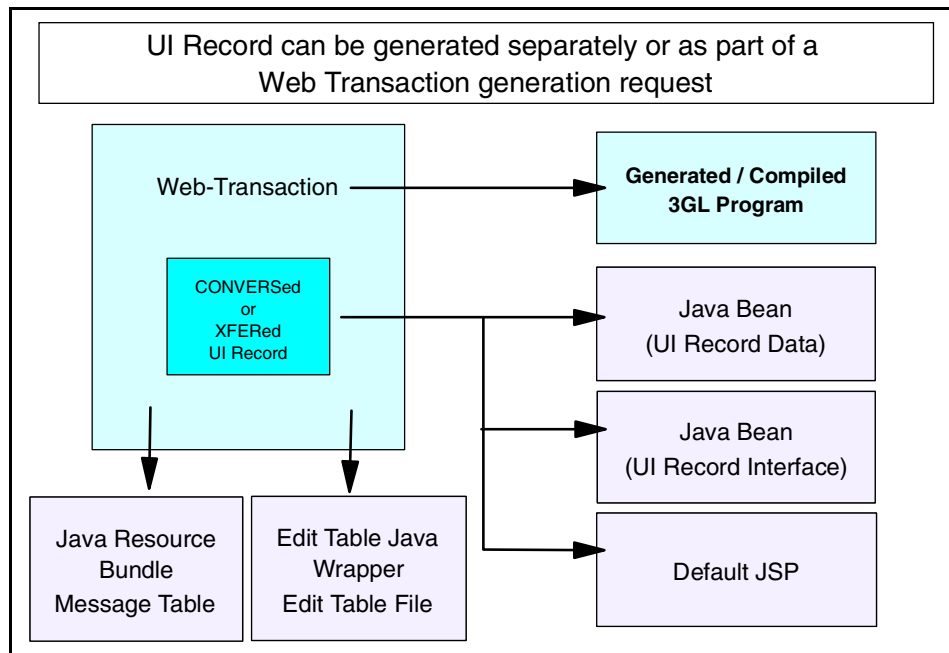


Figure 18. Generation of a Web-based system

You code a Web Transaction. Generation produces:

- A COBOL or C++ load module to run in the VisualAge Generator server environment
- A Java bean, which represents the data in the UI Record
- A Java bean to run any edits (other than VAGen server-side edit routines) for the UI Record, to fill out any titles and labels, handle help, and do any other UI Record processing
- A Java *Resource Bundle*, which represents the data in the message table
- A Java Server Page, which will handle formatting and sending the display of the UI to the Web browser
- Edit table(s); each consists of a binary file and a Java wrapper for it

2.2.1 Java terminology

The description of generation and runtime will make many references to Java. A few key terms are described here to ensure we do not confuse those of you who are new to Java.

Java is an object oriented (OO) language. This means you develop and compile a series of objects which interact with each other by sending messages. Objects are things which are meaningful to the users of your system, and are often real, such as a car, or a book.

Java is made up of supplied code (the Java APIs), and the *Java Virtual Machine* (JVM).

The Java code you develop does not run directly on the native operating system; instead, the JVM for a given operating system interprets your code. Your Java source (a .java file) needs to be compiled into bytecode for the JVM to be able to run it. This bytecode is platform independent and is often stored in either a .class or .jar file.

Classes and instances

Your source code for a particular object will primarily define data and functionality, just as you would for a typical COBOL program. This object source is called a *class*.

A class is a template from which you can create *instances*. For example, you might be defining a Book object. Your class defines what messages Book can understand, and the data, or properties of Book.

Book may have a property called Title. When you build your system, you will probably create many instances of Book. Each instance could have a different title.

Classes are grouped together into packages.

Messages and methods

When you specify the *messages* Book can understand, you write some Java code to represent the processing that should happen when the message is sent. This code is called a *method*.

For example, a message could be to calculate the tax payable on the Book. If you want to send the message to the Book, you may need to pass data to the message, perhaps to specify a country. Messages can receive objects (passed as arguments).

When you send the message to Book, the code in your calculation method will execute. The method may return an object, in this case, the Tax.

JavaBeans

JavaBeans are also referred to in this chapter. In simple terms, *beans* are Java objects whose properties are referenced and changed only through messages. The messages have a specific format and wording. Beans also signal *events*, which other Java objects can respond to. Java programmers typically develop beans, which interact with each other and represent reusable components.

2.2.2 Programs

When a Web Transaction is generated, the appropriate 3GL source code for implementation in the target runtime environment is generated.

Any referenced UI Records may be generated with the program if you want, otherwise individual UI Records can be generated independently.

How the generated program is started

Although the Web Transaction is logically behaving like a "main program", it is physically invoked like a remote "called batch" program; it is synchronously called through the catcher program with a single parameter (the UI Record) passed across the network from the Gateway Servlet.

The UI Record output by the program at CONVERSE time, or on an XFER with UI Record, is passed back over the network to the Gateway Servlet. Thus a UI Record is limited to a maximum of 32K.

Before it starts executing the actual user-written code, the Web Transaction will retrieve data from its WORKDB, if it is being invoked after a previous **CONVERSE** or **XFER webTran VAGenRecord,UI Record**. This is the same behavior as a regular segmented "main program".

If the UI Record on the CONVERSE or the FirstUI associated with the Web Transaction had any edit functions which you did *not* ask to run on the Web, these functions are generated to form part of the program load module. These functions will be invoked, if the appropriate fields are modified, before the rest of the user-defined code is executed.

How the generated program ends

If the Web Transaction issues a CONVERSE, all the storage of the program is saved in the WORKDB by the Web Transaction, and control is returned to the catcher program passing the UI Record data, so a response may be returned to the Web browser.

An alternative to **CONVERSE** is **XFER WebTran ,UI Record**; which will send the UI Record to the Gateway Servlet to be displayed in the Web browser.

When the browser user responds to any buttons defined to this UI Record, which are inside the default HTML FORM generated by VisualAge Generator, the identified Web Transaction is invoked and the passed UI Record data is passed to the program in the defined first UI Record.

Additional details are available in Chapter 5, "Web Transaction design concepts and considerations" on page 85.

The XFER statement refers to a program name, even in the CICS or IMS environment. The actual transaction to be invoked is determined from the linkage table referenced by the Gateway Servlet if you are starting a "conversation" or else a previous setting of EZESEGTR.

XFER can optionally include another VisualAge Generator record as well as the UI Record, for example; **XFER webprog record,UI Record**; . When the XFER runs, the data of this **record** is saved in the WORKDB by the Web Transaction before control is returned back to the catcher program. This **record** cannot be bigger than 32K.

Transferring control between programs

A Web transaction is *not* allowed to issue an XFER statement which does not include a UI Record. This restriction means that the only way for a Web Transaction to transfer control to another program, without sending information to the browser in between, is to use DXFR.

2.2.3 UI Records

The same UI Record generation processing is used for all target runtime environments. There is one common UI Record generator.

When a UI Record is generated:

- UI Record definitions are generated into a Java data bean (VGUIr***UIrecord***), a Java interface bean (***UIrecord***Bean) and a Java Server Page.
- The generated interface bean includes methods for access to the defined data and submit actions included in the UI Record definition and stored in the data bean.

Getters are generated that access all the defined items in output edit form.

Setters are generated that set input data in unedited form. The internal form is set when all input edits (except user edits) are successful. At that point, the internal data is set into the record to be accessible by the 4GL in a user-defined edit function.

- UI Record user edit functions that you ask to run Web-side cannot access the data of the Web Transaction that uses this record. This is because the Web Transaction and the edits are running in completely different address spaces; most likely different machines. Such edit functions are also not allowed to issue any I/O.
- Table edits are handled using separate VisualAge Generator table modules, see 2.2.6, “User edit tables” on page 35.
- A default JSP is produced.

UI Records cannot be bigger than 32K.

2.2.4 UI Record interface bean (UIrecord Bean)

Data accessors

- The interface bean contains Java getter and setter methods for attributes of the data bean (VGUIr**UIrecord**). The attributes are mostly just String data for lowest level data items.
- However, the getters for super structure and occurred items use more complex objects that can be queried to get the substructure/index data:
 - Getters on occurred items return an Array of data in output form.
 - Getters on superstructure items return a object based on the *com.sun.java.swing.table.TableModel* class definition. This is true also if the item is indexable (occurred). The Java code in the default JSP file uses this object to then access the row and column data for display.

Input editing

- When the user submits back to the gateway from the browser, all input data is set to the bean in unedited form. The gateway then invokes a Java method (**#processInput**) to process all the inputs.
 - The gateway invokes any VisualAge Generator edits, such as input required.
 - It then invokes any edit table checks.
 - user-defined edit functions which run Web-side are invoked after all the other edits have been performed and have successfully completed. At that time the edited input data is set into its internal form and is then accessible by the user functions.

Error messages for edit functions are set using EZEUIERR.
- The processInput method returns *true* or *false*.

- If *false* is returned, the gateway will serve the page where the input edits are running; this is usually, but not always the page that was previously sent. The Java code in the JSP that accesses the interface bean at this point will retrieve the error message and put it in an error field.
- If *true* is returned, the Gateway Servlet reads the linkage table and invokes the appropriate catcher program.

Output edits

- Output edits occur when the data is accessed (**#getFormattedText*** methods).

The Java API of the interface bean is discussed in Chapter 4, “Java Server Pages and the UI Record interface bean API” on page 75.

2.2.5 Java Server Page produced by VisualAge Generator

Generation of a UI Record produces a JSP. The JSP provides sample code to access the interface bean from the `HttpServletRequest` object and thus the attributes in the data bean. The JSP code shows how the HTML field names are mapped to the bean attribute getters and setters.

The display of error messages associated with input edit error handling is also included in the default JSP code. The **getErrorMessage*** methods are referenced and the message is placed into an error message field.

Control information regarding hidden fields that are used by the Gateway Servlet to differentiate certain browser request actions is included. These fields are not to be updated by the UI programmer.

2.2.6 User edit tables

If you generate edit tables used by a UI Record individually, do it twice:

1. Choose the Java runtime environment. The output from this generation is a Java bean wrapper to access the table data (**VGTbledtbl**).
2. The target environment is where the WebSphere Application Server is running. Generation creates a binary `.tab` file with the table data.

Otherwise, generate the edit table with the associated UI Record and Web Transaction.

The `.tab` file should be deployed in the root directory used for the generated JavaBeans (the beans include a directory tree based on package structure).

2.2.7 User message tables

To make the job of revamping existing 3270 applications easier to perform, User Message Tables can be generated into Java Resource Bundles that are then accessible by the generated UI Record; choose the Java target runtime environment and specify the GENRESOURCEBUNDLE generation option.

When a user message table is generated into a resource bundle, the numeric keys of the first column is turned into a String.

However, any table that has 2 columns of CHA or MIX items can be used as a message table in a Web Transaction and generated into a Java Resource bundle, provided that:

- The last 3 characters of its name is the NLS code.
- The table prefix is associated with the Web Transaction via the program properties.

Message display is requested using the EZEUIERR function.

2.3 Runtime system implementation

A typical runtime system implementation configuration is shown in Figure 19.

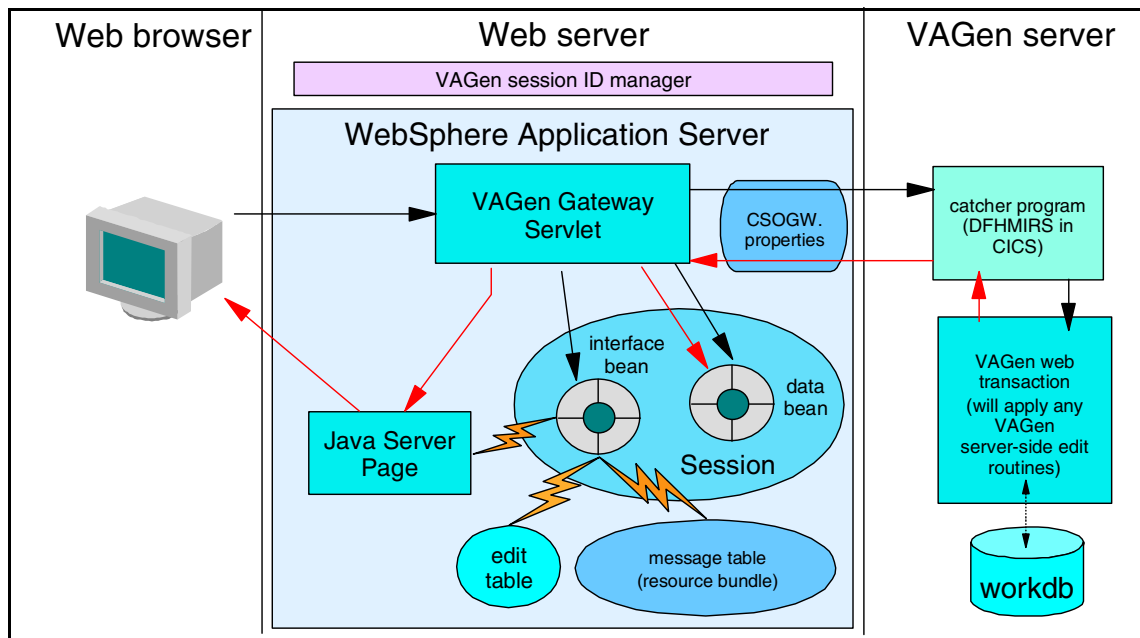


Figure 19. Web-based system development

- There are three logical tiers in the runtime environment::
 1. Web browser
 2. Web server with servlet support, perhaps using the IBM WebSphere Application Server
 3. VisualAge Generator runtime environment (there are many supported runtimes, including, and soon, Enterprise Java Server)

The Web server and VisualAge Generator runtime environment could potentially be on the same machine (3 logical tiers on 2 physical platforms).

- VisualAge Generator runtime includes the following:
 - Gateway Servlet
 - Session ID manager
 - Catcher program (when required, only used for some runtime environments, other runtime environment configurations make use of existing communications facilities)
- You code a Web Transaction. Generation produces:
 - A COBOL or C++ load module to run in the VisualAge Generator server environment
 - A Java bean, which represents the data in the UI Record
 - A Java bean to run any edits (other than VAGen server-side edit routines) for the UI Record, to fill out any titles and labels, handle help and do any other UI Record processing
 - A Java *Resource Bundle*, which represents the data in the message table
 - A Java Server Page, which will handle formatting and sending the display of the UI to the Web browser
 - Edit table(s); each consists of a binary file and a Java wrapper for it
- You need to create and deploy a CSOGW.properties file, for the Gateway Servlet to use, to define the parameters used to control communication with the VisualAge Generator Server tier.

2.3.1 Basic processing concepts

The basic concepts of a Web-based system are discussed below.

WebSphere Application Server

In simple terms, an application server is a plug-in to a Web server. This means that you need to install a Web server and then plug the application server into it. The application server is then started automatically when the Web server starts. IBM WebSphere Application Server ships with the IBM HTTP server, but there are a variety of other Web servers you can use for various operating systems.

IBM WebSphere Application Server provides an environment to run servlets. This includes a single shared *Java Virtual Machine (JVM)*.

IBM WebSphere Application Server Advanced also includes other features, such as:

- Enterprise Java Bean server
- Pooling of database connections made from Java objects executing in the JVM, through provision of an *IBM Connection Manager*
- Support for clustering and scaling of application servers
- Java classes to store and retrieve information about visitors to your Web site in a JDBC compliant database

Java servlet

A servlet is a Java class, that typically extends the class `HttpServlet` in the `javax.servlet.http` package. This package, together with `javax.servlet`, make up the standard servlet API as defined by Sun, which is available as the *Java Servlet Development Kit (JSDK)*. `HttpServlet` implements the `Servlet` interface, which provides servlet behavior.

Servlets run server-side, they are not downloaded to a Web browser. They are typically invoked by initiating a URL link; perhaps from an HTML FORM tag, an HTML ANCHOR tag, or simply by typing the URL in the browser location field.

When this URL is fed into the Web server you are communicating with, the request will be dealt with by the Web server itself, if it has servlet support, or passed on to a plug-in, such as IBM WebSphere Application Server (see Figure 20).

Servlet support, as provided by products like IBM WebSphere Application Server, gives you a servlet runtime environment:

- This is an implementation of the JSDK. Some of the JSDK code merely supplies interfaces, and the servlet engine vendors need to provide the full implementation. These implementations, and thus their available features, can vary from product to product.
- The runtime provides a single JVM which all the servlets and associated Java objects are loaded into, and run in.
- Some servlet engines, such as IBM WebSphere Application Server, provide support to process Java Server Pages.

Figure 20 shows a typical invocation of a servlet.

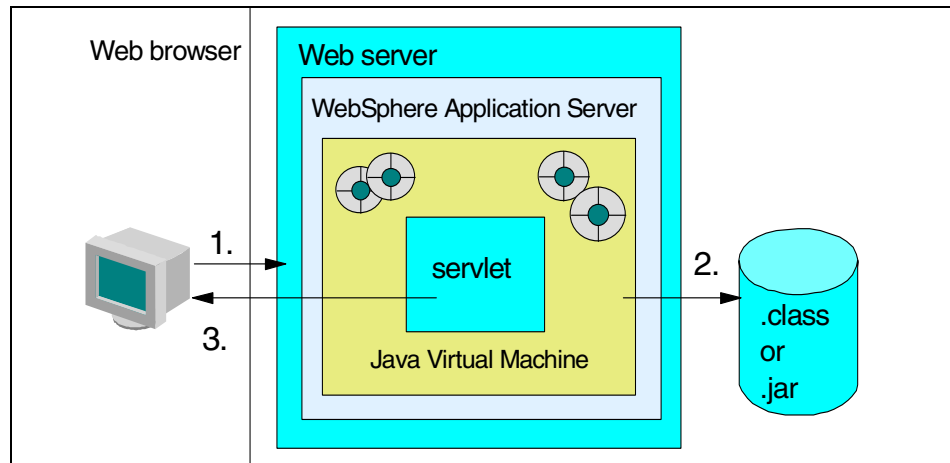


Figure 20. Servlets

1. The user clicks the SUBMIT button on their HTML FORM. This causes the Web browser to invoke a URL by sending an HTTP request to the Web server. The Web server sees that the URL matches an alias defined to IBM WebSphere Application Server and passes the request to the plug-in.
2. IBM WebSphere Application Server:
 - Loads an instance of the servlet into memory (if there is not one already loaded) and runs its init method.
 - Once the instance is loaded, or if there was already an instance loaded, its service method is invoked.

The servlet can read information that the user typed into the FORM using a `HttpServletRequest` object, which the system passes into the service method. It can also interact with JavaBeans, or even Enterprise JavaBeans, and connect to CICS, databases, or other platforms or products.

Typically, there can only be one instance of a servlet loaded into the JVM.

3. Once its processing is complete, the servlet can send an HTTP response back to the Web browser, in the following ways:

- By setting HTTP headers and printing out an HTML stream, using a `HttpServletResponse` object, which the system passes into its service method, as shown here.
- By invoking a Java Server Page.

Java Server Pages

The source code for a Java Server Page (JSP) is written as an ASCII file, suffixed `.JSP`. The file contains HTML tags, plus special JSP tags, which represent dynamic information in the file. Some of these special tags can contain actual Java code inside them. See Chapter 4, “Java Server Pages and the UI Record interface bean API” on page 75 for details of JSP syntax.

The JSP source needs to go through a process called page compilation before it can run. IBM WebSphere Application Server will perform this process (if it is necessary) on the fly. The output is a Java source and class file for a servlet, and these are permanently saved on disk for future use.

The servlet output from page compilation has two main functions:

1. It sets HTTP response headers and prints the straight HTML tags in the original JSP source to an output stream to go back to the Web browser.
2. It executes any of the Java code corresponding to the various special JSP tags in the original document.

Figure 21 shows a typical invocation of a Java Server Page.

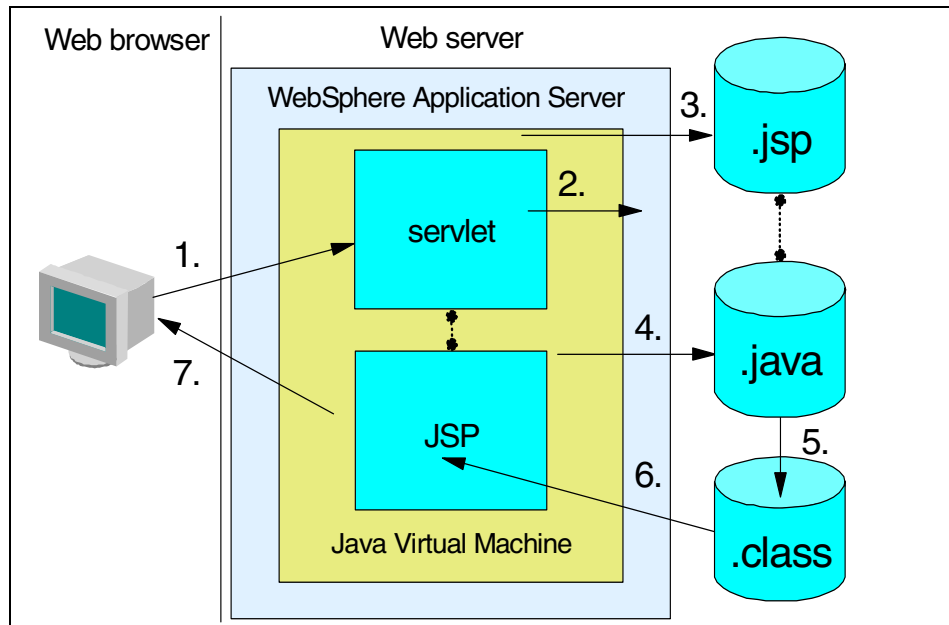


Figure 21. Java Server Pages

The processing steps identified in Figure 21 are:

1. The Web browser sends an HTTP request which invokes a URL which actually represents a servlet. The Web server passes this request to IBM WebSphere Application Server, which processes the servlet.
2. Once the servlet is ready to send an HTTP response back to the browser, it invokes a JSP. It specifies a URL corresponding to the .JSP file to do this. The servlet's HttpServletRequest object is passed through to the JSP.
3. Because IBM WebSphere Application Server has an alias corresponding to *.JSP, it processes this request.
4. If there is no current servlet source and class file for the JSP on disk, or if the .JSP file has changed, IBM WebSphere Application Server runs page compilation on the .JSP source.
5. Once page compilation has made servlet source code from the .JSP, the Java compiler JAVAC is run to compile it into a Java class file.
6. When the Java has compiled (if page compilation needed to run), an instance of this new servlet class is loaded into memory and its service method invoked.

7. This new object ultimately sends an HTTP response back to the browser by printing out a stream of HTML tags, some pulled out of a buffer, some dynamically created from the code originally held in the special JSP tags.

Implementing state data in a Web server system

Many (if not most) applications will require that data be maintained while a Web user interacts with a Web system. This data will store things such as the current state of the conversation, user specific information, and so on.

Sun's JSDK provides some level of support for the creation of *HttpSession* objects. Sessions (*HttpSession* objects) are a way to temporarily store a reference to objects which hold data associated with a particular Web user while they are interacting with your site.

Much of the JSDK support is merely interfaces while the servlet engine vendors provide the actual back end implementation code. Thus the actual session features may vary from vendor to vendor.

Figure 22 shows the use of a session object by a servlet and JSP.

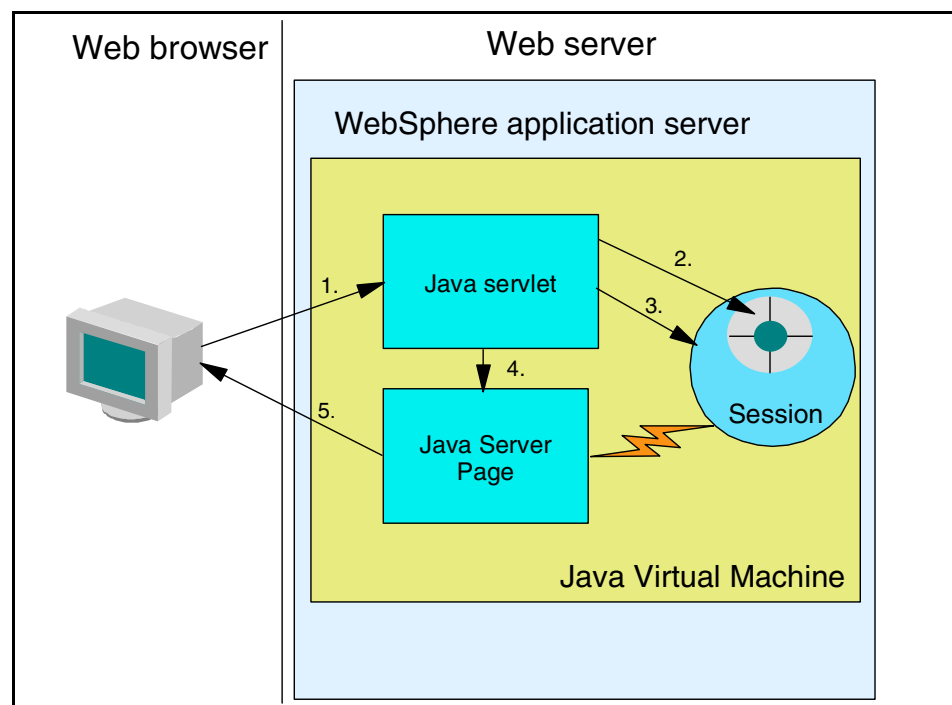


Figure 22. The *HttpSession* object

The processing steps identified in Figure 22 include:

1. The browser invokes a servlet from an HTML FORM.
2. The servlet reads the data from the FORM INPUT fields using its `HttpServletRequest` object. It may create, or *instantiate*, new Java objects which hold this user data. It may also initiate some enterprise access, for example, look up information on a database. The results of the enterprise access may cause the instantiation of other objects.
3. The servlet now needs to create a new session object for the user, or else reference an existing one. There is a single method, `getSession()` of the `HttpServletRequest` to do this.

The actual mechanism of creation and reference in an IBM WebSphere Application Server environment, under-the-covers, is typically as follows:

- When a session object is created, the application server also creates a *cookie*. The cookie is a Java object with properties and behavior. The cookie created for sessions holds a unique id. This id is stored in a correlation table which also stores a reference to the session object.
- HTTP responses include the actual HTML body to display, plus headers, which control aspects such as caching. When the HTTP response is sent to the browser in step 5, the session cookie is sent to the browser in the headers.
- The default behavior for the session cookie is for it to live in the browser memory while the browser remains open.
- The session cookie is passed back to IBM WebSphere Application Server with the HTTP request headers when a servlet is invoked from the browser. When you try to reference an existing session object, the session cookie in the request headers from that browser is used internally to find the session object. Thus the cookie ties up a browser with an IBM WebSphere Application Server object.

Once we have reference to a session object, we can store references to objects for that user in it. Typically the objects we referred to in step 2.

4. The servlet invokes a JSP, passing the `HttpServletRequest` object.
5. JSP syntax will allow you to effectively run a `getSession()` against the request and also to then query the session object to locate user objects, so they may be displayed to the user in the final HTML stream sent to the browser.

IBM WebSphere Application Server provides various facilities for sessions:

- Tuning of the JVM memory allocated to the objects, with the ability to swap them to disk, is possible.
- Every session object will timeout after a specified period of being unused.
- It is possible to use URL encoding instead of cookies, though it is not recommended. Users can turn off cookie support in their browsers, so you need to decide how your Web system will handle this.
- Session objects may be shared between a cluster of IBM WebSphere Application Servers.

VisualAge Generator's support of Web Transactions makes use of session objects.

2.3.2 Gateway Servlet

The Gateway Servlet links the JSP/JavaBeans on the Web server-side with the Web Transaction on the VisualAge Generator server-side. The management of data movement from the appropriate Java Bean to/from the Web Transaction is implemented by the Gateway Servlet.

Basic functionality

The Gateway Servlet has 5 basic jobs:

1. Provides a controlled view to the VisualAge Generator Web Transactions available on a given back end system (a valid target runtime environment, such as a CICS/ESA system).

Before running a Web Transaction that exists on a back end system, the Gateway Servlet will ask for a logon, and then will list the available Web Transactions. Which Web Transactions show up depends on your customization.

2. Acts as a gateway for all communication with Web Transactions on back end systems (hence the term Gateway Servlet).

To accomplish this, the Gateway Servlet works with a communication protocol and a catcher program that runs on the back end system. The catcher program will control invocation of the Web Transaction.

3. Associates the data being CONVERSEd or the UI Record passed on XFER in a Web Transaction with the appropriate Java Bean, and vice-versa.

The output data on the CONVERSE or XFER with UI Record must be given to the correct Java Bean; the input from the end user must be passed into the UI Record the Web Transaction receives.

This processing is coordinated with the catcher program, which will be involved with the processing required to communicate with the Web Transaction and pass it the UI Record data.

4. if the Gateway Servlet was invoked by a SUBMIT button on an HTML FORM, it applies any standard supplied edits, such as input required, or any Web server-side edit routines or any edit tables to the data input by the user. This is done using the interface bean associated with the data bean for a given UI Record.
5. Invokes the appropriate JSP that accesses the data bean associated with that user through the interface bean for that UI Record for rendering the HTML page back to the browser.

Configuration

The Gateway Servlet is configured at the Web server with various properties, such as:

- Logon Page: HTML/JSP page that takes user LOGIN data
- Entry point: HTML/JSP page that defines all possible programs to be accessed by the Gateway Servlet. This might also be a menu program written in VisualAge Generator as a Web Transaction, that is invoked directly, which would then CONVERSE or XFER a UI Record as a menu. This allows tailoring of menu based on user.
- Error Page: HTML/JSP page that is the default page for showing runtime errors

For more information on the properties of the Gateway Servlet, see Table 11 on page 315.

The CSOGW.properties file defines parameters used to access the back end catcher program. Its properties include:

- Underlying protocol to use to connect to the back end system
- Server name/location
- Data conversion

These will map to the Web Transaction runtime entry point.

Configuration is discussed further in Chapter 12, "Runtime environment scenario implementation" on page 247.

2.3.3 Session ID Manager (SIDM)

This is an independent server whose primary purpose is to generate unique ids. The SIDM is accessed through Java's RMI facility and remote method calls are used to obtain and release ids. Web Transactions themselves never communicate directly with the manager, it happens internally, in the Gateway Servlet or handler objects associated with a particular conversation with the user.

A given VisualAge Generator server platform may be accessed by multiple Web servers running on different operating systems. The SIDM allows unique ids for a user and each user conversation (the user could have many Web browser windows open simultaneously, and therefore many simultaneous conversations) regardless of the number of Web servers in a given complex.

The SIDM can be accessed remotely from a given Gateway Servlet. The Gateway Servlet has a property where you can specify the location of the SIDM, so a single SIDM can be shared.

The IDs the SIDM creates are 8 bytes long. They have a prefix followed by a number. The default prefix is CUIR, but you can start the SIDM with an argument, and use that argument to change the prefix. That way, the prefix could be as short as a single byte, if you so desire, leaving more room for the number.

A new id for the user is requested if:

- This is the first interaction with the user.
- The user's session object previously timed out, so they are asked to log on to the Gateway Servlet again.
- The user closed their browser since the last interaction with the Gateway Servlet; again they will be asked to log on.
- The user previously logged off the Gateway Servlet.

A new conversation id is requested if you:

- Select a Web Transaction from the entry point.
- Click on an HTML ANCHOR tag to invoke a Web Transaction.
- Click on a SUBMIT button inside a FORM you defined explicitly in the UI Record to invoke a Web Transaction.

All conversation ids and the userid are released if:

- The session object times out due to user inactivity.
- The user explicitly logs out of the Gateway Servlet.

An individual conversation id is released if:

- An XFER ' ' ,UI Record is issued.
- The Web Transaction ends normally without transferring control anywhere.

The conversation id is accessible in EZELTERM. It is used to key data stored in the WORKDB when the Web Transaction executes on the VisualAge Generator server platform and uses the WORKDB.

The SIDM generated id for the user is accessible in EZEUSR. The userid known to the runtime server system is available in EZEUSRID.

2.3.4 Web Transaction runtime scenario

This scenario is best explained by stepping through the invocation of a Web Transaction designed in any combination of the ways shown in Figure 23.

CONVERSE and XFER program runtime scenario

What we are focusing on here is user entry screens which make use of the HTML FORM tag that VisualAge Generator will generate for you to surround any HTML INPUT fields you specify when you define a UI Record. The structure of these types of systems is shown in Figure 23.

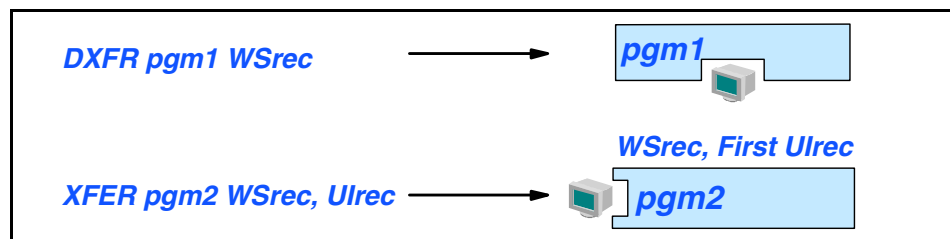


Figure 23. CONVERSE and XFER program Web Transaction structure options

Figure 24 shows a runtime implementation of PGM1 or PGM2.

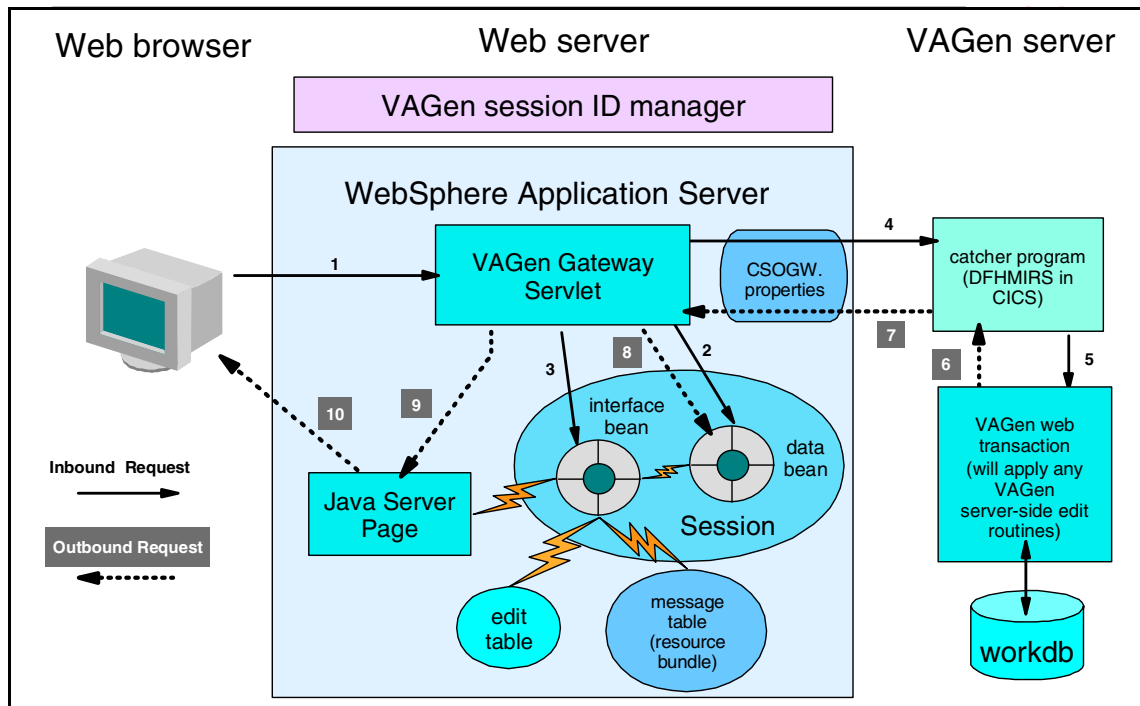


Figure 24. CONVERSE and XFER program Web Transaction runtime processing

The numbered processing steps shown in Figure 24 are described below:

1. At runtime, a user issues a Gateway Servlet request using their browser.

The processing sequence depends on whether it is the first time the user has invoked the Gateway Servlet, or whether the request is based on responding to data previously sent to the browser.

If this is the first time through:

- The user has the option of typing in a URL for the Gateway Servlet. This causes it to serve an HTML page/JSP, called the *entry point page*, which lists all the available Web Transactions.
- The session ID manager is invoked to assign a unique ID for the user, and a session object is created for the user. This ID for the user is unique over multiple Web servers and VisualAge Generator server platforms. The ID is stored in the session object and may be obtained through EZEUSR.

- If the Gateway Servlet is configured to request users to logon, then just before the entry point is served, the user will be asked to authenticate themselves. The security information is then stored on the WebSphere Application Server in a session object for use later.

If this is not the first time through:

- The Gateway Servlet would be invoked from one of the following:
 - A SUBMIT button or ANCHOR tag on the entry point page
 - A SUBMIT button on an HTML FORM which was output from a previous CONVERSE or XFER with the UI Record of a Web Transaction.
- If the entry point page is being used, so a new program is being invoked, the session ID manager is contacted to assign a unique ID for that conversation with the browser user. This ID is stored in the session object and can be obtained through EZELTERM.

2. If this is the first time through, we go straight to step 9. and the entry point page, and perhaps additionally the logon page, is served.

Otherwise, if the Gateway Servlet was triggered from a SUBMIT button, on a JSP sent as the result of a previous CONVERSE or XFER with UI Record, it stores this user data in a data bean.

The bean is associated with a session object for that user, so it can be referenced on later interactions with the user.

3. If the Gateway Servlet was triggered from a SUBMIT button, on a JSP sent as the result of a previous CONVERSE or XFER with UI Record, the interface bean is used to apply to INPUT or INPUT/OUTPUT fields any:
 - Standard supplied edits, such as numeric, input required, and so on
 - Edit tables
 - Web-side edit routines

The message table is used to look up any message keys associated with a particular supplied edit for a given data item. If an error is found, we go straight to step 9.

4. We are now ready to invoke the VisualAge Generator Web Transaction. The Gateway Servlet synchronously invokes the catcher program on the target runtime environment.

In a similar way to VisualAge Generator client/server systems, a CSOGW.properties file is used to determine:

- What underlying protocol should be used to connect to the catcher program, for example, CICS ECI
- Data conversion
- Remote system identifier
- Transaction id, if using CICS or IMS and there is no current CONVERSE or XFER with UI Record outstanding

CICS is a commonly used VisualAge Generator server platform. To connect to a CICS server in order to run a Web Transaction, you would use the CICS client. The Gateway Servlet connects to CICS client using the CICS Transaction Gateway. CICS client will connect to a CICS server and invoke the catcher program, otherwise known as the mirror program, DFHMIRS, by the external call interface (ECI) or the EXCI for MVS.

5. The catcher program invokes the requested Web Transaction.

If the page on the browser had originally been sent as a result of the CONVERSE of a Web Transaction, or XFER with UI Record *and* some other VisualAge Generator record, this data is retrieved from the WORKDB.

If the Web Transaction was invoked from an HTML FORM, and there were any VisualAge Generator server-side edit routines associated with INPUT or INPUT/OUTPUT fields in the UI Record, they are run at this time.

If the VisualAge Generator runtime environment is CICS, the Web Transaction is invoked through a CICS LINK from DFHMIRS.

6. If server-side edits failed, or the VisualAge Generator Web Transaction issues a CONVERSE or XFERs with a UI Record:

- The Web Transaction saves program storage, for CONVERSE, or saves the working storage record specified on the XFER Pgm WSRec, UIRec statement (if included), in the work database.
- Control and the UI Record data returns to the catcher program.

In a CICS environment WORKDB will be a Temporary Storage Queue, in IMS, some form of database. The queue name is uniquely keyed on EZELTERM.

7. The logical unit of work ends, all task resources are released and control returns to the Gateway Servlet.

8. The bean name to access and the data of the UI Record to be sent to the browser is returned to the Gateway Servlet. The Gateway Servlet stores the UI Record data in the user's data bean using methods in the interface bean. The data bean and interface bean are put into the session object for later use.
9. The Gateway Servlet then uses the interface bean to determine the Java Server Page name and serves it. The process of serving this JSP will access attributes in the data bean using the interface bean. Reference to the interface bean is passed through in the HttpServletRequest object.
10. The JSP sends an HTML page to the browser that is filled out with data that was in the UI Record passed from the VisualAge Generator server when it executed its CONVERSE or XFER with UI Record.

UI Record data items with a UI type of NONE will not be visible in the HTML source (unless they are referenced in the program link parameters for a form or program link definition).

This runtime process continues until the Web Transaction ends. At this point the Gateway Servlet knows it has ended and it serves a default entry point page.

XFER ' ', UI Record

It is possible in VisualAge Generator to create your own HTML FORMs or HTML ANCHOR tags when defining UI Records. It is also possible to code an XFER to a program name of ' ' when passing a UI Record (moving a blank to EZEAPP and issuing an XFER EZEAPP, UI Record also works).

The structure of this type of system is shown in Figure 25.

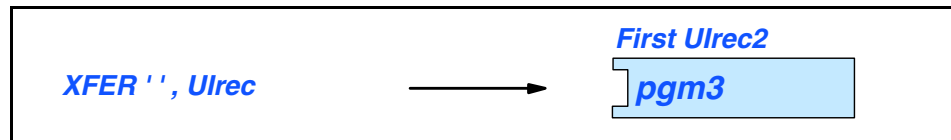


Figure 25. XFER ' ', UIRecord Web Transaction structure design option

When a Web Transaction issues an XFER ' ', UI Record transfer request, the specified UI Record is sent to the browser. Program data is not saved anywhere on the Web Transaction runtime platform, and the JavaBeans for the UI Record are not saved as session data on the Web server (no state). All you have is the data in the HTML page (visible or hidden).

UI type NONE data items should be used cautiously with UI Records that will be used in a no-state implementation. This is because UI Type NONE data

items are stored in the data bean and not sent to the browser and the data cannot be retrieved from a data bean that has not been saved on the Web server.

The UI Record you serve in these circumstances needs to contain programmer defined data item(s) of UI type FORM and/or data item(s) of UI type Program Link. You cannot leave VisualAge Generator to just create its default FORM for you, as that FORM is used for ongoing conversations with Web Transactions, achieved by using XFER to a named program or CONVERSE.

Since programmer defined FORMs and Program Links represent spawning a new thread of conversation, they are best suited to be used on a page served by XFER ' ' ,UI Record, that way no data from previous conversations is left hanging around on either the Web server or the VisualAge Generator server.

If a Web page which contains a Program Link is served by a CONVERSE, and the user clicks on that Program Link, all the data of the conversing Web Transaction will be left saved on the VisualAge Generator server, waiting for the user to come back and respond to a submit button on the default FORM generated by VisualAge Generator for that CONVERSEd UI Record.

XFER ' ' ,UI Record will give you the most Web-like system design. A CONVERSE structure or XFER program approach results in a more modal, conversational style system, like a 3270 TUI system.

The back button

Because data is stored in the IBM WebSphere Application Server session object, it is possible to use browser back and forward buttons and get pages originally built by VisualAge Generator Web Transaction rebuilt without having to rerun these Web Transactions.

- If you use an XFER ' ' , UI Record approach, any Web pages served this way are always reachable using the back and forward browser buttons, as they are served by the Gateway Servlet with caching switched on, so the browser can hang on to them and reload the page in response to back and forward requests.
- If you are trying to use the back button for Web Transactions which CONVERSE or use the XFER PGM WSRec, UIRec syntax you will only be able to get to the most recent JSP in the ongoing conversation.

Pages served as a result of CONVERSE or XFER to named program have their caching turned off by the Gateway Servlet when it serves them.

Resource clean up

All the objects stored in the user's session object will be deleted to release storage when WebSphere Application Server configuration settings cause the user's session object to time-out or the user asks to log out of the entry point page. The session ID manager is also contacted to release the assigned IDs for that user, so they may be reused elsewhere. The processing to do this is handled by the Gateway Servlet.

Work database data for a given Web Transaction on the VisualAge Generator server platform is only released when:

- The Web Transaction terminates normally without transferring control anywhere or sending an UI Record.
- The Web Transaction DXFRs to another program.
- The Web Transaction issues an XFER ' ' , UI Record.
- An id is reused by the session ID manager.

Chapter 3. HTML and UI Record definition

Web Transactions use a UI Record as the link between the user interface (client-side) domain and the business processing (server-side) domain. The UI Record holds various data which will be converted to HTML to show on a browser.

This chapter is aimed at VisualAge Generator programmers, who may be inexperienced with HTML, to explain how the HTML generated from a UI Record definition works. This knowledge is necessary for VisualAge Generator programmers so that they can see:

- What is and is not possible through UI Record definition
- What extra functionality may be added in runtime, but will be unavailable during ITF testing
- What limitations exist in browsers when compared to 3270 development

The HTML representation produced for a UI Record during VisualAge Generator generation is implemented as a Java Server Page (JSP). JSP syntax is reviewed in Chapter 4, “Java Server Pages and the UI Record interface bean API” on page 75.

To customize the runtime environment, the Java Server Page generated by VisualAge Generator can be customized as follows:

- Valid HTML may be inserted in the JSP to customize the look-and-feel. This includes additions such as using images (), applets, activeX controls, HTML tables, frames and framesets.

Note: An explanation of all these tags is beyond the scope of this redbook, but skills in this area will be a requirement for those who will be tailoring the generated Java Server Page.

- Client side JavaScript or VBScript can be added to the JSP.

This can do much to enhance the end user view and interaction with the page and complement the server side Web Transaction, rather than compete with it.

We will look at the above, to some extent, in Chapter 4, “Java Server Pages and the UI Record interface bean API” on page 75.

3.1 An HTML document

An HTML document includes two sections, HEAD and BODY, enclosed in HTML tags. This basic structure is shown below:

```
<HTML>
<HEAD></HEAD>
<BODY></BODY>
</HTML>
```

The BODY holds the information visible in the browser.

The interaction that the end user can have with the displayed document is created using LINK and FORM related tags present in the document BODY.

The HEAD section holds control information:

- <META> tags that can affect caching or redirecting to another URL..
- <SCRIPT> tags; often JavaScript is included in the HEAD so it is all available and loaded when the BODY starts to load.
- <STYLE> tags, to control document look-and-feel.
- <TITLE> tag; the window title.

3.2 General HTML tags

General purpose HTML tags (other than LINKs or FORM related tags) are discussed in this section.

3.2.1 TITLE

The document or window title is defined using a TITLE tag set:

```
<TITLE>a window title</TITLE>
```

This goes inside the <HEAD> tag.

VisualAge Generator supports this through a property defined for the UI Record itself. This property is called the UI title.

3.2.2 General displayable text

The UI Record implements displayable text using a data item with a UI type of OUTPUT. The presentation of an OUTPUT data item's processing depends on the number of occurrences:

- If the number of occurrences = 1, then the OUTPUT data item is presented as follows:

```
<B>Label UI property as defined for the data item </B>value of the data item
```

- If the number of occurrences > 1 and the data item is not inside a data item of UI type FORM, then the OUTPUT data item is presented as follows:

```
<B>Label UI property as defined for the data item </B><PRE>Value for each occurrence of the data item on a separate line</PRE>
```

Note: The label and value data presented in the HTML is obtained from the UI Record beans and filled in using Java Server Page syntax.

3.3 FORMs

In this section we examine FORMs and how VisualAge Generator supports them as part of UI Record implementation.

3.3.1 The HTML FORM tag

The basic structure for a FORM tag set is shown below:

```
<FORM ACTION="URL" METHOD="POST|GET">
</FORM>
```

The FORM tag encloses various other HTML tags:

```
<INPUT>, <TEXTAREA>, <SELECT>, <OPTION>, <OPTGROUP>, <BUTTON>,
<LABEL>, <FIELDSET>, <LEGEND>
```

These tags are only valid when enclosed by a FORM tag and are used to represent the data of the FORM.

An HTML FORM works with an `<INPUT TYPE="SUBMIT">` tag, whose definition the FORM tag encloses. This INPUT field displays as a button. If you do not include at least one such button, the FORM will be useless (unless you intend to do fancy stuff with JavaScript or VBScript, or the FORM happens to have only one input field).

When the user clicks the SUBMIT button, all the data in the form is transmitted to the URL named in the ACTION attribute of the FORM tag, this is called submitting the FORM.

Note: READONLY fields are passed on FORM submission, DISABLED fields are not.

The HTML tags representing the FORM data are discussed in detail below.

<INPUT>

The <INPUT> tag can represent:

- Text or password field the user can type into on the Web page:

```
<INPUT TYPE="TEXT|PASSWORD" NAME="?" VALUE="initial data"
SIZE="size the field displays as in the browser"
MAXLENGTH="maximum digits the user can type into the field"
[DISABLED] [READONLY]>
```

VisualAge Generator does not support the generation of the attribute TYPE="PASSWORD", but you can tailor the Java Server Page generated from a UI Record to add the use of the password attribute.

- Check box — A yes/no choice for the user:

```
<INPUT TYPE="CHECKBOX" NAME="?"
VALUE="data sent when the FORM is submitted if the box is checked"
[CHECKED] [DISABLED]>
```

The CHECKED attribute will preselect the checkbox.

- Radio button — Represents a set of mutually exclusive choices for the user:

```
<INPUT TYPE="RADIO" NAME="?"
VALUE="data sent when the FORM is submitted if the button is checked"
[CHECKED] [DISABLED]>
```

You would never have less than two of these. Radio buttons are grouped by specifying more than one INPUT field of TYPE="RADIO", where the NAME is the same.

The CHECKED attribute will preselect the radio button.

- Submit or reset button:

```
<INPUT TYPE="SUBMIT|RESET" VALUE="text shown on button" [DISABLED]>
```

Clicking a SUBMIT button will submit the FORM. Clicking RESET does NOT submit the FORM, it merely resets all the form data at the client-side to its initial values.

- A hidden field in the form which is not visible to the user:

```
<INPUT TYPE="HIDDEN" NAME="?" VALUE="?">
```

The hidden field data can be seen in the HTML source if you ask the browser to display the HTML and the hidden field data is sent on FORM submission.

- Image based button:

```
<INPUT TYPE="IMAGE" SRC="URL of image"
ALT="textual description of image"
ALIGN="TOP|MIDDLE|BOTTOM|LEFT|RIGHT"
[USEMAP="map for client-side image mapping of the button image"]
[DISABLED]>
```

An image button will act as a submit button.

- Scripted button:

```
<INPUT TYPE="BUTTON" VALUE="text shown on button"
onClick="JavaScript or VBScript"
[DISABLED]>
```

A scripted button has no functionality other than what you add with JavaScript for the event handlers available for this HTML tag; onClick, onFocus and onBlur.

- File upload field:

```
<INPUT TYPE="FILE" NAME="file name to upload"
VALUE="initial data" ACCEPT="mime types" [DISABLED]>
```

VisualAge Generator does not support the direct generation of image based, scripted or reset buttons or file upload fields, so we will not discuss them further here. You could, of course, tailor the Java Server Page produced at generation time to include some of these tags.

All input fields additionally support the following attributes:

- TABINDEX attribute for you to control the tabbing sequence
- ACCESSKEY short cut via the keyboard to get to that field

<TEXTAREA>

The <TEXTAREA> tag represents a multiline input field:

```
<TEXTAREA NAME="?" [ROWS="?"] [COLS="?"] [DISABLED] [READONLY]
[ACCESSKEY="a letter"] [TABINDEX="?"]
>initial display data in the entry area</TEXTAREA>
```

There is a WRAP attribute to allow user input to wrap to the next line without the user having to hit Enter.

<SELECT> and <OPTION>

The <SELECT> and <OPTION> tags will implement a single or multi-select list. The list can be presented as a scrolling list or a drop-down list.

Here is an example of the <SELECT> tag set:

```
<SELECT NAME="?" SIZE="?" [MULTIPLE] [DISABLED]
[ACCESSKEY="a letter"] [TABINDEX="?"] >
</SELECT>
```

If you omit MULTIPLE you get a single select list. If you also set SIZE="1" you will get a drop-down list.

SELECT encloses OPTION tags which represent what appears in the list:

```
<OPTION [SELECTED] VALUE="data sent when the FORM is submitted if
this option is selected" [DISABLED]
>text shown in option list in the browser</OPTION>
```

<OPTION SELECTED> will preselect that option entry.

The remaining tags are not generated for a UI Record by VisualAge Generator, but they are described here for completeness.

<OPTGROUP>

This tag encloses OPTION tags and therefore groups them together.

```
<OPTGROUP LABEL="?" [DISABLED] >
</OPTGROUP>
```

<BUTTON>

This tag has better three-dimensional representation and more functionality than the INPUT tag technique used to create a button:

```
<BUTTON TYPE="SUBMIT|RESET|BUTTON" NAME="?" VALUE="data sent when the
FORM is submitted, unlike INPUT button definitions"
[DISABLED] [ACCESSKEY="a letter"] [TABINDEX="?"]
>text to display on button in the browser</BUTTON>
```

TYPE="BUTTON" would be used if you wanted to implement some client-side script processing that would be invoked by the button.

<LABEL>

This provides a label to associate with another FORM field in such a way that speech-based or Braille browsers can use the label to prompt the user.

<FIELDSET>

Encloses a group of FORM fields to group them together.

<LEGEND>

Goes inside an enclosing FIELDSET tag to provide a caption:

```
<LEGEND ALIGN="TOP|BOTTOM|LEFT|RIGHT" ACCESSKEY="a letter"  
>caption text</LEGEND>
```

3.3.2 UI Record FORM support

UI Record supports, among others, UI types of HIDDEN, SUBMIT, SUBMIT BYPASS, INPUT, or INPUT/OUTPUT.

If you define any fields these types as the highest level data items in your UI Record definition, VisualAge Generator will helpfully generate a *default HTML <FORM> tag* for you. (It can also do this for a UI type of OUTPUT in certain circumstances.)

If you are using the default FORM, this will typically be because you are displaying the UI Record with an expected return program, so you coded either:

- A CONVERSE process option
- An XFER to a named Web Transaction, passing the UI Record

The FORM tag in the Java Server Page produced by generating the UI Record will look as follows:

```
<FORM METHOD="POST" ACTION="<% UI record name.getGatewayURL() %>"
```

In the case of the default FORM, this ACTION is generated for us to invoke the Gateway Servlet, passing it information of which Web Transaction you wish to go on to, and our thread of conversation with the active Web Transaction program continues.

The FORM UI type can also be defined in a UI Record. In a data item with UI type FORM you can build your own FORM tag with the associated attributes. In the FORM UI type data item you specify which VisualAge Generator program to invoke and what data to pass. This option is for beginning a new thread of conversation with the user. We will discuss this in more detail in "UI type FORM" on page 65.

The METHOD attribute of FORM indicates how the data from the <INPUT> fields is passed. Usually you use POST, as this sends the data separately in the body and includes no length limitations, other than what may apply to the UI Record itself, for example, in CICS environments. An alternative to POST, is GET. This sends the data as an extension of the URL string, by adding a *query string* to it. Data sent this way is limited to around 400 bytes or so (it depends on the Web server).

3.3.3 Creating FORM fields in a UI Record

Various FORM fields can be defined in a UI Record. The types of fields created for a UI Record depend on the data item attributes and UI type. The implementation of each UI type is described below:

UI type = INPUT or INPUT/OUTPUT

- If the data item length ≤ 80 you get:

```
<B>Label UI property as defined for the data item </B>
<INPUT TYPE="TEXT" NAME="VisualAge Generator data item name"
VALUE="Value of the data item"
SIZE="Byte length defined for the data item, plus room for signs,
separators and currency symbols, as required."
MAXLENGTH="Byte length defined for the data item, plus room for
signs, separators and currency symbols, as required.">
```

- If the data item length > 80 you get:

```
<B>Label UI property as defined for the data item</B>
<TEXTAREA NAME="VisualAge Generator data item name" WRAP="virtual"
>Value of the data item</TEXTAREA>
```

- If a *boolean edit check* is added as part of the UI properties of that data item you get:

```
<INPUT TYPE="CHECKBOX"
NAME="VisualAge Generator data item name.occurrence number of data
item"
VALUE="Y or 1 depending on whether the data item type is CHA or a
numeric type">
<B>Label UI property as defined for the data item</B>
```

The data item definition for the field should be CHA length=1 or a numeric data type with no decimal places.

If the data item is CHA, its value is 'Y' if the check box is checked, otherwise it is blank. If the data item is a numeric data type, its value is 1 if it is checked, otherwise it is 0.

Note: The label and value data presented in the HTML is obtained from the UI Record beans and filled in using Java Server Page syntax.

UI type = HIDDEN

```
<INPUT TYPE="HIDDEN"
NAME="VisualAge Generator data item name"
VALUE="Value of the data item">
```

UI type = SUBMIT or SUBMITBYPASS

```
<INPUT TYPE="SUBMIT"  
NAME="VisualAge Generator data item name"  
VALUE="Label UI property as defined for the data item">
```

You must always give a SUBMIT or SUBMIT bypass button a data item value. Either a default value as part of the data item's UI properties, or else move some data to the data item.

The data item value is not displayed in the browser unless you do not fill in a label as part of the data item's UI properties. If there is no label, data item value = HTML VALUE; this allows you to dynamically control the text the user sees on the button.

UI type = OUTPUT

- If the number of occurrences > 1 and the data item representing the *selected index item* in its UI properties occurs once:

```
<SELECT NAME="VisualAge Generator data item name" SIZE="1">  
<OPTION VALUE="Index position of OPTION tag in OPTION tag list"  
>Value of an occurrence of the data item  
</SELECT>
```

If an item is selected from the list, the index of which item is chosen is set in the selected index item. The index starts at 1.

Note: even if there is NO selected index item but the array of data items are inside a data item of UI type FORM, the data items are displayed as a single SELECT list.

- If the number of occurrences (n) > 1 and the data item representing the *selected index item* in its UI properties occurs > 1.

```
<SELECT NAME="VisualAge Generator data item name" SIZE="n" MULTIPLE>  
<OPTION VALUE="filled in using Java Server Page syntax, index  
position of this OPTION tag in the OPTION tag list">filled in using  
Java Server Page syntax, value of an occurrence of the data item  
</SELECT>
```

If items are selected from the list, the indices of the selected items are added to the selected index items. For example, if you selected the first and third items in the list, the selected index item[1] = 1, and the selected index item[2] = 3.

- If the number of occurrences > 1 and the data item representing the *selected index item* in its UI properties occurs once and the data item is itself substructured:

```
<TABLE BORDER="1">
<CAPTION ALIGN="top"><B>filled in using Java Server Page syntax,
label defined for the data item in its UI properties</b></CAPTION>
<TR>
<TH></TH>
<TH>filled in using Java Server Page syntax, label defined for a
sutstructured data item in its UI properties</TH>
</TR>
<TR>
<TD><INPUT TYPE="radio" NAME="name of selected index item"
VALUE="index position of row in table"></TD>
<TD>filled in using Java Server Page syntax, value of a substructured
data item</TD>
</TR>
</TABLE>
```

If an item is selected from the list, the index of which item is chosen is set in the selected index item. The index starts at 1.

- If the number of occurrences > 1 and the data item representing the *selected index item* in its UI properties occurs > 1 and the data item is itself substructured:

```
<TABLE BORDER="1">
<CAPTION ALIGN="top"><B>Label UI property as defined for the data
item</b></CAPTION>
<TR>
<TH></TH>
<TH>Label UI property as defined for a sutstructured data item </TH>
</TR>
<TR>
<TD><INPUT TYPE="checkbox"
NAME="name of selected index item.occurrence number" VALUE="1"></TD>
<TD>Value of a substructured data item</TD>
</TR>
</TABLE>
```

If items are selected from the list, the indices of the selected items are added to the selected index items. For example, if you selected the first and third items in the list, the selected index item[1] = 1 and selected index item[2] = 3.

3.3.4 Match valid edit tables

If you define an INPUT or INPUT/OUTPUT field and apply a match valid table as its edit routine, this will actually be displayed as a single select list in HTML. The selected item is stored in the actual UI Record data item.

3.3.5 Variable lists

Any data item which occurs in a UI Record (other than of type NONE) can have an Occurrences item. You define it as part of the UI properties. It is numeric and has no decimal places.

You can use this to dynamically control the size of the array displayed on the browser. Just set the actual occurs of the data item to the maximum possible, then fill in the value of the occurrences item to control the displayed array size. If you set the occurrences item to 0, then none of the array shows up.

3.3.6 UI type FORM

This allows you to define your own FORM. This represents starting up a new conversation with the user. You can specify the Web Transaction you wish to go to and you can optionally pass data.

The FORM tag you get looks like:

```
<FORM METHOD="POST" ACTION="http://localhost/webapp/janesserverWebApp/  
GatewayServlet?hptAppId=MYPROG&hptExec=Y&hptRecord=UIRECL">
```

There is a query string on the URL (all the data after ?) but it is short. Any other data you wish to pass becomes part of the FORM and is submitted in a separate body, so there are no data limitations other than that of the environment. For example, if you are using CICS, the data passed cannot exceed 32K.

When you ask to pass data you may just name the UI Record of the receiver program. Now any matching data item names between each UI Record are passed.

Because this is a FORM, data items of UI type INPUT or INPUT/OUTPUT which are in that SAME FORM and have matching data item names in the receiving UI Record will have user input into those fields passed across.

It is impossible to pass user input from fields outside this FORM.

Data items of UI type OUTPUT or NONE will NOT have their values passed, even if their names match with data item names in the receiver UI Record; you must add these data items as **Link Parameters**. Specify the receiving

Name and the data item name on this UI Record as the Value Item. Any "Name" you specify in this way is included in your FORM using this HTML tag:

```
<INPUT TYPE="HIDDEN" VALUE="data item value">
```

There are some implications of the above:

- Data items of UI type NONE are not usually visible in the HTML in the browser, even when you ask to view source. But, if wish to pass a data item of UI type NONE, the data item must be added as a Link Parameter, which will cause the data item value to be included in the HTML source (but not on the browser screen).
- End user input into data items in any FORM other than this FORM cannot be passed on by submitting this FORM, even if you match data item names between UI Records.
- End user input into data items in this FORM cannot be passed on if you list that data item in the Link Parameters; you must match up the data item name. When the data item is included in the Link Parameters then the value sent in the original HTML is passed, not what the end user entered in that field.
- Data item validation. Suppose you have edits associated with a data item of UI type INPUT or INPUT/OUTPUT, such as numeric, and you are passing this data item when the FORM is submitted, because you have matched the data item name in the sending and receiving UI Record. The edit checks are not performed until the RECEIVING UI Record receives the passed data. So, if the original field was of UI type INPUT or INPUT/OUTPUT you really ought to either:
 - Define the receiving data item as UI type INPUT or INPUT/OUTPUT (not OUTPUT) with the required edits. In fact, it is not worth defining the edits on the sending data item at all, unless your sending and receiving of the UI Record happen to be the same. The receiving UI Record is doing the validation, and it is this record which is displayed to the user with error messages if any of the received data does not meet the edit requirements of the receiving data items.
 - Not apply any edits (other than perhaps user-written edit routines) to the receiving data items. If you apply user-written edit routines they should not set EZEUIERR if errors are found; set a flag in a data item defined to the UI Record as HIDDEN instead. In the program code you should check the flag and then send the appropriate UI Record back to the browser with error messages for the user to correct. Bear in mind that if you XFER to a named program, it is not possible to send an error

message using EZEUIERR in user-written code outside of edit routines.

- You might wonder how can we pass through information on which push button was clicked on our defined FORM? Well, clicking a push button is just the same as user input, so, you must create data items in the receiving UI Record with the same name as the sending submit data items. Do NOT specify these data items representing the submit buttons on the Link Parameters in the UI properties of the FORM data item. The receiving UI Record must be specified in the UI properties of the FORM data item. The UI type of the receiving data items could be:
 - SUBMIT. If you do this you must specify on each one a label which matches up with the label of the sending data item. You must also assign a default value. Then the default value of the receiving data item which corresponds to the sending data item which was clicked is looked up and that default value is set as the actual value of that receiving data item. All the other receiving data items corresponding to other submit buttons will be blank.
 - Anything else, for example NONE or HIDDEN. When control passes to the receiving Web Transaction the LABEL (not the VALUE) of the button you clicked is passed into the corresponding receiving data item. All the other receiving data items which represent the other submit buttons remain blank.
- If you want to pass selections over, these are input fields, so you need to have matching data item definitions for the selected index item (if you have one) or else the actual selected item in the receiving record. If it is a single selection list and you are defining the selected item in the receiver, it only needs to occur once. If you are using a selected index item, there is no way to pass over the actual selection data list as well, so do not define it in the receiver.

Beware of having several FORMs on the same Web page. If the user clicks a submit button on one of the FORMs, any data typed into input fields on other FORMs will be lost. This is because a SUBMIT button only submits the data on its FORM to the Web Transaction, nothing else gets sent. So if the same UI Record is served back to the browser again, the data input into the other FORMs will have gone.

After reading this section on defining your own FORMs you may be thinking how complicated this is, and you would not be wrong. Much of the above complication arises when the sending and receiving UI Records are different; they do not have to be, they could be the same. If they are the same, the data

validation and transfer problems disappear, unless your user-written validation is split between edit routines and mainline code.

You do not have to create your own FORM, you can use the FORM VisualAge Generator creates for you in conjunction with XFER to a named program passing the same UI Record or else CONVERSE.

3.4 LINKs

VisualAge Generator supports the specification of a UI Record UI type of Program Link.

A LINK in HTML terms is defined by `<A>` tags. It represents an area on a page where you may click and be transferred to another URL or another location on the same page. The area you click may appear as:

- Text, in which case:
 - It is colored differently from the rest of the page, and differently again once it has been "visited"
 - It is underlined

These settings are changeable in the browser itself, and in JavaScript. The actual text you see on the page is the information between the `<A>` tags.

- An image. In this case you put an `` tag between the `<A>` tags.

`<A>` has an `HREF` attribute, which indicates the location to go to when a browser user clicks on the text/image.

`<A>` supports a `TARGET` attribute which describes which window or frame you wish the output document to be sent to. If `TARGET` is set to a name which does not correspond to an existing browser window or HTML `FRAME`, a new window is opened. HTML `FRAMES` allow a single browser window to display several documents at the same time. Each document appears in a `FRAME`. The `TARGET` attribute can be used to interact with a particular `FRAME`.

`<A>` also supports `ACCESSKEY` and `TABINDEX` attributes which we have already seen with FORMs.

3.4.1 LINKs in UI Record definition

When defining a data item of UI type Program Link, you must specify which Web Transaction you wish to invoke when the LINK is clicked. You can ask

for a new window to be opened to display the output from this specified Web Transaction.

This will generate: label specified as part of the UI properties of the LINK definition in the UI Record

The label you specify as part of the UI properties of the LINK definition in the UI Record is what shows up as the link text you click on. If you do not specify a label for the Program Link as part of its UI properties, then the value of the data item is displayed in the browser instead.

When you define the LINK you may ask to pass information. You do this by specifying Link Parameters as UI properties of the Program Link. You must name the receiving UI Record. You name a receiving UI Record data item and a sending UI Record data item. If the link item is an occurred item, the corresponding items in each occurs will be passed over. There is no point in specifying a receiving UI Record unless you are also going to add Link Parameters, this is different from FORMs discussed above.

This data you ask to pass is added as a query string to the end of the HREF when the Java Server Page runs and so looks "hard coded" into the HREF is you view the page source in the browser. An example URL viewed in the source would be:

```
http://localhost/webapp/janesserverWebApp/GatewayServlet?  
hptAppId=MYPROG&hptExec=Y&hptRecord=UIREC1&PASSEDUIRECFIELD=FIELD+DATA
```

There are several implications of this:

- There is a limit on the amount of data that can be passed in this way, a URL string cannot exceed around 400 bytes (Web server dependant).
- The <A> is separate from <FORM>, it does not have to be in a <FORM> and cannot send user input into fields inside any FORMs on the browser side. The only data passable on the LINK would already have to be present somewhere in the UI Record data items on the Web server side when the Java Server Page ran to build the HREF string and send the page to the browser.
- UI Record fields of type NONE can be passed but they are added to the HREF string and so can be seen by the browser user if they ask to view source.

The LINK represents starting up a new thread of conversation with the user, unless both of the following conditions exist:

- What you are clicking on is a program which CONVERSEs or XFERs to a named program.
- The user had previously clicked on that link to start the program and the program had not ended; that is, it had not XFERd to ' ', or EZECLoSd. In that situation, the user is taken straight to where they were in the conversation with that program, rather than spawning a new thread.

You can create images to click on rather than just text by altering the label or by what you move to the data item, if you have not specified a label. Simply type an HTML IMG tag as the label text, for example:

```
<IMG SRC='visage.gif'>.
```

In this case, c:\ibmvjava\ide\program would be the place that this GIF file would be looked for during testing. When you deploy the code, you could then put this GIF in the same place that you are instructed to put the GIFs in 14.2.6, "Deploy JSPs and GIFs" on page 317. You do not have to do this during UI record definition, as you can add whatever HTML you desire to the JSP produced by generation.

If you use the data item value rather than specifying a label on the UI properties you may dynamically change what text the user sees for the hypertext link, or which image you see. If you wish to dynamically change the HREF you would need to use a scripting language, such as JavaScript in the generated JSP.

3.5 HTML layout and look-and-feel

This is really an issue for the HTML page designer not the programmer. The generated JSP can be altered to rearrange fields, add more HTML tags or client-side scripting, such as JavaScript.

You can affect the page layout programmatically by arranging fields in substructures.

For example if you have 2 UI data items as top level items in the UI record they will be display on the browser vertically underneath one another. If you define a single top level filler data item (name= *) which contains these 2 items, they will display on the browser side-by-side.

3.6 UI Record specific features

This section documents features unique to the VisualAge Generator environment.

3.6.1 Record properties

UI Record properties include:

- **Help Text** — Default help text for the entire UI Record.
- **Submit Value Item** — Name of data item in UI Record that will contain the value of the actual submit button that the user clicks.

If you do not specify a submit value item, EZE Aid is used to hold the information.

The EZE Aid/submit value item works with submit buttons, and every submit button must have a value.

You may define this as a default value for the button as part of its UI properties or else move some data to the UI Record data items of UI type submit. If you are using EZE Aid as the submit value item, you must move values which are valid for EZE Aid; that is ENTER, PF1-PF42, PA1-PA3. These values are all upper-case. EZE Aid will default to ENTER if the Web Transaction invocation is not responding to a previously sent UI Record or it cannot cope with the values you have set up in the submit buttons.

Once you have set up the submit button values, then the submit value item receives the value of the submit button you clicked, and you may check it in your code. If you are using EZE Aid, you may code IF EZE Aid IS <response>, just as you could for 3270 programs.

You only have one submit value item even if the UI Record contains many forms.

3.6.2 Special VisualAge Generator UI types

There are two special types:

NONE

Defines that this field is not to show on the user interface and that no edits are to be defined for it. Items with this setting are typically used as control data for user defined edits or as items such as the one defined as the **Submit Value Item**.

NONE fields are not HTML HIDDEN fields. They are not sent to the browser. If you CONVERSE a UI Record or XFER to a named program, these fields are stored on the Web server and their values retrieved when processing continues.

You may ask to pass a NONE field in a user defined FORM data item or in a Program Link data item. If you do this the field is transformed into a HIDDEN field (if its is a FORM) or else appended to the URL string. In this situation then it is sent to the browser and can be seen by the user if they ask to view page source.

NONE fields are useful to store data associated with the UI Record that do not need to be sent to the browser, but this only works when the UI Record data is saved in a session bean by the Gateway Servlet. This only occurs for CONVERSEd UI Records or UI Records displayed in a browser as the result of an XFER Pgm WS, UIRec transfer request.

SubmitBypass

Same as UI type Submit however, when these buttons are pressed, all input edits are bypassed.

3.6.3 VisualAge Generator features for UI Record data items

This section documents miscellaneous features.

HELPTEXT — Item Help Text

Help text defined for the item. This attribute is saved with global data items to facilitate sharing of help text between all records that use the item.

Nothing is actually done to make this help text display. You can tailor the JSP produced at generation time to access this information from the interface bean and combine this with client-side JavaScript so that the information can easily be accessed by the client at runtime.

IF UI Record data item IS ... and SET UI Record data item ...

You cannot set UI Record data items in the way that you can for 3270 map fields, even though DISABLED and/or READONLY are attributes of HTML FORM fields.

If you wish to achieve the DISABLED or READONLY you should add a flag data item to your UI Record for any field you want to protect/unprotect and add JSP syntax to the generated JSP to read this flag from the interface bean and generate the appropriate HTML. Bear in mind DISABLED fields are not sent back on FORM submission; this will impact UI Records displayed as a result of XFER ' '.

If you wish to affect the cursor positioning, you are better off using client-side JavaScript.

Setting colors or other extended attributes should not be done in the Web Transaction code. You may want a mixture of client-side JavaScript and style sheets.

Allowing the user to request a SET map EMPTY can very easily be achieved by editing the JSP produced at generation time and adding an HTML INPUT field of TYPE="RESET".

It is possible to code the syntax :

```
IF UI Record data item IS MODIFIED
```

However, you cannot set the data item modified.

3.6.4 Data Item Edits

The data items in a UI Record can have edits defined:

Input Defines that input can be entered by end user and that edits will be run on the input data.

Output Defines that output edits will be performed on data received from the server.

InputOutput Defines that INPUT and OUTPUT attributes are defined.

Edits can be defined for items in similar fashion as they are defined for Text Map fields. The following edits are designed to implement processing very similar to that provided for Text Maps:

- Input required
- Minimum input (characters)
- Check SO/SI space
- Date
- Time
- Minimum value
- Maximum value
- Zero edit
- Numeric formatting - separators, decimal point, sign
- Folding
- Fill characters - Null is not a valid fill character

The following edits work differently in a UI Record (as compared with a text map edit):

- Currency Symbol — Same as in Map definition; but a 1-3 character currency symbol is also supported.
- User defined function — this is an edit routine which you may ask to be run at the Web server or at the VisualAge Generator server. Functions run on the Web server may not do I/O. The scope of the data accessed by the function is limited to the items in the UI Record.
- Date and Time — There are no specific edits. A flag is set to tell the UI Record that the data in the item is in the form of Date or Time.
- Justification is not supported. Since justification is not supported, fill characters do not work very well.
- Hex edit is not supported.

New edits for a UI Record data item include:

- Boolean — Valid item values are 0 (false) and 1 (true) for numeric items and Y or " " (blank) for character items.

Substructuring is allowed. However, only the lowest layer in the substructure can have edits defined.

Chapter 4. Java Server Pages and the UI Record interface bean API

This chapter discusses the Java API of the interface bean and how this API can be used to enhance the default JSPs generated by VisualAge Generator.

See Chapter 15, "Enhancing the generated JSPs" on page 253 for examples on how the generated JSPs can be enhanced.

4.1 JSP syntax

JSPs are ASCII files, with a .JSP suffix, which contain a mixture of regular HTML tags and JSP syntax. A JSP is taken by IBM WebSphere Application Server and transformed into a Java servlet by a process known as page compilation (see "Java Server Pages" on page 40).

VisualAge Generator currently supports the JSDK 2.1 and JSP 1.0 specifications. The full details of these specifications and associated syntax can be downloaded from the Sun Web site for Java. See the URL:

<http://www.javasoft.com>

All IBM WebSphere Application Server V3 platforms (standard, advanced, and enterprise) support JSDK 2.1 and either JSP 0.91 or JSP 1.0.

IBM WebSphere Application Server V2 supports only JSDK2.0 and JSP 0.91.

While VisualAge Generator may provide support for JSDK 2.0 in a fixpak, the JSPs generated by VisualAge Generator will always be at the JSP 1.0 level. Utilities to help with the conversion of JSP 1.0 code to JSP 0.91 are being considered.

In the remainder of this chapter we will look at the most commonly used features of the JSP 1.0 syntax. The text indicates where syntax is invalid in at the JSP 0.91 level.

4.1.1 Scriptlets

A scriptlet can be inserted in a JSP with the following syntax:

```
<% --- any legal Java code --- %>
```

Here Java code is enclosed in <% %> symbols. At runtime the Java executes dynamically on the server side, nothing of it is seen in the HTML stream sent to the browser once the JSP completes running, except the results of Java statements to print to the special "out" PrintWriter object.

The "out" object is created and made available to the scriptlet as part of the page compilation process of the JSP.

Scriptlets are typically used to code some logic to selectively decide what to print to "out". An example of a very simple scriptlet is shown below:

```
<% if (x == 1) {out.println(EMPNO.getLabel()); } %>
```

Scriptlets may be placed anywhere in the JSP source. What they print will be inserted into the HTML stream at that point.

Valid print operations include: `out.print(some string value)` and `out.println(some string value)`. `println` puts a carriage control character on the end which makes the HTML source look tidy when viewed in a browser. If the user asks to view source, it does not have the effect of HTML `
`.

What you print should be an object of class `String`. You can convert all objects (other than primitives) which are not already `Strings` to strings via a `toString` method. It is up to the object creator to decide whether to provide their own implementation of the method or inherit a `toString` from a class above it in the inheritance hierarchy. (All classes you define in Java extend (or inherit) characteristics from some object, the `Object` class is the root.) If the object inherits a `toString` method, then what you see when running that method might not be exactly what you desire.

Primitives can also be converted to `Strings` by using their class wrappers, for example, an `int` can be converted to a string using a static method of the `Integer` class: `"String myConvertedString = Integer.toString(anInt);"`

We can use the interface bean API documented in section 4.2 to obtain access to the objects which represent data items in a UI Record and to print them.

Scriptlet code all ends up inside the same Java method of the page compiled JSP (the "service" method). Thus objects created in one scriptlet can be referenced by another.

To write scriptlets other than very simple ones requires a basic knowledge of Java.

4.1.2 Expressions

An expression can be inserted in a JSP with the following syntax:

```
<%= --- an object reference --- %>
```

These expressions are a short-hand way of simply printing the object to the HTML output stream sent to the browser. The object is automatically converted to a string for you and then printed, all you need to code is an object reference. Expressions must not contain ";".

The expression `<%= anObject %>` is equivalent to the expression:

```
<% out.print(anObject.toString()); %>.
```

Expressions can be inserted anywhere in the JSP source and their print output is interleaved with the HTML surrounding it. They end up inside the same Java method of the page compiled JSP as scriptlet (the "service" method).

Most dynamic content equates to the insertion of a few expressions among regular HTML and client side scripting.

If you have indexed properties (that is properties which occur more than once) you will need to use scriptlets to code a Java "for" or "while" loop, to cycle through the occurrences. Remember that Java indices start at 0.

There is an IBM WebSphere Application Server V3 specific JSP tag:

```
<tsx:repeat
  index=anIndexName start=startingIndex end=endingIndex>
  . . . .
</tsx:repeat>
```

This JSP tag can be used to build loops without having to code Java. There is even an IBM WebSphere Application Server V2 equivalent tag:

```
<repeat index=ind start=startingIndex end=endingIndex>
  . . . .
</repeat>.
```

4.1.3 Bean tag

A bean tag can be inserted in a JSP with the following syntax:

```
<jsp:useBean id="referenceName"
  class="fullyQualifiedClassnameOfThisBean" scope="?" />
```

This syntax is not valid with the 0.91 JSP specification, although there is an equivalent tag.

The bean tag allows the JSP to reference or even create a Java bean. Once we have the bean we can reference or change its properties.

The bean tag does not have to be empty, it can have a closing tag.

The id attribute defines the name you use to reference this bean elsewhere in the JSP inside scriptlets or expressions.

The scope attribute can equal:

- "session"— which means the bean was stored in the HttpSession object. The generated JSP does not reference the interface bean using session, although the interface bean and the data bean are stored in the HttpSession by the Gateway Servlet unless "XFER ' ', UI Record" was coded.
- "request"—which means the bean was stored in the HttpServletRequest. (This is what the Gateway Servlet does.)
- "page"—which means the bean was stored in the JSP page context
- "application"—which means the bean was stored in the servlet context

Refer to the Sun documentation for more details on "page" and "application", they are included here for completeness.

There is also a "type" attribute, which represents an interface implemented by the bean and can be used instead of, or in addition to, the "class".

In our case a reference to the interface bean for our UI Record for an individual browser user is passed to us by the Gateway Servlet by storing it in the HttpServletRequest object which the Java system provides to the servlet as part of the JSDK architecture. There is a separate HttpServletRequest object for each browser user and this object is passed to the JSP when it is invoked by the Gateway Servlet.

An example bean tag for the JSP generated from the UI Record named JANEUI1 is shown below:

```
<jsp:useBean id="JANEUI1" scope="request" class="my.pkg.JANEUI1Bean" />
```

This tag appears near the beginning of the JSP source.

We could now use JANEUI1 to get reference to UI Record data items, since they are properties of the data bean in expressions. However, this is done for us in the generated JSP, straight after the bean tag, for example:

```
<% VGDataElement TXTFLD = JANEUI1.getTXTFLD(); %>
```

So all we have to do to refer to the data item TXTFLD is to use the name TXTFLD. There are several methods of UI Record data items which are listed

in section 4.2.2, “VGDataElement Interface” on page 81. For example, we could code: `<%= TXTFLD.getLabel() %>` to print the label we defined in the text field’s UI properties to the HTML output stream.

4.1.4 Directives

A directive can be inserted in a JSP with the following syntax:

```
<%@ page aValidDirective %>
```

This syntax describes JSP wide controls.

This syntax is not valid with the 0.91 JSP specification, although there are equivalents to some of the attributes.

Directives used in JSPs generated by VisualAge Generator include:

import

```
<%@ page import="partial package name" %>
```

With "import" you can add Java import statements which will apply to any scriptlets or expressions anywhere inside the JSP. Java import statements are a shorthand to save you having to type out the fully qualified name of the package everywhere when referring to elements inside it.

An example, which VisualAge Generator generates into the JSPs for UI Records is:

```
<%@ page import = "com.ibm.vgj.uibean.VGDataElement" %>
```

errorPage

```
<%@ page errorPage="jspName" %>
```

This allows you to specify a JSP to show if this JSP throws an unhandled exception.

4.2 The interface bean API

The String class is used by all getter and setter methods implemented for data items defined with a User Interface Type of **InputOutput** in the Java Bean generated for the UI Record.

Setters and getters implemented for data items defined with a User Interface Type of **None** return the appropriate Java class for the item.

4.2.1 UI Record Bean Interface

The getter and setter methods implemented for the Java interface bean generated for the UI Record are described in this section.

String getTitle();

Returns title property of UI Record

String getHelpText();

Returns help text property of UI Record

String getGatewayURL();

Returns the gateway URL - used as ACTION in HTML form

String getSecureGatewayURL();

Returns the gateway URL but with HTTPS protocol - used as ACTION in HTML form

String getPageID();

String form of number that uniquely marks the page that is served to the client

String getAppID();

Returns ID that identifies to the gateway the application that the bean is associated with.

String getSessionID();

Returns ID that identifies the current gateway session that will process the submit request.

boolean hasInputError();

Indicates whether or not any item in the record is in error.

VGDataElement elementNamed(String name);

Returns element in bean named name

VGDataElement get<item name>();

Generated get methods for each bean instance

void set<item name>(String value);

Generated set methods for each bean instance

4.2.2 VGDataElement Interface

Methods available in the VGDataElement interface are described in this section.

Enumeration getEditTableValues();

Returns the elements in an edit table associated with an input field.

String getErrorMessage();

Returns the error message associated with the element.

String getGatewayURL();

For items that are not UIType = Program Link, this will return the same value as the UI Record Bean version of this method.

However for elements with **UIType of Program Link** this method returns a URL string that contains all the parameters as defined by the link properties. This string is then usable as an HREF in an **<A>** HTML element.

String getSecureGatewayURL();

Same as getGatewayURL() above but with the HTTPS protocol

String getHelpText();

Returns the help text property of the element.

int getIndex();

Returns the current occurrence of an element which occurs.

String getLabel();

Returns the label UI property of an item. If item is an element of an array, the label defined for the receiver's index will be returned.

String getTextValue();

Returns String value of element with all output formatting on data done.

TableModel getTextValuesTable();

Returns a TableModel of all formatted text values for the receiver occurrences and sub-elements.

boolean hasInputError();

Returns whether or not the element has an input error.

boolean isDisplayable();

Returns *true* if the data item associated with a submit button has a non-blank value.

boolean isEmpty();

Returns *true* if the associated **Number of Occurrences Item** value is zero. If the item is not occurred then this method always returns *false*. If this item is occurred but has no **Number of Occurrences Item** then this method returns *false*.

boolean isSelected();

Returns *true* If the index of the element is a value in the associated **Selected Index Item**.

Enumeration occurrences();

Returns Enumeration of VGDataElements for each valid index (index <= numOccurs item value). If occurs value is 1 then this will be a single element Enumeration.

Enumeration subElements();

Returns Enumeration of VGDataElement that are valid sub-elements (they are not UIType None) of the receiver. Only the lowest level sub-elements will be returned. The index of each sub-element is that of the receiver.

Part 2. Web Transaction design and development

Chapter 5. Web Transaction design concepts and considerations

There are three basic Web Transaction program design options available in VisualAge Generator:

- CONVERSE UI Record program design
- First UI Record program design (single segment) with named program navigation (XFER Program WRec, UI Record)
- First UI Record program design (single segment) with form directed program navigation (XFER ' ', UI Record)

Each of these program design options has different design implications that should be understood before designing and implementing a Web Transaction application system.

5.1 Concepts

Basic Web Transaction implementation options are discussed in this section.

To better understand how Web Transactions work, we will compare them with the typical structure of VisualAge Generator text user interface programs and study the available Web Transaction implementation structure alternatives.

5.1.1 Main Transaction and Web Transaction program comparison

The design of a Web Transaction program presenting a browser interface is similar to the design of a Main Transaction program displaying a text user interface (TUI). Both Main and Web Transaction programs can be implemented using a CONVERSE or Single Segment design option.

A comparison between the CONVERSE model programming, as implemented by a traditional TUI Main Transaction program and a Web Transaction program, is shown in Figure 26.

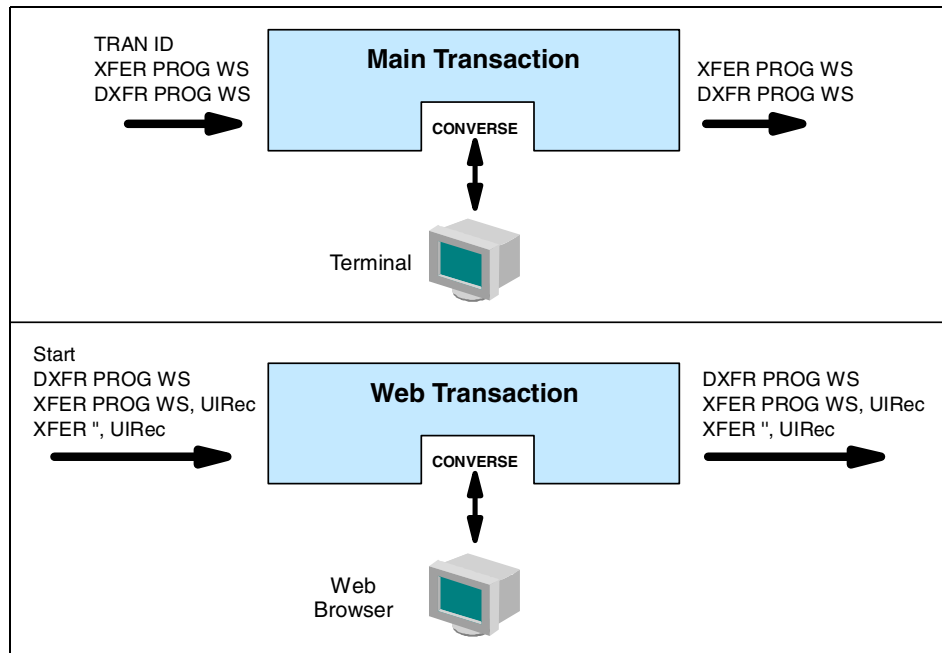


Figure 26. Main and Web Transaction program CONVERSE model comparison

The basic shape of a Main or Web Transaction is the same, but there are some key differences with how the end-user interacts with each type of system:

- In a TUI system, the end-user works with only one screen at a time (modal display processing). The need to save state during end user interaction was simplified because the end-user only interacted with one screen at a time.

When the active screen is displayed, the system resources are released and state is saved automatically by VisualAge Generator, in cooperation with the transaction system in use. For example, state is saved in CICS using temporary storage (a standard CICS pseudo-conversational programming technique).

- In a Web-based system, the end-user working in a Web browser has the capability to open multiple browser windows simultaneously (modeless display processing). Depending on the design of the Web Transaction application system, state may or may not be saved for every one of those browser interactions.

In addition, other processing considerations must be made. If the Web Transaction application system is deployed externally on the Internet (as opposed to an intranet application system), system usage will be far less predictable. For example, the target end-user community for an Internet Web-based system may not be as predictable as that found when implementing an internal TUI based system.

This suggests that system design must consider both if, and how, state will be saved for Web Transaction programs. The processing profile of a Web Transaction will determine the workload that will be placed on the WebSphere Application Server and the VisualAge Generator runtime platform during peak usage and expansion of the end-user community.

5.1.2 Web Transaction state saving options

When designing Web Transaction programs, it is important to understand the issues surrounding the saving of program state. Program state is saved in a variety of ways, depending on the design of your VisualAge Generator program.

The CONVERSE and First UI Record design options provide for three levels of state saving support in a Web Transaction (see Table 1).

Table 1. Web Transaction state saving options

State Saving Options	Command Syntax	Program Design
Complete State	CONVERSE UIREC	CONVERSE with named program transfer navigation
Controlled State	XFER PROG WS, UIREC	Single segment with named program navigation
Stateless	XFER ' ', UIREC	Single segment with form directed program navigation

The choice of how state will be saved on the Web server and VisualAge Generator runtime server platforms will have an impact on the scalability and performance for the entire Web Transaction application system.

To save program state, a combination of session data on the Web server and a work database on the runtime platform is used.

An overview of the basic processing for each type of Web Transaction is shown in Figure 27.

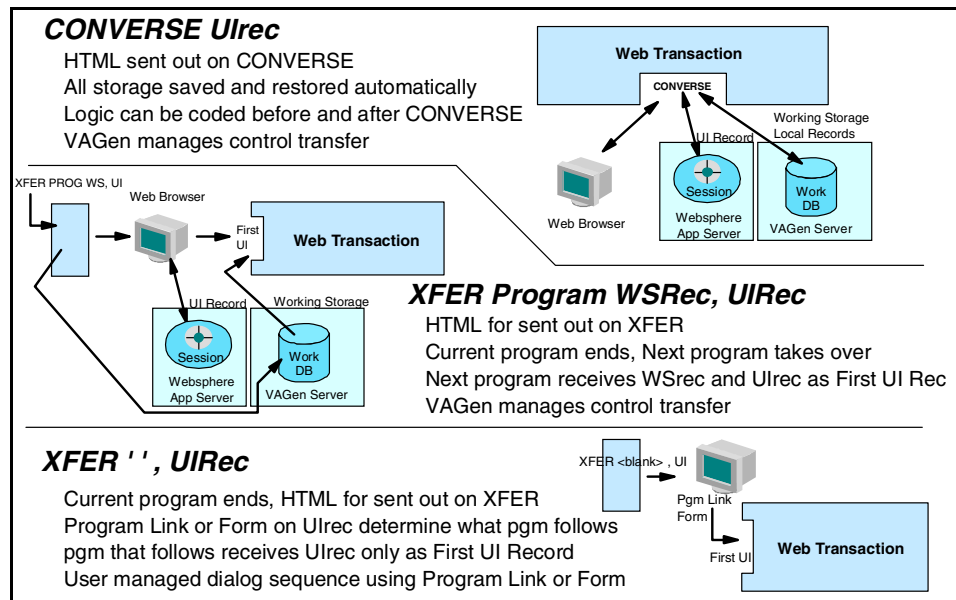


Figure 27. Web Transaction processing for each program type

Note: No data is stored (work database or data beans as session data) for a stateless Web Transaction (XFER ' ', UI Record).

5.2 Program structure options

Web Transaction program structure options are discussed in this section. Table 2 compares Web Transaction program structure and processing.

Table 2. Comparison of Web Transaction program types

Program Type	Program State	Active Session Data Beans	Back Button Response
CONVERSE UIREC	Complete State: Program segmented at point of CONVERSE and all program data stored in work database.	Yes	Reloads current page from active beans.
XFER PGM WS, UIREC	Controlled State: Program segmented at initiation point. Passed working storage record stored in work database.		No ability to back up to a previous cached response.
XFER ' ', UIREC	Stateless: No program active during browser interaction.	No	Loads previous cached response

The choice of Web Transaction program structure affects the design and processing of the complete system. Each option must be understood so that the appropriate choice for the system to be developed can be made.

5.2.1 Using CONVERSE UI Record (complete state)

Program design when using the CONVERSE model is similar to traditional Main transaction programs in VisualAge Generator using maps. This design saves the complete program state during each CONVERSE. The UI Record data is saved in the session bean on the Web server, while the data for the working storage and local records used in the Web Transaction program are stored on the VisualAge Generator runtime server platform. When the end user clicks on a submit button on the Web page, all state data is restored and the program continues processing.

A complete state Web Transaction program is initiated by direct invocation or by transferring control to a CONVERSE program with a working storage record. The syntax is as follows:

```
DXFR Program WSRec;
```

Figure 28 shows how state data is saved automatically during a CONVERSE by a segmented VisualAge Generator program.

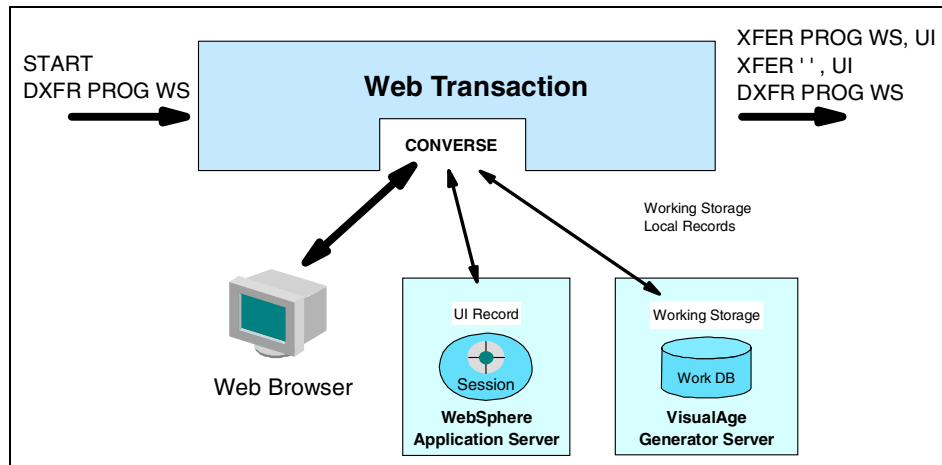


Figure 28. CONVERSE (complete state) program design

The submit button on the UI Record must be part of the default form for the program conversing the UI Record to continue. This means that the submit button must not be a sublevelled item under a parent item with the type of **FORM**. A submit button defined as a sublevelled item under a form item requires a program link definition. This form will specify another program to transfer control to when the button is clicked. Therefore, a submit button that is defined for a form will *transfer control* to another program. This means that the current program that issues the converse will not continue.

This is an important issue to understand from a design perspective. If a program link or submit form is defined in the UI Record, the current program will transfer control to another program. If a CONVERSE design is being used, the program's state data will be saved upon this transfer of control. This could lead to potential performance and capacity issues for the Web server and VisualAge Generator server. This design issue is shown in Figure 29.

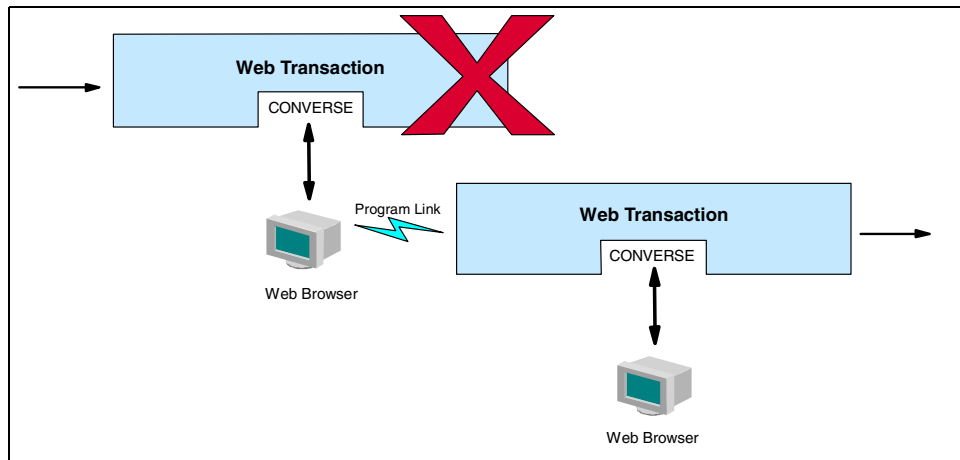


Figure 29. Data lost after program link during CONVERSE

The CONVERSE program design is the most convenient programming option for the programmer because it saves the complete state of the program during a CONVERSE.

However, because of the state implications shown in Figure 29, this type of programming should be used with caution. This is true even though session data will time out, and as session IDs are reused (see 2.3.3, “Session ID Manager (SIDM)” on page 46), work database entries will eventually be recycled.

5.2.2 Using XFER Program WSRecord, UI Record (controlled state)

The program design using an XFER Program model allows the developer some control of the saved state data (content and amount). After the XFER to the target program, HTML is sent to the Web browser; the UI Record and passed working storage data are saved; and the transaction ends. Once the user submits a request back to the Web server, a new transaction is initiated and the passed UI Record and working storage data are restored.

A controlled state Web Transaction program is initiated by transferring control to a first UI program with an optional working storage record and a UI Record. The syntax is as follows:

```
XFER Program WSRec ,UI Record;
```

The UI Record referenced on the XFER must match the first UI Record defined for the target Web Transaction program.

Use of the working storage record is optional. By not using a working storage record during the transfer, you can further reduce the amount of data being stored during display of the UI Record.

During transfer processing, the target Web Transaction program actually starts and controls the work database processing and return of the UI Record data to the Gateway Servlet.

The logic flow of the controlled state program design is shown in Figure 30.

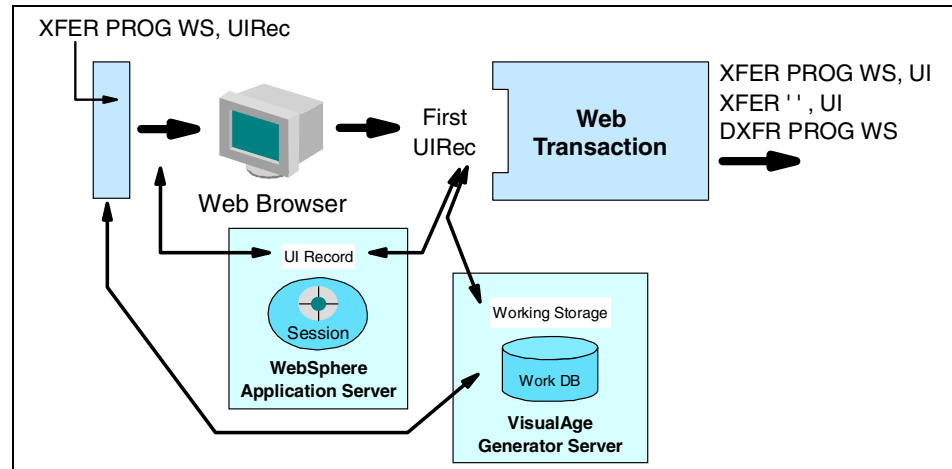


Figure 30. XFER Program WSRecord, UI Record (controlled state) program design

This design option is similar to a CONVERSE in that the program saves state and waits for a response from the end-user. The difference is that the state is saved for only the UI Record, and the working storage is passed on the XFER statement. Therefore, the developer has more control over how state will be saved during each interaction with the user in the browser.

Because some data is saved during browser interaction, the same implications found in the CONVERSE design apply (see Figure 30), if the UI Record has program links defined to start another program directly from the browser. You could also argue that:

- With a controlled state design, less information is saved on the Web server and VisualAge Generator server, because only the UI Record and passed working storage record are being saved.
- The session data used to support the UI Record state will time out and the work database will be recycled over time.

Therefore, stranded state data might not be a serious concern.

5.2.3 Using XFER ' ', UI Record (stateless)

The program design using XFER ' ' UI Record model (stateless) saves no state data for the program or UI Record. Upon sending HTML to the Web browser, the transaction ends. Once the user submits a request back to the Web server, a new transaction is initiated. Other than the data passed to the First UI Record, no other state data is automatically available to the Web Transaction program.

During transfer processing, the Web Transaction program is performing a *send page* to the Web browser. The syntax is as follows:

```
XFER ' ', UI_RECORD;
```

Note: To avoid a possible bug in some runtime environments (GA code) and to remove the question mark (?) from the program structure diagram, this syntax is recommended:

```
MOVE ' ' TO EZEAPP;  
XFER EZEAPP , UI_RECORD;
```

The logic flow of the First UI Record with blank (stateless) program structure is shown in Figure 31.

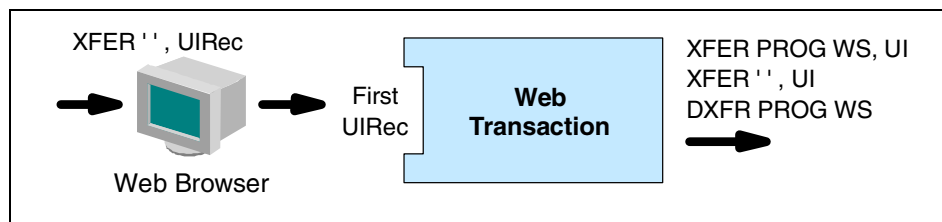


Figure 31. XFER ' ', UI Record (stateless) program design

The program started is identified in the FORM defined for the UI Record. More than one form can be defined; this would allow a program switch to occur based on which portion of the UI Record (which FORM) the end user interacted with in the browser.

There are two ways to save state using the XFER ' ', UIRec program design:

- Save state data in hidden fields on the UI Record in the user browser.
- Manually (specific program logic) save state data, in a database or temporary storage queue, as available on the VisualAge Generator Web Transaction runtime platform.

Developing programs that are truly stateless places more of a burden on the application programmer who must develop programs that satisfy the business requirements. While the programming may become a more difficult task, it will result in an application design that both performs well and is scalable.

5.3 Implementing self-managed state support for XFER ' ' programs

While you can choose to use either stateless (XFER ' ') or managed state (CONVERSE or XFER Pgm) Web Transaction program structures, there are times when you need the stateless structure with some form of state management. When this occurs, you must implement state management in your application logic. This requires that an approach be defined and a data storage technique be chosen.

5.3.1 Introduction

Two basic forms of self-managed state can be implemented:

- **Global state** — Saved State information stored using a common format. Data accessible by any Web Transaction program invoked by a given end user.
- **Conversation state** — Saved state information stored and retrieved by one or more Web Transaction programs is specific, in content and format, to a given conversation (program interaction) with a given end user.

Multiple keys must be used to ensure that end user manipulation of hidden key values passed on forms or program link fields (as program link parameters) is not possible.

There are, of course, variations on these forms of saved state data. We will only investigate the implementation of a limited set of the available options.

Notes: Self-managed state is similar to the state processing implemented with other program structure options:

- The XFER Pgm, WSRec design implements controlled state support using either global or conversation state approach.
Which is used depends on the number and format of the working storage records used as part of the XFER Pgm, WSRec transfer statement.
- While a similar working storage record convention can be applied to a CONVERSE model Web Transaction when a common working storage record is used on the DXFR Pgm, WSRec transfer statement, this is not really state management, in that the passed working storage record is not stored (no end user interaction occurs during a DXFR).

All data in the records associated to the CONVERSE model Web Transaction program are saved during the CONVERSE of a UI Record.

5.3.2 Global state

To implement support for the global state form of self-managed state data, the initial Web Transaction program must be able to generate a key to store data before the Web Transaction completes and identify the key used to store the data when the target XFER ' ' Web Transaction begins.

Our implementation approach for global state management is based on a pair of keys constructed from both predictable and unique elements:

- **EZEUSR** — Unique identifier for current user generated by Gateway Servlet during user logon (explicit or implicit). Available in any active Web Transaction program being used by one user from one workstation.
- **Conversation Start Timestamp** — Created at start of original conversation and used as a blind key.

The blind key can be passed as part of the program link parameters defined in the UI Record for the defined form that starts the target XFER ' ' Web Transaction program. The target Web Transaction program obtains the generated current user identifier from the EZEUSR EZEword. These two values can be used to find any saved state data.

5.3.3 Conversation state

To implement support for the conversation state form of self-managed state data, the initial Web Transaction program must be able to generate a program unit specific key to store data before the Web Transaction completes, and pass the data required to identify the key used to store the data when the target XFER ' ' Web Transaction begins.

A program unit might be one Web Transaction program or a set of tightly related Web Transaction programs that use state data to pass business function control information forward, as required.

One implementation approach for conversation state management could be based on keys constructed from these predictable and unique elements:

- **EZEUSR** — Unique identifier for current user generated by Gateway Servlet during user logon (explicit or implicit). Available in any active Web Transaction program being used by one user from one workstation.
- **Conversation Start Timestamp** — Created at start of original conversation and used as a blind key.

- **Conversation Identifier** — Hardcoded value generated by initial Web Transaction in a program unit to control access to the conversation specific state data.

Parallel use of a program unit should result in the generation of a complete key that is still unique for the current user of the Web Transaction set.

5.3.4 Implementation

The implementation of self-managed state processing is discussed in this section. The design reviewed is used in the state management server program that is used in the Web Transaction state management programming skill exercises (see 8.2.3, “Self-managed state implementation (XFER ’ ’ model)” on page 168).

State data storage

For the purposes of this redbook we chose to use DB2 to store our self-managed state data. This provided us with portability, simple testing using the VisualAge Generator test facility, and ease of implementation.

In a true production environment, this choice might not be appropriate, given the implicit overhead associated with a database management system such as DB2 (when compared with less functional temporary data storage techniques).

The appropriate choice for your environment will depend on the storage techniques available on your target runtime platform. For example, if CICS is your target runtime platform, CICS temporary storage queues, which can be accessed using VisualAge Generator function I/O options, might be a more efficient choice.

Database design

We designed a DB2 table that could support the implementation of both global and conversation self-managed state data. Multiple keys are defined, and a common data area, which can be refined (substructured) in the Web Transaction program. The table definition is shown in Figure 32.


```

CREATE TABLE VGDBA.WT_State (
WTS_EZEUSR          CHAR( 8) NOT NULL,
WTS_KEY_TMSTMP     TIMESTAMP NOT NULL,
WTS_Active_USR     CHAR( 8),
WTS_CRT_TMSTMP     TIMESTAMP,
WTS_DATA           CHAR( 100),
PRIMARY KEY        (WTS_EZEUSR,WTS_KEY_TMSTMP)
)

```

Figure 32. DB2 table definition for self-managed state

The WTS_Active_USR column can be used to either record a user identifier (other than EZEUSR) or to save a conversation identifier value.

Web Transaction program design

A reusable called server program can be implemented to provide save, read, update, and delete access to the state data in the DB2 table.

When using the server program for global state data access:

- The active EZEUSR value is passed to be part of the key.
- When the state data is being saved for the first time, the current timestamp is obtained from DB2 to represent the blind key value. This is stored in the WTS_KEY_TMSTMP column.
- To age the state data (in case it must be purged) the last access is recorded in the WTS_KEY_TMSTMP column.
- The blind key value is returned to the calling Web Transaction where it is kept in a NONE or HIDDEN field in the UI Record.
- The blind key value is passed to the target XFER ' ' Web Transaction as a program link parameter in the defined form or program link.

Data to be saved, along with the processing request (save, read, update, or delete), are passed to the state management server program.

For conversation state management, the calling Web Transaction would have to provide a conversation identifier to isolate the state data from other program units in use by the current end user.

The server program provided to support the self-managed state data exercise only implements global state management functions (see 8.2.3, "Self-managed state implementation (XFER ' ' model)" on page 168).

5.4 Design considerations

Programming decisions, and their implications on implementation options, are discussed in this section.

5.4.1 Data transfer

Considerations for how data can be passed between programs, using a UI Record, are discussed in this section.

Forms

- A default FORM will be created for input fields and submit buttons not defined as sublevelled data items under a defined FORM item.

This type of UI Record is defined for CONVERSE or XFER Program Web Transactions.

- Each explicitly defined FORM item must include a program link definition to identify the target Web Transaction.
- Programs that CONVERSE a UI Record will continue execution after the CONVERSE when a submit button has been clicked on the *default* FORM.

If a submit button in an explicitly defined FORM item is clicked, a transfer of control to the target program defined in the FORM program link will occur.

- A FORM item will pass all items of UI type INPUT, INPUT/OUTPUT, and HIDDEN defined on the FORM.

These items will be passed to the *same named* items of the receiving UI Record for the program receiving control.

If an item does not exist in the receiving UI Record, it will not be passed. In this case, the UI Record acts as a working storage record.

Notes:

- Be cautious when using defined forms and different records on the XFER ' ' UIRec transfer and First UI Record definition for a Web Transaction. If the form names are the same, and the structure defined does not match, sublevelled fields may be passed unexpected data values.
- If you use different UI Records, do not define a form in the First UI Record specified for the Web Transaction.

UI Records and program transfer

- A FORM item will not pass items of UI type OUTPUT or NONE defined on the FORM.

To pass OUTPUT or NONE UI types of items, they must be explicitly defined on the Link Parameters section of the UI Record properties window.

- A FORM can also pass items not defined in the FORM by explicitly defining them on the Link Parameter section of the UI Record program link properties window.
- FORMS are *dynamic* and Program links are *static*.

A form will pass the input and input/output data entered by the end user while a program link hardcodes the data to be passed as part of the link when the HTML is sent.

It is not possible to pass data entered in a field as part of a program link (the program link will pass the data in the field before it has been shown in the browser).

A FORM will pass the data entered into an input or input/output field with the same name in the first UI Record defined for the Web Transaction.

- PROGRAM LINK items must specify Link Parameters to pass data.
- In an XFER ' ', UI Record Web Transaction program you have the option of using the same UI Record to both send and receive data, or to use different UI Records to send and receive data.

Submit buttons

- SUBMIT items defined in a UI Record sent to a browser will show only if they contain data. This data may be defined statically through the *Initial Value* property of the item (Submit tag), or this data may be defined dynamically in the program logic.
- SUBMIT items defined in a FORM must have matching SUBMIT items in the receiving UI Record.

The receiving UI Record will map the label of the button clicked to the SUBMIT item in the receiving UI Record. Then the SUBMIT VALUE will be sent to the defined SUBMIT VALUE ITEM of the receiving UI Record.

The SUBMIT VALUE ITEM can then be interrogated by the receiving program to determine the action requested.

Table 3 shows the techniques and implications for passing data between the Web Transaction and the browser (and back).

Table 3. Data management by UI Type

Web Transaction Type	User Interface Type	In HTML	In Session Bean ¹	In UI Record for Target Web Transaction ¹
Converse UIRec XFER Pgm WSRc, UIRec	Hidden	Yes	Yes	Yes
	Input Input/Output	Yes		
	None	No ²		
	Output	Yes		
XFER ' ', UIRec	Hidden	Yes	Not Applicable ³	Yes
	Input Input/Output	Yes		Yes
	None	No ²		No ⁴
	Output	Yes		Yes
Table Notes:				
1. Data can be modified by Web-based or host-based edit functions defined for the UI Record data items.				
2. UI Type None data items will be in the HTML if they are referenced as a program link parameter.				
3. Session beans are not created for a UI Record after an XFER ' ', UIRecord transfer request.				
4. A None data item can have a value if it is the target of a program link parameter or the destination of a move in an edit function.				

5.4.2 UI Record edits

Items defined as INPUT or INPUT/OUTPUT, if modified, will have input edits defined for the item run upon FORM submission. If the edits fail, the entire FORM will be redisplayed with the corresponding item error messages. If the edits pass, the item is submitted to the Gateway Servlet.

Editing rules

Input or input/output fields are edited in the following manner:

- All fields, if modified, must pass one level of edit before any fields are subjected to the next level of edits.

- If one field fails, the UI Record is redisplayed with the error shown (if the JSP in use gets the message and puts it out, this may have been altered during JSP customization).

Editing processing levels

Several levels of edit processing exist on the WebSphere Application Server platform:

- **Elementary edits** — These include data type checks (such as characters in a numeric field), minimum input, required input, hex, date, and so on. If one data item edit fails, the UI Record is redisplayed.
- **Table edits** — If one data item fails a table edit (range edits, match valid and match invalid), the UI Record is redisplayed.
- **Data item edit functions** (executed on the bean by the Gateway Servlet) — If one data item has an error (identified using EZEUIERR), the UI Record is redisplayed.

Note: There were problems with EZEUIERR processing for XFER ' ' programs in the GA and fixpak 1 level code. EZEUIERR identified problems do not result in a display of the identified error message. We expect this problem to be corrected in fixpak 2.

If all of the fields pass the above edits, the data is passed to the Web Transaction which runs the edit functions defined to run on the Web Transaction runtime platform (as opposed to those edits processed on the WebSphere Application Server platform by the Gateway Servlet).

In the case of a form or program link (XFER ' ', UI Record structure for target program), the input is put into a new bean (other than the bean for the UI Record that is being displayed). The input data is checked by the same process described above. If a field fails any level of editing, the UI Record re-displayed.

Modification trigger

In a TUI environment, identifying modified fields was simple. The terminal device would identify which fields had been modified.

In the case of a UI Record, the Gateway Servlet gets all the data for the UI Record, whether or not it has changed. The Gateway Servlet will identify that the field is modified if data is received for the field and the data is different than what was originally in the field.

The end user may enter data in the field but if the contents remain the same, data item edits will not be triggered. Data item editing options for each UI Type are reviewed in Table 4.

Table 4. UI Record editing options

Web Transaction Type	User Interface Type	Support for UI Type edits	Modification required to trigger edit
Converse UIRec XFER Pgm WSRec, UIRec	Hidden	Yes	Yes
	Input Input/Output	Yes	Yes
	None	No	Not Applicable
	Output	Yes ²	Yes
XFER ' ', UIRec	Hidden	Yes	Not Applicable ³
	Input Input/Output	Yes	
	None	No	
	Output	Yes ²	
Table Notes:			
1. Data can be modified by Web-based or host-based edit functions defined for the UI Record data items.			
2. Output edits are only triggered in an XFER ' ', UIRecord program and should probably be avoided.			
3. Session beans are not created for a UI Record after an XFER ' ', UIRecord transfer request. Because of this, all data looks modified, so all data edits run.			

Using shared data item definitions

The UI Type related specifications for a shared data item will be used when the shared data item is used in a UI Record.

Data item edit functions

Edit function field modifications occur after the bean has been loaded with the data passed back from the browser (input and input/output fields by name and any defined link parameters).

An edit function can alter the input or generate additional data for the UI Record before the Web Transaction is given control.

Note: The UI Record specification and HTML implementation was discussed in detail in Chapter 3, "HTML and UI Record definition" on page 55.

5.5 System architecture considerations

Key architectural issues related to Web Transaction program design, system structure, and runtime system configuration must be identified and understood so that the appropriate decisions can be made during Web Transaction system development.

The approach used to design a Web Transaction system must incorporate the identified requirements for how key architectural issues will be resolved. These decisions will determine how Web Transaction programs will be developed and the configuration of the runtime environment.

An overview of the issues and implementation considerations is provided in Table 5. You should consider the identified issues along with environment specific considerations that apply in your domain when designing a Web Transaction-based system.

Table 5. Web Transaction architecture issues

Domain	Issue	Considerations
Web Transaction program design	Program structure: Complete State Controlled State Stateless	Migration of existing TUI system
		Session data, work database processing, self-managed state processing, and scalability
	UI Record edit definitions and edit routines	Preexisting edit logic may require host implementation of edit functions. Output data is not available for redisplay when a form is returned during an error situation. This may impact presentation processing.
	Error management	Default JSP implementation or alternative, system specific, implementation and management technique.
Existing edit logic may require alternative error management approach.		
System structure	One program type or mixed structures	UI Record definition management
		Recycling abandoned work database entries.
	Transfer techniques	Work database performance and capacity
		Management of long running transaction (multiple page dialog)

Domain	Issue	Considerations
System implementation	User identification	VisualAge Generator login page (Vagen1LogonPage.jsp) or WebSphere Application Server user management.
		Availability of user identifier in Web Transaction program logic (EZEUSR or EZEUSRID).
		Application specific logic for identification.
	Security (authentication and authorization)	WebSphere Application Server user management authentication and authentication.
		CICS-based authentication: Hardcoded, User specific, Trusted (no password).
		Application specific logic.
JSP implementation and customization	Implementation of common architectural changes to default JSP	Customization gets more difficult if the UI Record definitions are constantly changing.
		Style sheets may be effective for some look and feel customization requirements.
	Integration with existing HTML page frameworks	Logon processing must be considered.
		Navigation must target the required Web Transaction programs.
	Interaction with existing servlet-based systems	Understanding the Gateway Servlet API (how target Web Transaction programs are invoked and parameters passed)
		Servlets could invoke the Gateway Servlet to use Web Transaction processing (sharing state data may be difficult).

As you can see in Table 5, we do not state that there is *one best way* to design and implement a Web Transaction-based system. There are trade-offs, external considerations, and existing code reuse objectives that must all be factored into the design process. That said, consider these statements:

- CONVERSE model programs are best when migrating existing TUIs to Web Transactions.
- The XFER Program model provides for both controlled state and solid support for UI Record defined edits and edit functions.
- The best possible scalability scenario is found when using the XFER ' ' model (as long as the recreation of internal working data or self-managed state processing does not take more resources than state management using the controlled state model).

Chapter 6. Web Transaction Web site development

The development of Web Transactions using VisualAge Generator represents only part of the process required to implement a functional e-business Web site. The generated default JSPs must be customized and included in a complete Web site before the system is ready for most end user scenarios.

This chapter discusses front end Web site development considerations for a system that includes the use of Web Transactions implemented using VisualAge Generator.

6.1 Development process overview

An overview of the Web Transaction development and implementation process that will be followed, regardless of how many people participate, is provided in Figure 33.

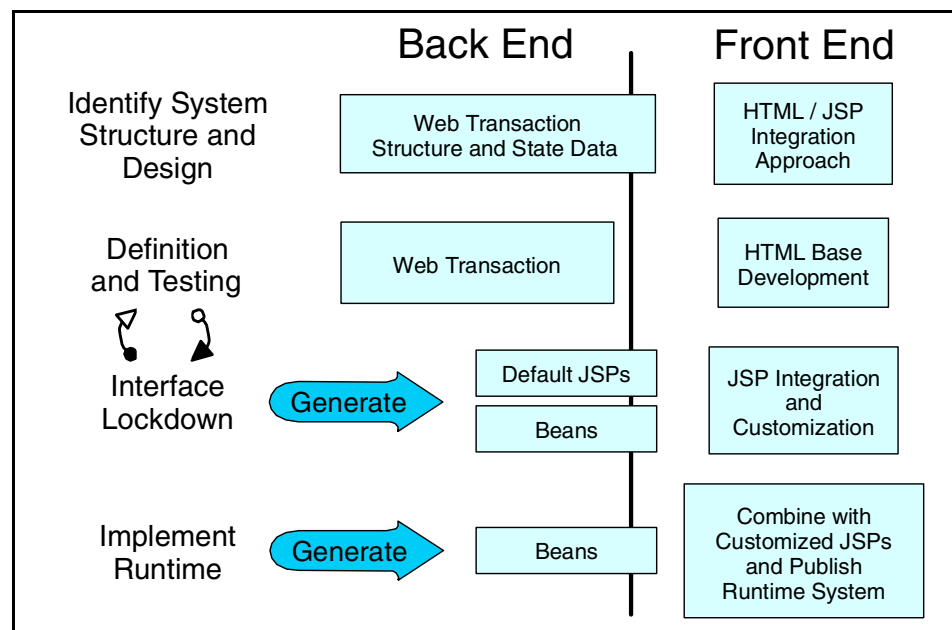


Figure 33. Web Transaction development process overview

The basic process requires that basic architecture decisions be made first. Next, UI Record definitions must be stabilized, followed by generation of default JSPs, customization of the JSPs to fine-tune the user interface, and then final generation and implementation of the runtime system.

6.2 Roles and skills in the development process

The development process that should be followed for turning the default JSPs generated by VisualAge Generator into a presentable Web site can vary widely. The actual process will be based on the skills available in the development team and the desired level of presentation for the finished Web site.

In an ideal world, there would be no required communication or dependencies between the different groups involved in the development process. However, it is impossible to avoid the unique and almost limitless possibilities of Web site architecture, and therefore impossible to lay down one set of rules for attacking the problem of how much each group should know about the others' responsibilities.

If the browser pages seen by the end user are directly based on the generated default JSPs, and these pages are being delivered one by one by a Web server with minor modifications, then there are significantly fewer issues than if the default JSPs are being included in a multi-nested frameset (a common HTML design technique) with appropriate navigation control in the hands of the end user.

The most basic issue involved in marrying VisualAge Generator's default JSPs with modern Web site design is that of scope and context:

- In a "single-serve" Web site design, pages containing all of the essential elements of a user-friendly Web site (clear navigation, predictable content groupings, and so on) are served up one at a time, giving the impression of navigating through sets and subsets of pages.

In reality, this is a linear movement, with each hyperlink spawning a single new page that occupies the whole browser window.

- In a FRAME-based Web site, the initial page (and possibly subsequent pages within it) are broken up into discrete areas called frames, with each separate frame acting as an independent browser window, although they are children to the top-level browser window.

This gives Web site designers the ability to "store" a menu or other information (in the form of an HTML page) in one frame while changing content in another, thereby allowing users to visually see the context in which they are navigating the site.

There are a few things we need to examine more closely in order to better understand the way that the design process will flow. Ideally, the process from start to finish will involve three distinct developer roles (Web Transaction skills, Web site/HTML skills, and JSP skills). However, all skills may not be available, so other alternatives will be presented to deal with these scenarios.

We will also take a short look at the issues involved in session/request scope in JSP programming, and will present some solutions for the problems that result from multiple HTTP requests in a FRAMESET.

6.3 Function and presentation

The convenience and power inherent in being able to publish Web-enabled systems from a 4GL environment is readily apparent. However, system defaults are almost never immediately presentable, and such is the case with the default JSPs generated for Web Transactions by VisualAge Generator.

With JSP source modifications an HTML programmer would consider moderate, we can increase the presentation factor immensely. The look and feel of the generated default JSP is shown in Figure 34.

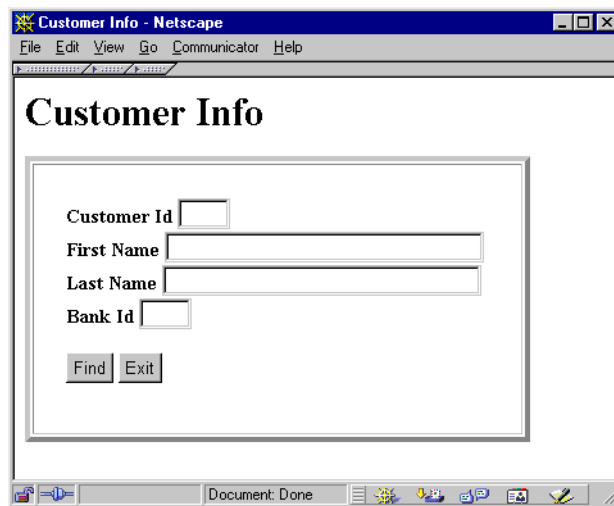


Figure 34. Generated default JSP

The look and feel of the same basic page after some basic HTML design enhancements have been applied is shown in Figure 35.

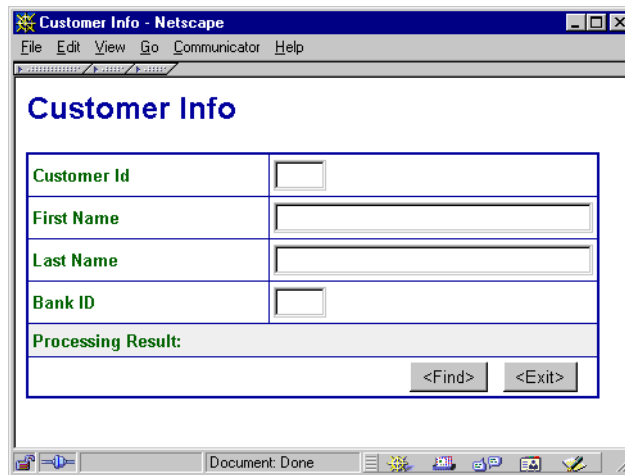


Figure 35. Default JSP after HTML presentation enhancements

As you can see, the changes made during customization are almost a necessity to implement effective presentation, but the presentation enhancements did not change the basic functionality of the JSP (or Web Transaction). This is the main concept to realize: that function and presentation are two entirely separate and distinct entities (although each influences the other in varying degrees).

Functionality refers only to whether or not the system works as it is supposed to work (data management, edit processing, event processing). Presentation refers to whether or not people will be able to efficiently use (or even want to use) the final version of the system.

There are limits to what can be done in the UI Record definition to control the actual presentation in the generated default JSP (see Chapter 3, “HTML and UI Record definition” on page 55 for details). To achieve an acceptable level of effective and efficient presentation, some level of modification of the generated default JSPs will be required. This presents several challenges for the development team:

- Who should perform the modification?
- When should the modifications be made?
- How can the process be managed?

Note: This is the first time that modification of parts generated by VisualAge Generator is not only permitted, but encouraged (if not required). This fact alone represents a code management and ownership challenge to the development team.

6.4 Level 1: the stranded Web Transaction developer

There may be situations in which a Web Transaction development project will not have at least one person from each of the three recommended skill groups (Web Transaction, Web site/HTML, and JSP).

One possible scenario is that one or more Web Transaction developers would be called upon to implement a fully *functioning* (note the emphasis) system. While the Web Transaction development team would have no trouble with creating a functioning system, we should not expect them to have the knowledge required to then transform the generated default JSPs into a Web site of professional quality.

In a level 1 approach (one developer skill set) we can expect the Web Transaction developer to have (or obtain) the knowledge required to perform basic JSP modifications (but nothing more).

The development process to be followed using this level 1 approach is depicted in Figure 36.

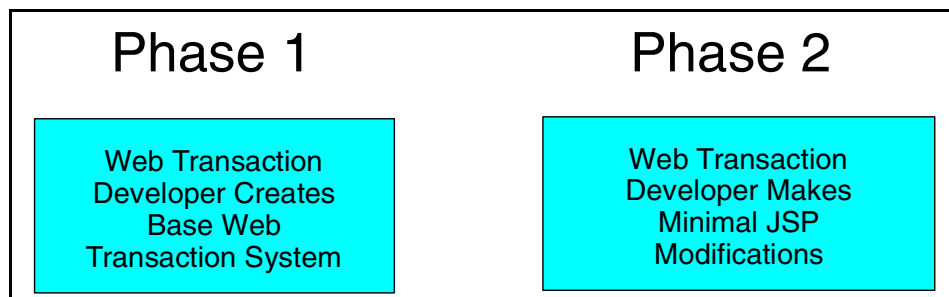


Figure 36. Level 1 development process diagram

A guided exercise showing how level 1 JSP enhancements can be made to the generated default JSPs is provided in 11.1, “Level 1: What’s a Web Transaction developer to do?” on page 195.

The generated default JSPs are created from the UI Record definition, therefore the format of the pages is influenced by the ordering and structuring of the UI Record definition. For example, consider the UI Record definition shown in Figure 37 and the corresponding default JSP shown in Figure 38.

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared
BUTTON2	1	Char	8	0	Submit Bypass	Custom	Nonshared
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared
FNAME	1	Char	30	0	Input/Output	Custom	Nonshared
LNAME	1	Char	30	0	Input/Output	Custom	Nonshared
BANKID	1	Char	4	0	Input/Output	Custom	Nonshared
MESSAGE	1	Char	70	0	Output	Default	Nonshared

Figure 37. Sample UI Record definition

Customer Info

Customer Id

First Name

Last Name

Bank Id

Find Exit

Figure 38. Generated default JSP for sample UI Record

This mapping of UI Record definitions into the HTML created by the default JSP allows the VisualAge Generator developer some measure of control over the basic presentation constructs. However, there are numerous *small* HTML modifications that a VisualAge Generator developer can make to further enhance the look, feel, and usability of the final JSPs.

6.5 Level 2: Web Transaction developer and JSP developer

The second level of Web Transaction Web site development is the addition of a JSP developer who has an understanding of how a VisualAge Generator Web Transaction system works, and has moderate skill in HTML.

In a level 2 approach (two sets of developer skills) the JSP developer will be responsible for making modifications to the generated default JSPs to satisfy the system "look-and-feel" specification, and possibly the development of a simple Web site, as well as the integration of the Web site and the customized JSPs. See Figure 39.

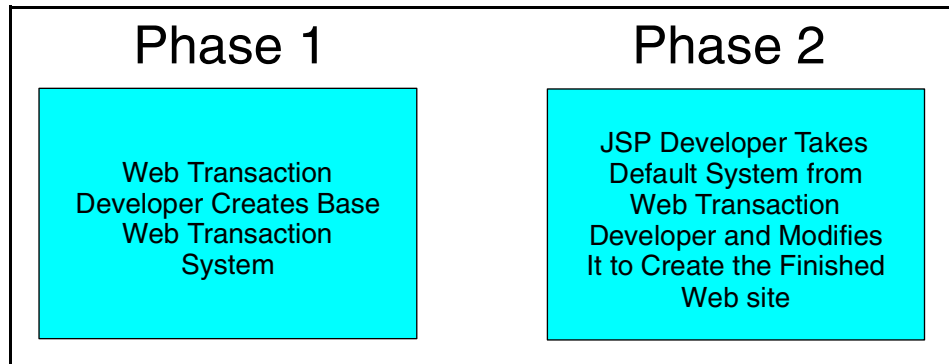


Figure 39. Level 2 development process diagram

These individual methods for development will be reviewed more thoroughly later on, but it is necessary at this point to say that there are certain advantages and disadvantages to each method, and these need to be weighed carefully with the JSP developer's skill level in the areas of HTML, HTTP, and JSP programming.

A guided exercise for how level 2 JSP enhancements can be made to the generated default JSPs is provided in 11.2, "Level 2: Enter the JSP developer" on page 205.

Web Transaction developer

The Web Transaction developer will have the same basic responsibilities as with level 1, except that now the presentation design and implementation phase of the project will be handled by the JSP developer. This allows the Web Transaction developer more time to concentrate on the definition and debugging of the system in VisualAge Generator. That said, it is still the UI Record definition created by the Web Transaction developer that determines the basic format of the user interface implemented by the generated JSP.

JSP developer

The JSP developer will be responsible for design and implementation of the front end system. The user interface and interaction defined for the Web Transaction system must be fully complete (or nearly so) in order for the JSP developer to begin to deploy it.

After the Generator developer has completed the system and generated the Web Transaction JSP pages, the JSP developer will assume control of further development of the Web site itself.

The types of customization required will determine how much modification the JSP developer will have to do to. For example, we have already seen the results of a customized HTML design applied to a default JSP (see Figure 35 on page 108).

There many ways to set up the Web site, including creating a simple "dummy" system with static data which is then replaced with the JSP data (scriptlets, expressions, and so on) that is generated as a part of the default system. Another option would be to create the Web site by simply modifying the existing pages into what is required; in most cases, this will be the preferred method for level 2 development. This will require that navigation and process control be implemented as part of the UI Record definition (program links) and the Web Transaction program logic.

6.6 Level 3: Web Transaction developer, JSP developer, HTML designer

The most efficient use of time and talent will be accomplished by dividing up the Web site production between three roles: Generator developer, JSP developer, and HTML designer. In this scenario, the Generator developer would be able to develop the Web Transaction system concurrently with the development of a "dummy" Web site by the HTML designer.

After both are completed and in working order, the JSP developer would take over both sets of finished code and combine them together to form the finished product. Ongoing dialog between all three levels will be necessary, even during the final phase of JSP development. See Figure 40.

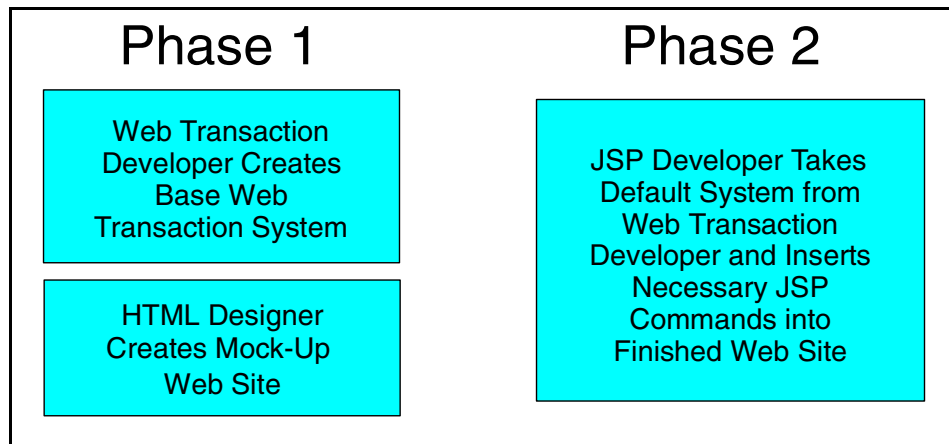


Figure 40. Process Diagram for Level 3 Development

A guided exercise for how a level 3 development scenario works is provided in 11.3, “Level 3: Integrating Web Transactions into a Web site” on page 226.

Web Transaction developer

The Generator developer’s role in level 3 development will be the same as it is for level 2 development. The Generator developer’s main responsibility is the development and testing of the default Web Transaction system.

The advantage of a level 3 development approach is that the Web Transaction developer will be able to develop the system without worrying too much about the JSP developer and HTML designer being forced to wait for the finished product. Web Transaction development can take place simultaneously with the HTML development (as shown in Figure 40).

HTML designer

It has already been mentioned that the main advantage to this three-role development process is the ability of the Generator developer and the HTML designer to develop simultaneously. The HTML designer in this situation will do what HTML designers do best: develop a great-looking Web site!

The realm of Web development this book is concerned with is text-based HTML; these concepts do not apply to the more adventurous current methods of Web development such as vector graphics animation (Macromedia Flash, Macromedia Shockwave), until such time as these technologies make it easily possible to incorporate JSP statements and dynamic data.

With Level 3 development, there is an opportunity to create the best-looking Web site possible due to the increased focus on presentation and the inclusion of a third role to address this in its entirety.

The reason the addition of this role is recommended (and the reason that Level 3 development is the most efficient) is because it allows the developers filling all three roles to remain *precisely* within their respective areas of expertise. In this Level 3 scenario, the Generator developer develops only in Generator, the HTML designer develops only HTML, and the JSP developer only has to do JSP tasks (such as the replacement of the static placeholders in the mock-up Web site with the actual JSP commands from the default system).

JSP developer

The JSP developer is not actively involved in development for Phase 1 of a level 3 Web Transaction project. While it may be that the other two roles will need questions answered and other information discussed, by and large the JSP developer will have other tasks to be concerned with, such as setup of the JSP deployment environment (WebSphere Test Environment, WebSphere Application Server, Web server).

When the HTML designer and the Generator developer are finished, it will be the responsibility of the JSP developer to take the respective finished products and combine them.

Chapter 7. Transforming TUIs into Web Transactions

With the introduction of the UI record capabilities of VAGen V4, existing Cross System Product (CSP) or VisualAge Generator Text User Interface (TUI) programs can be transformed into Web Transaction programs. This capability was designed to maximize the reuse of the existing business logic and therefore retain the investment made in current systems.

In this chapter we review these considerations and provide information on the processes that should be followed to perform this transformation. This is not a migration; there will be coding changes required to the logic in the Web Transaction program.

Note: This information is based on an article previously published in the VisualAge Generator Newsletter.

7.1 Considerations

To get started, you must first analyze carefully what you intend to accomplish by making the current TUI system function available from the Web, and decide whether you intend to add any additional functionality to the Web Transaction.

For example, suppose you strictly need to quickly get the TUI transactions on the Web, you do not necessarily care about utilizing the Web UI functionality, and you would like not to touch the code. In this case, you may want to consider using a screen scraping implementation such as CICS Host-On-Demand, CICS Web Interface with 3270 Bridge, or NetCICS. For more information see:

<http://www.ibm.com/software/ts/cics/library/whitepapers/cicsweb>

These alternative approaches would allow your CICS TUI transactions to become immediately available on the Web. You may even consider this as a preliminary entry into the world of Web-based application systems.

However, if you would like to take advantage of the usability features of the Web such as radio buttons, drop down lists, forms, and program links, then you may want to consider using the VAGen V4 UI record feature to transform your TUI application into an e-business Web Transaction.

7.2 Phase 1 — analysis

This transformation effort should be approached as any other new Web development. First, you should have an analysis phase to identify the functionality to be included in the Web application.

The main purpose of this step is to determine what functions in the existing code are to be retained and to define what is required for screen navigation. This step can save converting unneeded program functions and serve to clarify end user interface components.

For example, a screen that displays a selection list of 20 codes or records and has forward, backward, right, and left scrolling, could be presented a Web page that just does forward scrolling and handles the selection of a record. The code that handles the back, left, and right functions would be eliminated. A small (<100 items) single field selection list can be better presented as a drop-down list, eliminating even more code. Additional consideration will need to be given to how many items are shown in the list and how a selection in the list is made (radio button versus program link).

7.3 Phase 2 — basic transformation

Once you have identified the functions and code that will be used to implement a Web Transaction, you can begin the transformation.

The following list of tasks outlines the process of transforming a TUI program into a Web Transaction program with a Web-based interface.

Prepare the code to be transformed

- Import your code into VAGen V4 using the migration tool or the import function (refer to *VAGen V4 Migration Guide*, SH23-0267 for more information). There are additional considerations if you are moving from CSP 3.3 or earlier to VAGen for the first time, since going from an interpretive execution to generated COBOL may cause additional changes to be made to your source code (see *Migrating CSP Applications to VAGen*, SH23-0244).
- Test using the TUI base code to ensure that all components are present and accounted for, as well to establish that the development environment is functioning properly. You can choose to implement the TUI base code at this time if this is your first VisualAge Generator implementation.

- Establish naming conventions and library organization (Projects, packages...) for your transformed parts and also for the new part types like the UI record and Web Transactions. Refer to *Guide to Migrating MSL's to ENVY*, SH23-0252 and the *VAGen V4 Migration Guide*, SH23-0267.

Convert the map to a UI Record and replicate code

- For each user map in the VAGen parts browser, choose the map, then right mouse click and select ***Create UI Record from Map***. This will create a new VAGen UI Record with fields for each named map field (variable field). Note that any label text or constant fields will not be included in the UI record.

The default label text will be the field name. Pay special attention to follow your naming standards for the new UI record name. It is recommended not to exceed the length of the name of the old field. For example, if the original map was named XY01M02 then a good name for the new UI record would be XY01U02. The reason for this suggestion is so you can more easily change the ESF syntax.

- Export the program using ***VAGen Export with Associates***. Using your favorite text editor, globally change the map name to the UI record name where it is referenced in the code, change all occurrences of EZEMSG to EZMSG, and optionally, change other component names such as program name and function names (change the prefix to avoid conflicts with existing TUI parts). Good naming standards will simplify this step and reduce duplicate parts if you have both the old TUI program and the new Web Transaction in your workspace at the same time. Once the ESF changes have been made, import your new program from the changed ESF file.

Note: Once you are more comfortable with the Web Transaction programming model, you will be able to identify a level in your program structure where TUI logic may be reusable. For the purposes of an initial transformation, a complete replication of the existing code is recommended.

Implement UI Record processing using the Converse model

- Change the program type to Web Transaction.
- Assure that any CONVERSE functions are using the new UI Record. If you have several maps (header, detail, trailer) that are DISPLAYed prior to the final CONVERSE, then all the maps need to be merged into one UI Record. Popup maps will have to be considered separately. Using a drop-down list or other UI functions may serve to replace their purpose.

- Add labels to each Input/Output field. UI Record fields can be customized to add table, function, and other edits, as well as help text. Table edits will appear as drop-down lists. For function edits that are not doing I/O or server calls, then you have a choice where they are executed: on the Web server or on the host server. Help text entered in for each field is included in the generated Java bean and can be referenced in the JSP. This is not referenced in the generated default JSP (see 15.1.3, “Implementing help” on page 255 for a hint on how this help text can be used).
- Add submit buttons to your UI Record for each function key processed and the ENTER key as well. You can set the initial value for each button to the EZEAIID equivalent value (ENTER, PF1, PF3...); however, if your program executes a `SET uirecord EMPTY` this will clear the initial values. If submit buttons have no value, they are not displayed. So set the submit values prior to the CONVERSE (see Figure 41).

```

MOVE 'PF3' TO EXIT-BUTTON;
MOVE 'ENTER' TO SUBMIT-BUTTON;

***----- CONVERSE UGDETBMAP1 -----***

IF EZEAIID NOT PF3;
...

```

Figure 41. Setting submit values before CONVERSE

Transform existing logic

- Statements that SET or TEST the TUI map item's attributes are not recognized and will cause errors to occur. For example, SET map item PROTECTED will be in error. These statements will need to be commented out or removed.
- How fields are displayed or hidden is left to the UI designer. In order to make this determination, additional data will need to be passed in the UI Record.

For example, if your map has certain fields that are updatable depending on user security, then that user security information will need to be included in the UI record. The UI designer can then change the JSP to access the security data passed and set the display or hide fields appropriately.
- Testing a field for MODIFIED is allowed, but setting a field MODIFIED is not. This may also have an impact on your security or edit processing.

- Add a title to your UI Record by editing the properties for the UI record itself.
- Change any date fields that are defined as numeric and have a date mask to be a character field (length 10). Then you must move EZEDTELC into this field if you want the date to be formatted.
- For map array fields that have been converted to items with occurrences, you need to add a counter item to the UI record to hold the number of elements to display in the list when the UI record is conversed. Then, in the custom settings for the item with occurrences, you need to specify the counter item as the occurrences item.

Test your program

- When a CONVERSE is reached, the browser is displayed with your UI record contents (see Figure 42). You may need to further refine the code and UI Record changes to complete the required baseline function.

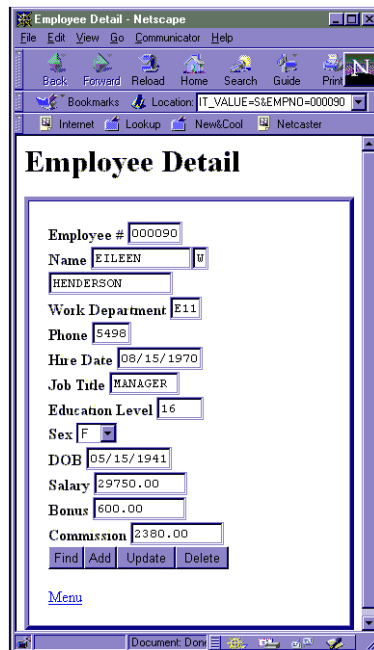


Figure 42. Conversed UI Record

Other things that may need special consideration in this new environment include:

- **Security** — Use of EZEUSR or application-specific security for read-only access versus update authority.
- **Attributes** — Setting of extended or highlight attributes — how this will be identified in the UI Record. You could use a separate attribute field for each field, or a general one if possible.
- **Messages** — If you are using a Message Table, references to EZEMNO and EZEMSG will need to be changed to EZEUIERR. One advantage of this new environment is for multiple fields in error; all field level error messages are displayed below the field rather than one at a time in EZEMSG. General or informational messages like “Key selection and press enter” moved to EZEMSG will need to be moved to a UI record field like EZMSG instead. Be sure the text of the message still applies to the Web interface.
- **Error checking** — If you currently have edit functions specified in your map, these will be included in the UI record you create from the map. Additionally, you can specify where the edits are to be performed on the Web Server or on the VAGen Server. However, if you SET map fields as MODIFIED to force the edit routine, then you will need to change the program to always execute the edit routine after the CONVERSE.
- **Navigation** — Consider uses of menus, “fastpath” fields, or other navigation. This function may be better implemented by the Web designer.

7.4 Phase 3 — make it more Web-like

At this point your application will still look very text-like, even though it is displayed using a browser. To better utilize the usability benefits of the Web, the UI record and Web transaction can be enhanced to make the navigation and internal functionality more Web-like. Some additions may include:

- **Using Program Links as a record selection** — In the UI Record, change a table column to be a type of program link, then customize it to specify the program and the first UI Record that will be passed. Be sure that only key information is passed in this manner, because there is a limit to the amount of data that can be passed (around 400 bytes). The linked program must be prepared to handle the First UI record information and use the key information to read the detail record. The creation of a program link is shown in Figure 43.

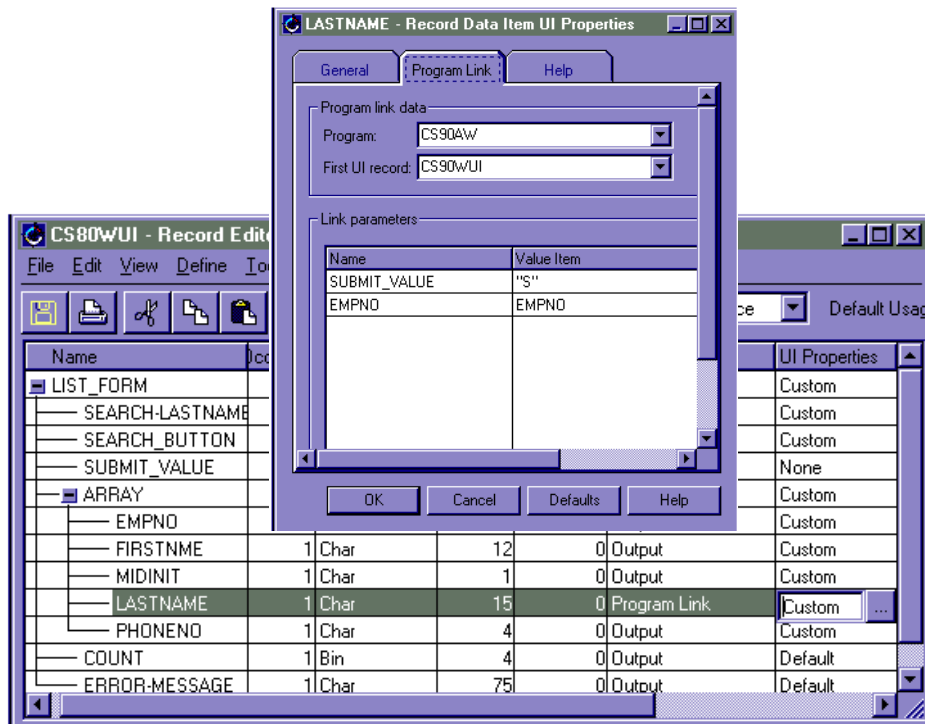


Figure 43. Adding program link

Figure 44 shows a typical UI Record with program link.



Figure 44. UI Record with program link

- *Use forms to combine multiple Text screens into one UI Record.* Because you are no longer limited to a 24x80 screen, you can possibly improve user productivity by retrieving more information on each request. Work with your Web designer to identify what would be best for your users.
- *Consider using the XFER stateless model for selection lists and inquiry-only programs, and stay with the CONVERSE model for maintenance (Insert, Update, Delete).* To implement the XFER model, divide your single application into two programs. Everything prior to the CONVERSE is in the first program, and everything after the CONVERSE is in the second program. Each program would use the same UI Record. The UI Record would have a parent field with UI Type of Form that links to the second program. Instead of the first program doing a CONVERSE, it would XFER ' ',UIRECORD. Once the user selects a submit button, the second program is invoked and will evaluate the request and continue.

For additional information on program structure options, see Chapter 5, "Web Transaction design concepts and considerations" on page 85.

Some code is so intertwined with the map attributes and other settings, or so heavily modified, that it may be faster to use VisualAge Generator Templates (see <http://www-4.ibm.com/software/ad/visgen/about/v4temp.pdf>) to recreate the functionality of the program. This is especially true for simple (not much business logic) programs. For example, search for a record based on selection fields, then display a selection list that has a program link column to display or maintain the detail record.

7.5 Phase 4 — modify default JSP

Up to this point you have only seen the default Java Server Page (JSP) displayed in the Browser. To make the user presentation more Web-like, this JSP will need to be enhanced with graphics and incorporated in existing Web pages using Web authoring tools such as WebSphere Studio. The UI designer can use the default JSP generated as a base or guide to create the customized JSP.

See Chapter 11, “Front-end customization techniques” on page 195 for additional information.

Part 3. Web Transaction programming and front end customization

Chapter 8. Developing Web Transaction programming skills

This chapter contains a set of scripted definitions and test exercises that will help you understand how the Web Transaction program and the User Interface Record (UI Record) support the implementation of a Web-based application system. Two key areas will be studied in the exercise set:

- The different Web Transaction program structure options that are available.
- How state data can be managed using UI Records and the different program structure options that exist.

We will create a series of simple Web Transaction programs that provide read and update access to database information. A set of pre-existing VisualAge Generator database server programs (called batch) will be used to access the Customer table found in the ITSOBANK database. You will:

- Define and test multiple Web Transaction programs.
- Understand how UI Records are used to implement a Web page.
- Customize the properties of UI Record items to define the user interface view and function.
- Use the edit capability implemented for UI Record data items.
- Implement multiple Web Transaction program structures.
- Implement self-managed state data access.

Note: For information on the ITSOBANK database, see A.3, “Database” on page 367.

For additional information:

- See Chapter 5, “Web Transaction design concepts and considerations” on page 85 for a discussion of concepts.
- See Chapter 10, “Demonstration system” on page 181 for a detailed study of the options.

8.1 Program structure

As described in 5.2, “Program structure options” on page 89, there are multiple structure options available when implementing a Web Transaction. Several Web Transaction program structure implementation exercises are provided in this section.

8.1.1 Loading code base

To start this exercise, you need to have the startup version of the *Web Transaction System* added to your repository and loaded in your workspace.

The code provided for this exercise was shipped with the redbook. See A.1, “VisualAge Generator code” on page 365 for details.

To load the code:

1. From the Workbench, choose the *Selected -> Import...* Workbench menu option. Select *Repository* as the import source and click on the *Next >* push button.
2. Choose Local repository as the import option and use the Browse push button to find the file:

```
<localdrive>:\WebTDisk\VAGen\WebTran.dat
```

3. Select the Project radio button and use the Details... push button to import:

```
VGv4 Redbook WebTran Programs - Start 2.4
```

Select the *Add most recent project edition to workspace* toggle to automatically load the imported project. If you did not do this, use the *Selected -> Add -> Project...* Workbench menu option to add the imported project versions to the workspace.

Notes:

- The completed answers to these exercises are in the WebTran.dat file. Load version *Labs Done 8.2.3.c* of the *vgv4.web.codebase* package.
- This first set of exercises uses a server program (CUSTPGM) that accepts a UI Record as the passed parameter. We found during later testing that this architecture does not function at runtime (a bug). We fixed one program set in our provided solution (CSTCNV->CUSTPGM). To get this fix, load the version *Labs Done 8.2.3.d* of the *vgv4.web.codebase* package.

8.1.2 Converse model programming

We are going to define a Web Transaction program (**CSTCNV**) that uses a CONVERSE to prompt the user with a Customer Info Web page implemented using a UI Record (CUSTUI).

The structure of this system is shown in Figure 45.

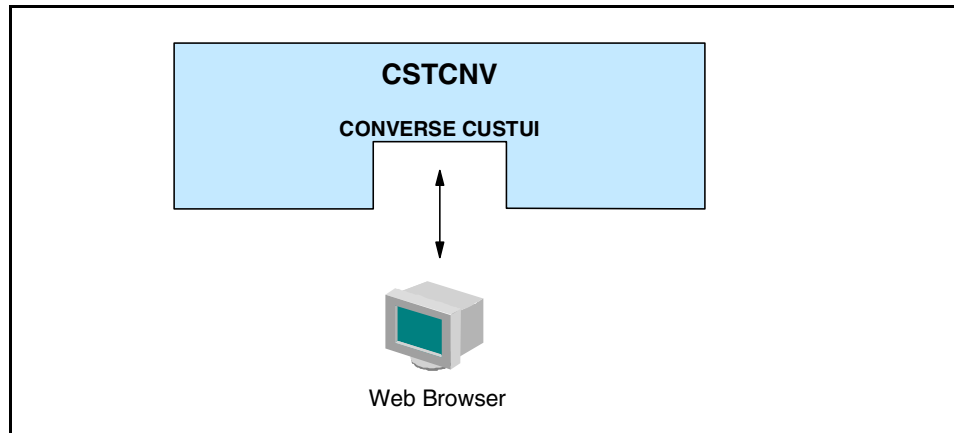


Figure 45. *CSTCNV Web Transaction program structure*

This Customer Info Web page allows the user to enter a customer ID and click a FIND button to invoke a server that searches the database to find the customer record. The record data is returned if found, otherwise an error message is displayed.

We will use UI Record input edits to verify that input error checking takes place before invocation of the server.

In the scripted exercise, we will define a UI Record (**CUSTUI**) that will be used by a CONVERSE model Web Transaction program (**CSTCNV**). The UI Record definition must implement the user interface shown in Figure 46.

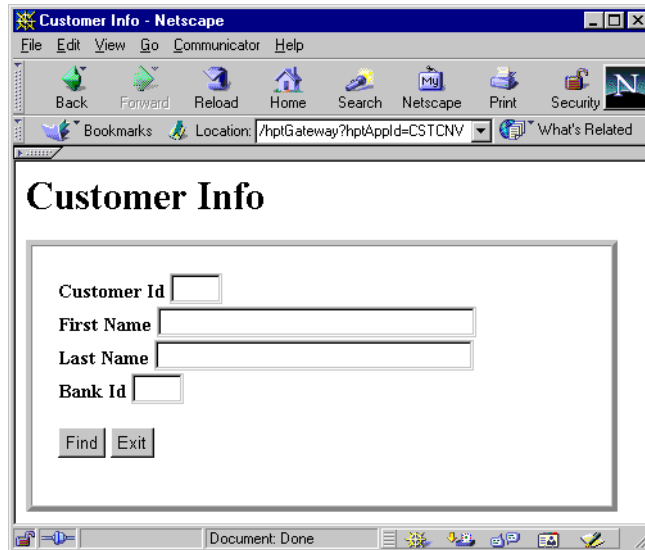


Figure 46. Customer Info Web page

Create Customer Info UI Record

1. The first task is the definition of the UI Record **CUSTUI**. This is the UI Record that will define the basic properties of the user interface that will be implemented in a Web browser. To create the **CUSTUI** record:
 - Open the VAGen parts Browser and add a new VAGen part.
 - In the New VAGen Part dialog, enter **CUSTUI** for Part Name, select **Record** for Part Type, and select `vgv4.web.codebase` as the package for the part.
2. Select **User Interface** in the record type definition drop down list.

- Next define all the data items, with the identified UI Type choices, as shown in the Record Editor view in Figure 47 (initially the UI Properties column will say default; we customize these properties later).

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage
BUTTON	2	Char	8	0	Submit	Custom	Nonshared
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared
FNAME	1	Char	30	0	Input/Output	Custom	Nonshared
LNAME	1	Char	30	0	Input/Output	Custom	Nonshared
BANKID	1	Char	4	0	Input/Output	Custom	Nonshared
MESSAGE	1	Char	70	0	Output	Default	Nonshared

Figure 47. CUSTUI UI Record definition

- Define the Web Page title:
 - In the record editor, choose the *Define -> Properties* menu option. This will bring up the Record UI Properties window.
 - Enter **Customer Info** for the UI title in the *General* tab. This will display **Customer Info** as the title for the Web page.
 - Identify **BUTTON-VALUE** as the Submit Value item using the record properties dialog. This identifies the data item that will identify end user interaction decisions (which button is clicked in the browser).
 - Close the properties window.
- Save the **CUSTUI** record.
- Define the label text that will be shown for the **CUSTID** data item.
 - Click the *more* button for the *UI Properties* of the **CUSTID** data item. This will bring up the UI Properties window.
 - Enter **Customer Id** as the *UI Label* shown on the *General* tab page.
- Repeat the label text definition task to define meaningful label names for the **FNAME**, **LNAME**, and **BANKID** data items.
- Define the contents for the UI Record Submit Value Item.
 - Click the *Default* cell for the *UI Properties* of the **BUTTON** item. This will bring up the UI Properties window.

- Enter **Find** and **Exit** on two separate lines in the *UI Label* multi-line edit. This will display **Find** as the label for that submit button.
- Next click the *Submit* tab and type **FIND** and **EXIT**, again on two separate lines for the *Initial value*.

This allows buttons to be displayed in the browser and defines the response value to the Web Transaction program (we will explore this in more detail in a subsequent exercise).

9. Save the **CUSTUI** record close the record editor.

Create Customer Info Web Transaction

1. The next task is the definition of the **CSTCNV** Web Transaction program.
 - Open the VAGen Parts Browser and add a new part.
 - In the New VAGen Part window, enter **CSTCNV** for Part Name, select **Program** for Part Type, and select *vgv4.web.codebase* for the Package. Click on **OK**. This will open the program editor.
2. Define program main function.
 - Select Web Transaction as the program type.
 - Use mouse button 2 on the Structure Diagram and select the *Insert Main Functions...* menu option. Enter **CSTCNV-MAIN** as the part name.
 - Open the **CSTCNV-MAIN** part (select *vgv4.web.codebase* as the target package for the new part).
 - Converse the **CUSTUI** UI Record (select **CONVERSE** as the I/O option and **CUSTUI** as the I/O object).
3. Define the function logic required to call the server program as part of a loop around the Converse of the UI Record. The loop should continue until the Exit submit request is received.
 - Exit submit request is defined as when the data item **BUTTON-VALUE** equals the text string *EXIT*.
 - The pre-defined server **CUSTPGM** is called using the **CUSTUI** UI Record as a parameter. The server program either returns data, if the customer ID is found, or returns an error message in the **MESSAGE** field.

The required function logic is shown in Figure 48.

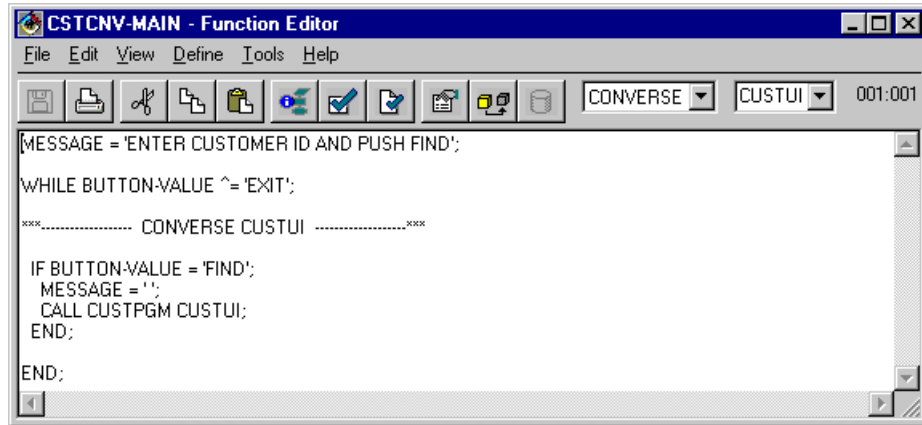


Figure 48. CSTCNV-MAIN processing logic

4. Save the **CSTCNV-MAIN** function. The Customer Info Web Transaction program is now complete. Save the **CSTCNV** program and review the structure shown in the program editor.

Invoking and testing the Customer Info program

1. Click on the Test icon in the **CSTCNV** program editor. This will start the test facility for the **CSTCNV** program.

Note: You can use the restart function of the test facility if you click on the Test icon when the **CSTCNV** program is selected in the VAGen Parts window.

2. Use the test facility step action to walk through the specified processing logic (see Figure 48).

When the CONVERSE option is encountered, the test facility will invoke the Web browser and dynamically build the HTML required for the **CUSTUI** UI Record definition (at runtime this is done by a generated JSP).

In the browser we see the *Customer Info* Web page, with labels (UI type of output) and input fields (UI type of input or input/output).

3. View the HTML source for the browser display. If options such as View->Page source do not work, try saving the current page source to a file and edit the file.

You should be able to see a form with text and input field components implemented from the **CUSTUI** UI Record definition.

4. Enter a valid CustID, for example, the value **101**, and click *Find*.

You should see how the program loops back on the CONVERSE with the information found in the database.

- Enter additional CustID values or click on the Exit button to end the Web Transaction program.
 - Use the test facility to check the UI Record data item input values when interacting with the Web Transaction.
5. Complete the library management tasks required after definition of the Web Transaction program:
- Create an open edition of the *vgv4.web.codebase* package.
 - Create an open edition of the *VGV4 Redbook WebTran Programs* project.
 - Version the *VGV4 WebSphere RAD System* project using a version name such as **Labs Done 8.1.1**.

8.1.3 Single segment (XFER PGM) programming

A UI Record can be sent to a browser using a CONVERSE or an XFER statement. In this section, single segment programs, which use an XFER approach, are studied.

8.1.3.1 Using one UI Record

There are multiple Web Transaction structure options available. As a demonstration of one of the alternative structures available, we will now convert the CONVERSE model **CSTCNV** program (as shown in Figure 45 on page 129), into a program implemented using a single segment (XFER PGM) structure (see Figure 49).

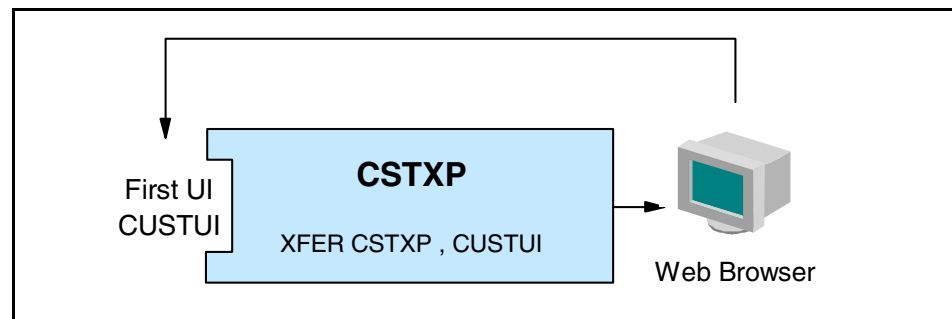


Figure 49. CSTXP Web Transaction program structure

The XFER PGM logic shown in Figure 49 does not include the use of a working storage record (the full syntax is `XFER PGM WSRec, UIRec`). When a working storage record is not passed, only the data sent out in the UI Record

can be returned to the target Web Transaction program; which data is returned depends on the data item properties in the UI Record.

Edit Customer Info UI Record

Edit **CUSTUI** UI Record to:

- Require input for CustID before run XFER.
- Allow Exit button to work without checking for CustUI input

Follow these steps:

1. Modify the UI properties to define the input edits required for **CUSTID** data item:

- Select the *Edit* tab page.
- Define 3 as a minimum input value.

This will ensure that at least 3 digits are entered in the CustID text field.

2. Rename **BUTTON** data item to **BUTTON1**; change **Occurs** data item value to 1 and then:

- Click the *Custom* cell for the *UI Properties* of the **BUTTON1** item. This will bring up the UI Properties window.
- Remove the **Exit** value from the *UI Label* multi-line edit. Next click the *Submit* tab and remove the **EXIT** value.

3. Use mouse button 2 on the **BUTTON1** data item and select *Copy*, then select *Paste*. Name the resulting data item as **BUTTON2**.

- Select *Submit Bypass* as UI Type.
- Click the *Custom* cell for the *UI Properties* of the **BUTTON2** item. This will bring up the UI Properties window.
- Enter **Exit** as a value in the *UI Label* multi-line edit. Next click the *Submit* tab and enter **EXIT** as a value.

The resulting UI Record definition is shown in Figure 50.

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared
BUTTON2	1	Char	8	0	Submit Bypass	Custom	Nonshared
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared
FNAME	1	Char	30	0	Input/Output	Custom	Nonshared
LNAME	1	Char	30	0	Input/Output	Custom	Nonshared
BANKID	1	Char	4	0	Input/Output	Custom	Nonshared
MESSAGE	1	Char	70	0	Output	Default	Nonshared

Figure 50. CUSTUI UI Record definition

4. Save the **CUSTUI** UI Record.

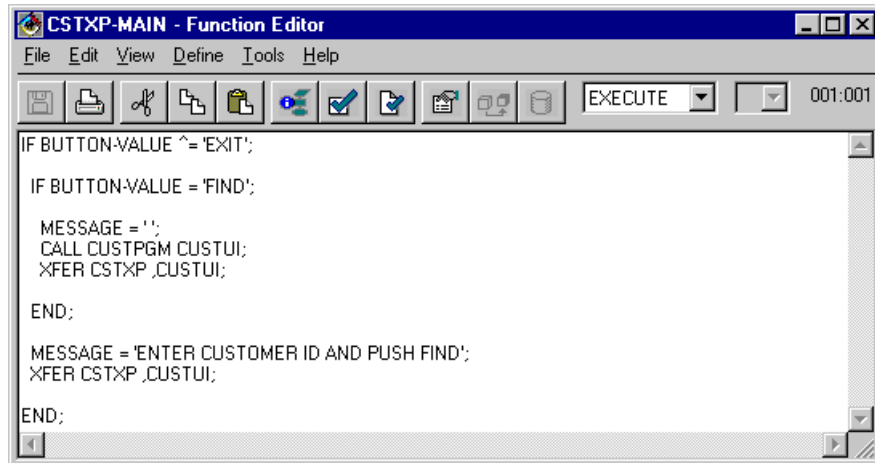
Basic single segment program structure

Using the same **CUSTUI** UI Record used in the **CSTCNV** program, create a new Web Transaction program (**CSTXP**) to implement the same functionality using the XFER Program, UIRec model.

1. Create a new Web Transaction program (**CSTXP**). Open the VAGen Parts Browser and add a new part.
 - In the New VAGen Part window, enter **CSTXP** for Part Name, select **Program** for Part Type, and select *vgv4.web.codebase* for the Package. Click on **OK**. This will open the program editor.
2. In the Program Editor window, select Web Transaction as Program Type.
3. Define **CUSTUI** as the first UI Record for the new program.

Use mouse button 2 on Specifications and select the *Add First UI Record...* menu option. Select **CUSTUI** as the first UI record.
4. Define the main function for the program:
 - Use mouse button 2 on the Structure Diagram and select the *Insert Main Functions...* menu option. Enter **CSTXP-MAIN** as the part name.
 - Open the **CSTXP-MAIN** part (select *vgv4.web.codebase* as the target package for the new part).
 - Select **EXECUTE** as the I/O option.
5. Reuse portions of the processing logic shown in Figure 49 on page 134 to implement the same processing function as provided by the CONVERSE program structure.

Note that the first time the function is invoked, no button has been pushed, so the program must loop to itself. The function logic required is shown in Figure 51.



The screenshot shows a window titled "CSTXP-MAIN - Function Editor". The menu bar includes "File", "Edit", "View", "Define", "Tools", and "Help". Below the menu bar is a toolbar with various icons for file operations and execution. A dropdown menu is set to "EXECUTE" and a counter shows "001:001". The main text area contains the following code:

```
IF BUTTON-VALUE ^= 'EXIT';  
  
IF BUTTON-VALUE = 'FIND';  
  
    MESSAGE = '';  
    CALL CUSTPGM CUSTUI;  
    XFER CSTXP ,CUSTUI;  
  
END;  
  
MESSAGE = 'ENTER CUSTOMER ID AND PUSH FIND';  
XFER CSTXP ,CUSTUI;  
  
END;
```

Figure 51. CSTXP-MAIN processing logic

6. Save the **CSTXP-MAIN** function and the **CSTXP** program.
7. Use the test facility to test the **CSTXP** program.
8. Test the input edits and check the UI Record input fields:
 - Enter a CustID value with only 2 digits and click on the **Find** button. You should receive an error message in **red** that is positioned under the field in error. This is the default error management behavior associated with predefined edits. Additional customization is supported.
 - Enter invalid input and attempt to use the **Find** button. Unlike with the **Find** button, the **Exit** button bypass definition allows the button to trigger the target program even when invalid input has been entered. A bypass edit button will not pass data values for a field in error back to the target program, but other field values, where the input is not in error, are passed back to the program.
9. Version your code, once it is working.

8.1.3.2 Using two UI Records

Convert the current single segment Web Transaction program into a set of programs that use two UI Records, one for end user input and one for end user output (see Figure 52).

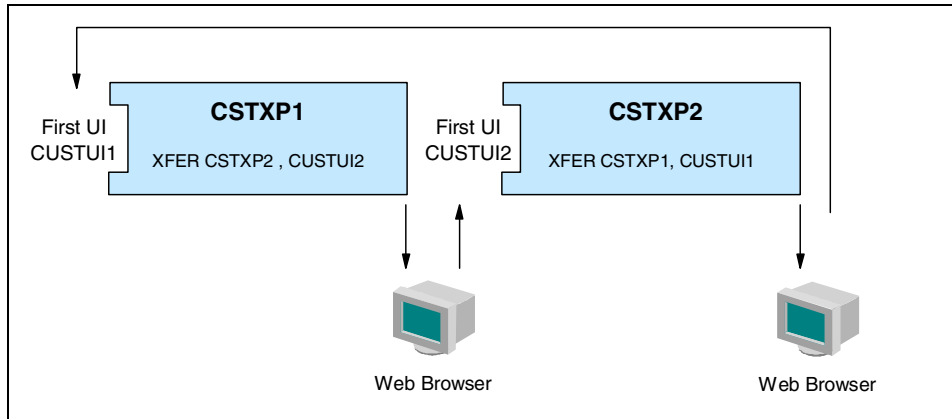


Figure 52. CSTXP1 and CSTXP2 Web Transaction programs structure

The UI Record definition for end user input, **CUSTUI1**, must implement the user interface shown in Figure 53.

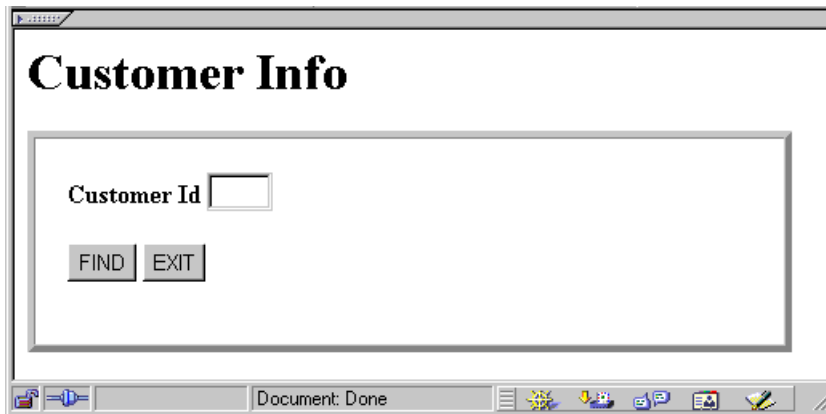


Figure 53. Input Customer Info Web page

The UI Record definition for end user output, **CUSTUI2**, must implement the user interface shown in Figure 54.



Figure 54. Output Customer Info Web page

Create input and output Customer Info UI Records

1. Create the UI Records **CUSTUI1** and **CUSTUI2** as copies of the **CUSTUI** UI Record. Open the VAGen Parts Browser and use mouse button 2 on the **CUSTUI** UI Record and select the *Copy...* menu option.
2. Edit the **CUSTUI1** UI Record to remove the data items no longer needed and use *Input* for CustID Data Type as shown in Figure 55.

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared
BUTTON2	1	Char	8	0	Submit Bypass	Custom	Nonshared
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared
CUSTID	1	Char	4	0	Input	Custom	Nonshared
MESSAGE	1	Char	70	0	Output	Default	Nonshared

Figure 55. CUSTUI1 UI Record definition

3. Edit the **CUSTUI2** UI Record to set the UI Type to *Output* for all database-only data items, as shown in Figure 56.

Name	Occurs	Type	Length	Decimals	UI Type	UI Properties	Usage
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared
BUTTON2	1	Char	8	0	Submit Bypass	Custom	Nonshared
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared
CUSTID	1	Char	4	0	Output	Custom	Nonshared
FNAME	1	Char	30	0	Output	Custom	Nonshared
LNAME	1	Char	30	0	Output	Custom	Nonshared
BANKID	1	Char	4	0	Output	Custom	Nonshared
MESSAGE	1	Char	70	0	Output	Default	Nonshared

Figure 56. CUSTUI2 UI Record definition

4. Reset the customized edit UI Properties for the **CUSTID** data item by changing the minimum input value to 0 (or use the defaults button to reset the page).
5. Click the *Custom* cell for the *UI Properties* of the **BUTTON1** item. This will bring up the UI Properties window. Enter **Next** as the *UI Label* multi-line edit value. Next click the *Submit* tab and enter **NEXT** as the edit value.

Single segment programs structure with multiple UI Records

1. Create a new Web Transaction program (**CSTXP1**). Open the VAGen Parts Browser and add a new part.
 - In the New VAGen Part window, enter **CSTXP1** for Part Name, select **Program** for Part Type, and select *vgv4.web.codebase* for the Package. Click on **OK**. This will open the program editor.
2. In the Program Editor window, select Web Transaction as Program Type. Define **CUSTUI1** as the first UI Record for the new program; use mouse button 2 on Specifications and select the *Add First UI Record...* menu option. Select **CUSTUI1** as the first UI record.
3. Define program main function.
 - Use mouse button 2 on the Structure Diagram and select the *Insert Main Functions...* menu option. Enter **CSTXP1-MAIN** as the part name.
 - Open the **CSTXP1-MAIN** part (select *vgv4.web.codebase* as the target package for the new part).
 - Select **EXECUTE** as the I/O option.
4. The required function logic is shown in Figure 57.

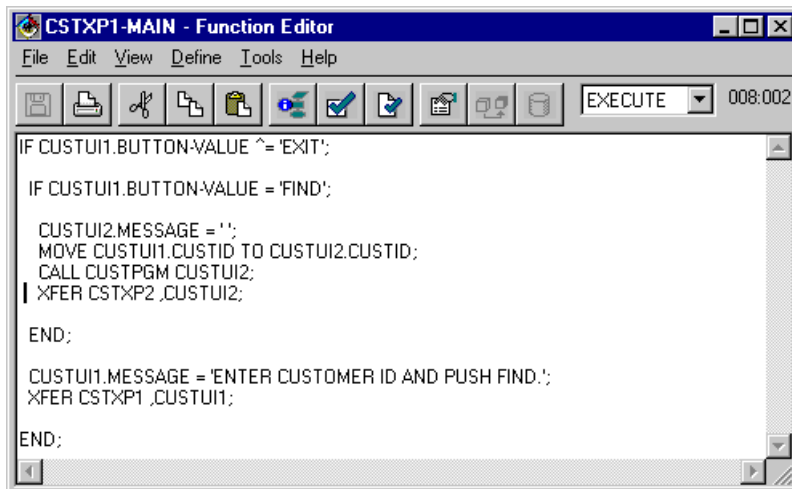


Figure 57. CSTXP1-MAIN processing logic

Note that short names for the data items do not work any more because there are two UI Records associated to the Web Transaction. This means that logic must fully qualify the data item name (*CUSTUI1.BUTTON-VALUE*).

5. Save the **CSTXP1-MAIN** function and the **CSTXP1** program.
6. Do the same to define the **CSTXP2** Web Transaction program (use CUSTUI2 as the first UI Record).

The required logic for the **CSTXP2-MAIN** function is shown in Figure 58.

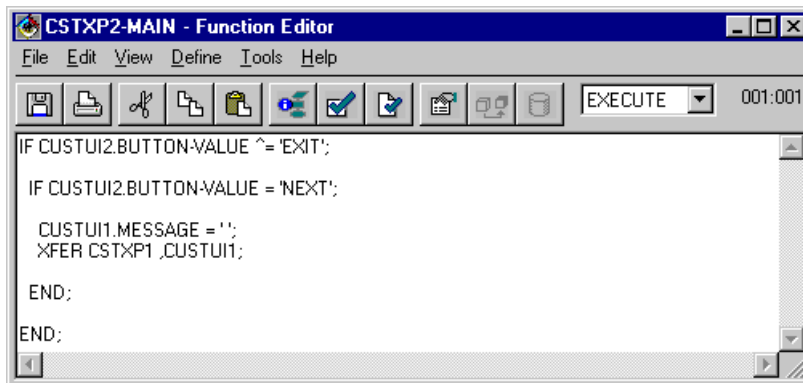


Figure 58. CSTXP2-MAIN processing logic

7. Save the **CSTXP2-MAIN** function and the **CSTXP2** program.

8. Use the test facility to test the **CSTXP1** program (which transfers to the **CSTXP2** program).

Use the test facility to check the UI Record data item input values when interacting with the Web Transactions.

9. Version your code, once it is working.

When using the `XFER PGM` statement and two different UI Records (input and output views), two different Web Transaction programs must be used. Only the first UI Record defined for the target program can be referenced on the XFER statement, so one program is required for each UI Record.

8.1.4 Single segment (XFER ' ') programming

In this section we implement Web Transaction programs using UI Records with defined forms. This programming structure can be used with one or more than one UI Record (different input and output views) per Web Transaction.

8.1.4.1 Using one UI Record

We will now implement the same processing provided in the CONVERSE model **CSTCNV** program in a Web Transaction program implemented using a single segment (XFER ' ') structure (see Figure 59).

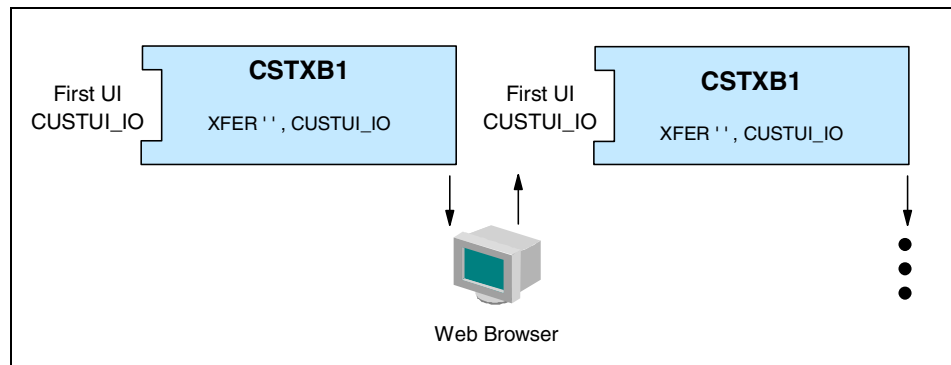


Figure 59. *CSTXB1* Web Transaction program structure

The UI Record definition for input and output, **CUSTUI_IO**, will implement the user interface shown in Figure 60.

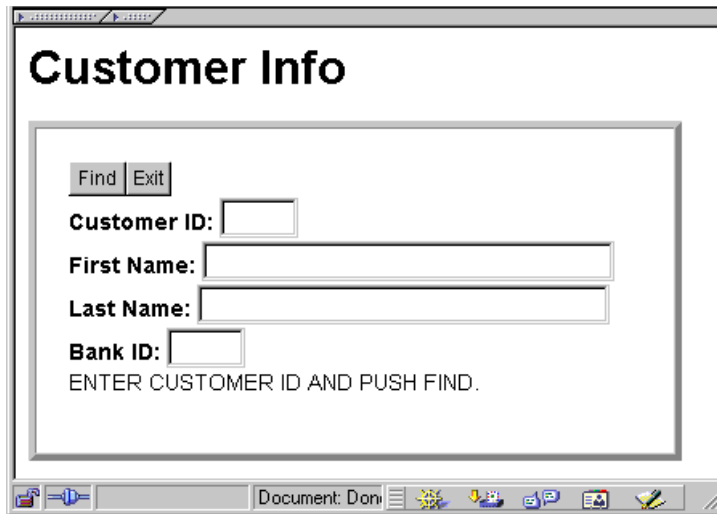


Figure 60. Customer Info Web page

Create input and output Customer Info UI Record

1. Create the UI Record **CUSTUI_IO** as a copy of the **CUSTUI2** UI Record. Open the VAGen Parts Browser and use mouse button 2 on the **CUSTUI2** UI Record and select the *Copy...* menu option.
2. Edit the **CUSTUI_IO** UI Record to change the following data items to Input/Output type: CUSTID, FNAME, LNAME and BANKID.
Change the general and submit values for the **BUTTON1** data item from *Next/NEXT* to *Find/FIND*.
Add minimum input (3 characters) edit for the **CUSTID** data item.
3. Insert a new data item named **FORM_IO** in the **CUSTUI_IO** UI Record. Set the **FORM_IO** data item UI Type to *Form*. In the UI Properties window select the Program Link tab and then enter **CSTXB1** in the Program field (will not be in the drop down list) and **CUSTUI_IO** as the First UI Record.
4. Use mouse button 2 on the **Form** data item and select the *Insert...* menu option to insert a substructure. Drag and drop all the other data items so that they are substructures of the **Form** data item.
5. Add a substructure above the **BUTTON1** and **BUTTON2** data items.
Save the **CUSTUI_IO** UI Record, accept the VisualAge Generator calculation for the superstructure data item length.

The final structure of **CUSTUI_IO** UI Record is shown in Figure 61.

Name	Occ	Type	Len	Dec	UI Type	UI Properties	Usage	Des
FORM ID	1	Char	162	0	Form	Custom	Nonshared	
BUTTONS	1	Char	16	0	Output	Default	Nonshared	
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared	
BUTTON2	1	Char	8	0	Submit Bypass	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared	
FNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
LNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
BANKID	1	Char	4	0	Input/Output	Custom	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 61. CUSTUI_IO UI Record definition

Single segment program structure

1. Create a new Web Transaction program (**CSTXB1**):
 - Open the VAGen Parts Browser and add a new part.
 - In the New VAGen Part window, enter **CSTXB1** for Part Name, select **Program** for Part Type, and select *vgv4.web.codebase* for the Package. Click on **OK**. This will open the program editor.
2. In the Program Editor window:
 - Select Web Transaction as the Program Type.
 - Define **CUSTUI_IO** as the first UI Record for the new program.
 - Using mouse button 2 on Specifications and select the *Add First UI Record...* menu option. Select **CUSTUI_IO** as the first UI record.
3. Define program main function:
 - Use mouse button 2 on the Structure Diagram and select the *Insert Main Functions...* menu option. Enter **CSTXB1-MAIN** as the part name.
 - Open the **CSTXB1-MAIN** part (select *vgv4.web.codebase* as the target package for the new part).
 - Select **EXECUTE** as the I/O option.
4. The required function logic is shown in Figure 62.

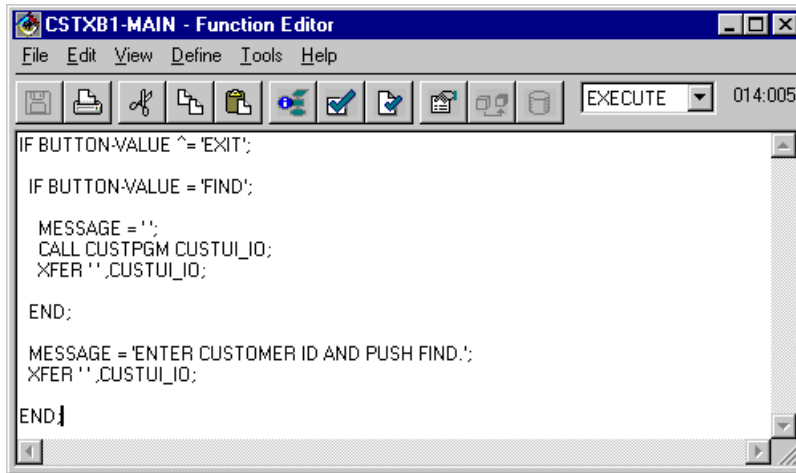


Figure 62. CSTXB1-MAIN processing logic

5. Save the **CSTXB1-MAIN** function and the **CSTXB1** program.
6. Use the test facility to test the **CSTXB1** program and check data values.
7. Version your code, once it is working.

8.1.4.2 Using two UI Records

Convert this Web Transaction system into one Web Application that uses two different UI Records, one to collect user input (**CUSTUI_I**), and one to show the use of the output (**CUSTUI_O**), and a new record which will be defined as the first UI Record for the one Web Transaction program (see Figure 63).

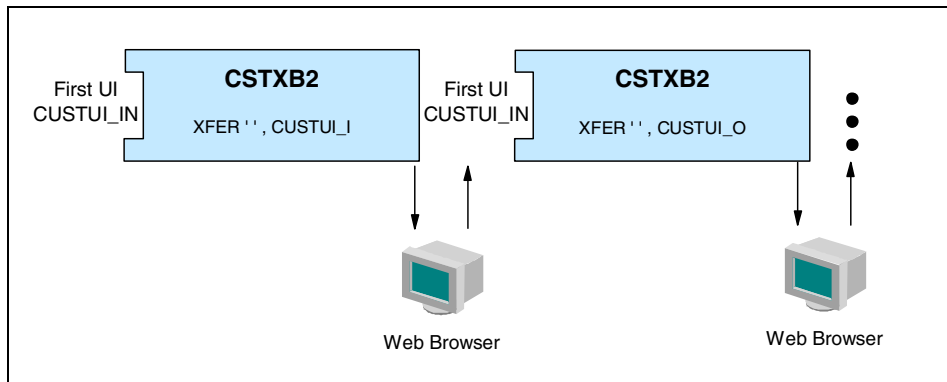


Figure 63. CSTXB2 Web Transaction program structure

The UI Record definition for input, **CUSTUI_I**, must implement the user interface shown in Figure 64.

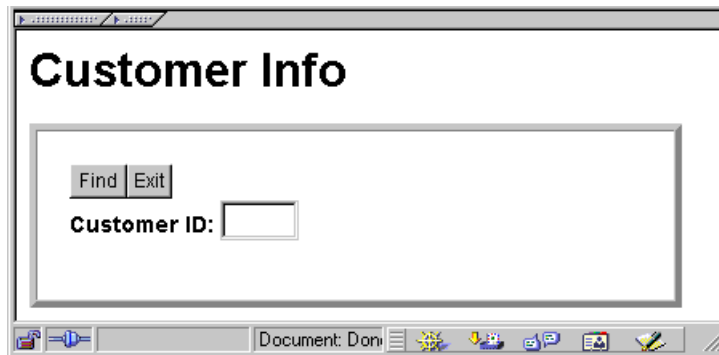


Figure 64. Input Customer Info Web page

The UI Record definition for output, **CUSTUI_O**, must implement the user interface shown in Figure 65.

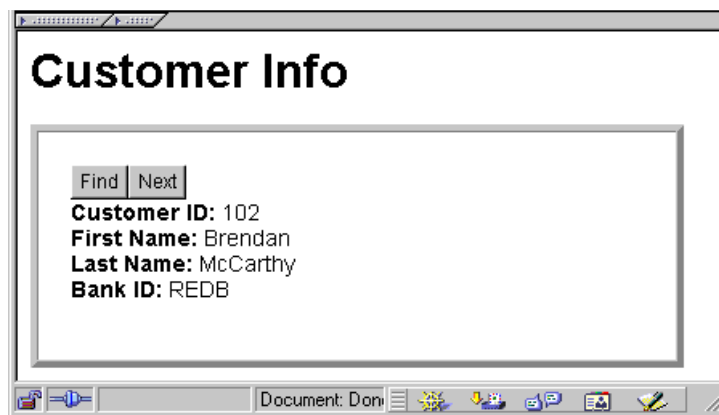


Figure 65. Output Customer Info Web page

Create Customer Info UI Records

1. Create the UI Records **CUSTUI_IN**, **CUSTUI_I** and **CUSTUI_O** as copies of the **CUSTUI_IO** UI Record.

Open the VAGen Parts Browser and use mouse button 2 on the **CUSTUI_IO** UI Record and select the *Copy...* menu option.

2. Edit the **CUSTUI_I** UI Record to match the structure shown in Figure 66.

Name	Occ	Type	Len	Dec	UI Type	UI Properties	Usage	Descrip
FORM I	1	Char	162	0	Form	Custom	Nonshared	
BUTTONS	1	Char	16	0	Output	Default	Nonshared	
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared	
BUTTON2	1	Char	8	0	Submit Bypass	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 66. CUSTUI_I UI Record definition

3. Make these customizations to the **CUSTUI_I** record:

- Define general and submit values of *Find/FIND* for **BUTTON1** and *Exit/EXIT* for **BUTTON1**.
- Customize **BUTTON1** data item to implement the **Find** button and **BUTTON3** data item will implement the **Exit** button.
- On the Program Link tab in the **FORM_I** UI Properties window define **CSTXB2** as the Program and **CUSTUI_IN** as the First UI Record.

4. Edit the **CUSTUI_O** UI Record to meet the structure shown in Figure 67.

Name	Occ	Type	Len	Dec	UI Type	UI Properties	Usage	Des
FORM_O	1	Char	162	0	Form	Custom	Nonshared	
BUTTONS	1	Char	16	0	Output	Default	Nonshared	
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared	
BUTTON2	1	Char	8	0	Submit Bypass	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Output	Custom	Nonshared	
FNAME	1	Char	30	0	Output	Custom	Nonshared	
LNAME	1	Char	30	0	Output	Custom	Nonshared	
BANKID	1	Char	4	0	Output	Custom	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 67. CUSTUI_O UI Record definition

5. Make these customizations to the **CUSTUI_O** record:
 - Define general and submit values of *Next/NEXT* for **BUTTON2** and *Exit/EXIT* for **BUTTON3**.
 - In the UI Properties window for **FORM_O** pick the Program Link tab, then use **CSTXB2** for the Program field and **CUSTUI_IN** for First UI Record.
6. Edit the **CUSTUI_IN** UI Record to meet the structure shown in Figure 68.

The screenshot shows a window titled "CUSTUI_IN - Record Editor" with a menu bar (File, Edit, View, Define, Tools, Help) and a toolbar. Below the toolbar is a table with the following data:

Name	Occ	Type	Len	Dec	UI Type	UI Properties	Usage	Descrip
FORM_IN	1	Char	106	0	Form	Custom	Nonshared	
BUTTONS	1	Char	24	0	Output	Default	Nonshared	
BUTTON1	1	Char	8	0	Submit	Custom	Nonshared	
BUTTON2	1	Char	8	0	Submit	Custom	Nonshared	
BUTTON3	1	Char	8	0	Submit Bypass	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 68. CUSTUI_IN UI Record definition

7. Make these customizations to the **CUSTUI_IN** record:
 - Define general and submit values of *Find/FIND* for **BUTTON1**, *Next/NEXT* for **BUTTON2**, and *Exit/EXIT* for **BUTTON3**.
 - In the UI Properties window for **FORM_IN** pick the Program Link tab, then use **CSTXB2** for the Program field and **CUSTUI_IN** for First UI Record.

We use three different names for Submit data items because Submit items defined on a Form that identifies the target Web Transaction must have matching Submit items on the receiving UI Record (to calculate value to be stored in the submit value item value).

Single segment program structure with multiple UI Records

1. Create a new Web Transaction program (**CSTXB2**). Open the VAGen Parts Browser and add a new part.
 - In the New VAGen Part window, enter **CSTXB2** for Part Name, select **Program** for Part Type, and select *vgv4.web.codebase* for the Package. Click on **OK**. This will open the program editor.

2. In the Program Editor window, select Web Transaction as Program Type. Define **CUSTUI_IN** as the first UI Record for the new program; use mouse button 2 on Specifications and select the *Add First UI Record...* menu option. Select **CUSTUI_IN** as the first UI record.
3. Define program main function.
 - Use mouse button 2 on the Structure Diagram and select the *Insert Main Functions...* menu option. Enter **CSTXB2-MAIN** as the part name.
 - Open the **CSTXB2-MAIN** part (select *vgv4.web.codebase* as the target package for the new part).
 - Select **EXECUTE** as the I/O option.
4. The required function logic is shown in Figure 69.

The screenshot shows a window titled "CSTXB2-MAIN - Function Editor" with a menu bar (File, Edit, View, Define, Tools, Help) and a toolbar. The main area contains the following code:

```

IF CUSTUI_IN.BUTTON-VALUE ^= 'EXIT';

IF CUSTUI_IN.BUTTON-VALUE = 'FIND';

  CUSTUI_O.MESSAGE = '';
  MOVE CUST_IN.CUSTID TO CUSTUI_O.CUSTID;
  CALL CUSTPGM CUSTUI_O;
  XFER '',CUSTUI_O;

END;

IF CUSTUI_IN.BUTTON-VALUE = 'NEXT';
  CUSTUI_O.MESSAGE = '';
  XFER '',CUSTUI_I;
END;

CUSTUI_O.MESSAGE = 'ENTER CUSTOMER ID AND PUSH FIND.';
XFER '',CUSTUI_I;

END;

```

Figure 69. CSTXB2-MAIN processing logic

5. Save the **CSTXB2-MAIN** function and the **CSTXB2** program.
6. Use the test facility to test the **CSTXB2** program and check data values.
7. Try entering less than 3 characters in the CUSTID field. Notice that the error is returned in the **CUSTUI_IN** record, as all edits are processed in the first UI Record defined for a single segment program.
8. Version your code, once it is working.

8.2 Implementing global state management

It is common in application systems to have logic that implements different functions based on the user (or user type) currently using the application. In our Bank database example, we can add control logic that allows update access for some users, so they can use the application to change the actual value of some fields in the database.

To keep the implementation simple, a pre-defined application will ask for the user name and then pass data back to the target Web Transaction for use in determining application level authority. This passed information is the *state data* that must be kept active by all subsequent Web Transaction programs.

In traditional systems, this state data would be shared by all active programs, using techniques such as a passed common working storage record. There are multiple options available for storing state data in a Web Transaction system:

- UI Record data kept in either active beans (session data) or hidden fields in the HTML sent to the browser (and defined to return to the target program)
- Working storage record(s) whose state is saved by VisualAge Generator runtime processing during user interaction
- Self-managed state processing

The implementation of this global data access in Web Transactions is studied in this section.

8.2.1 UI Record-based state management implementation

State data can be kept active in the UI Record used by the Web Transaction. The approach depends on the Web Transaction program structure.

8.2.1.1 CONVERSE model

In this exercise we will convert the **CSTCNV** Web Transaction into one that accepts state information from the **CSTIDUS** Web Transaction (provided) that identifies the user type and implements the state management by storing data in the UI Record. The structure of this system is shown in Figure 70.

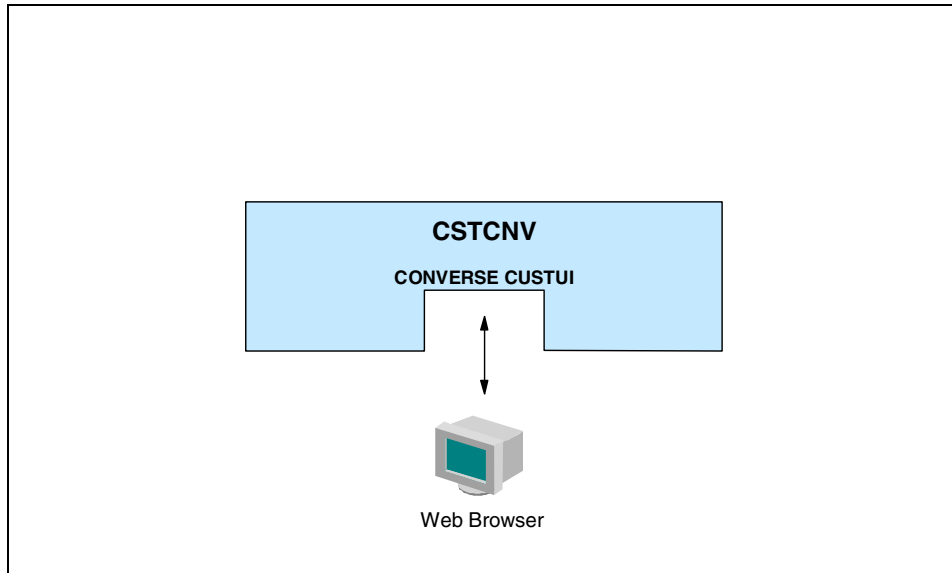


Figure 70. CSTCVS Web Transaction program structure

The UI Record provided for input, **CUSTUI_ID**, implements the user interface where the user type and next Web Transaction are selected (see Figure 71).

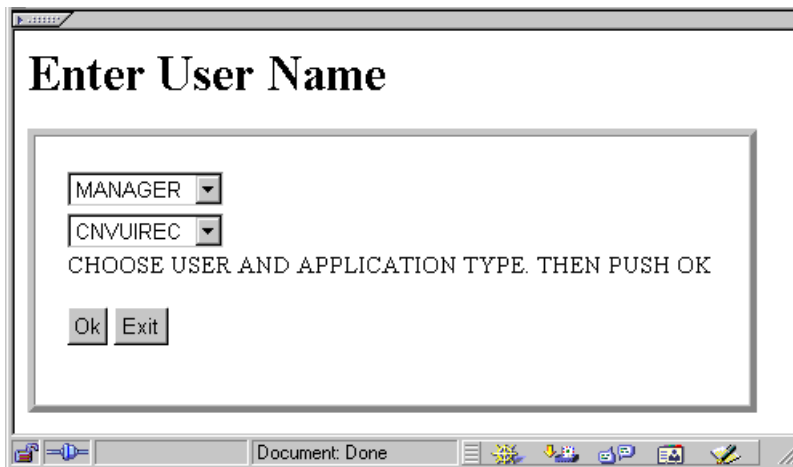


Figure 71. User Name inquiry Web page

The UI Record definition for output, **CUSTUIS**, must implement the user interface shown in Figure 72, without update support, or the user interface shown in Figure 73, with update support.

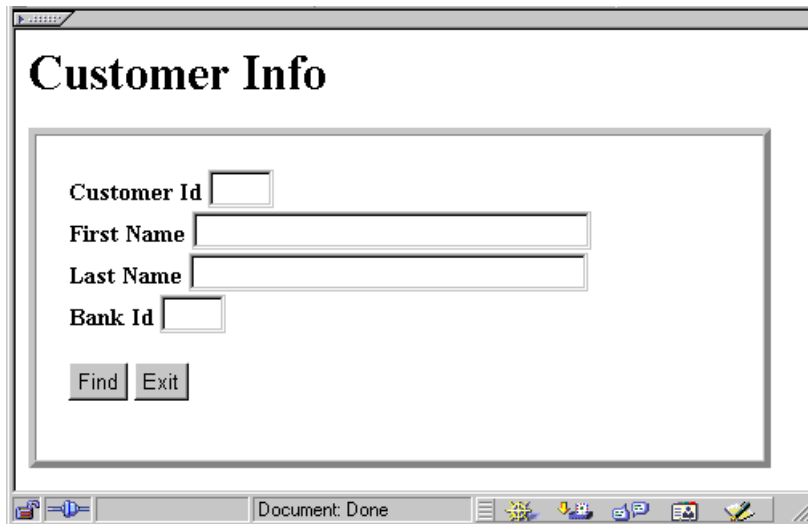


Figure 72. Customer Info Web page without the Update button

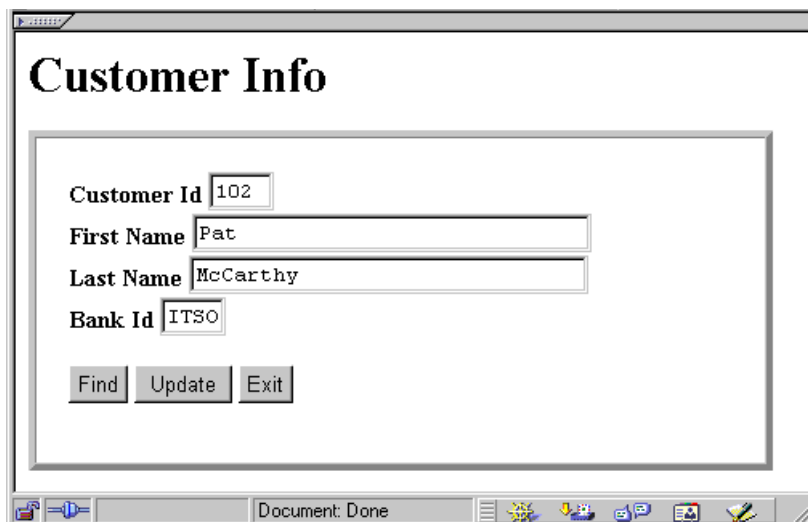


Figure 73. Customer Info Web page with the Update button

Create Input/Output UI Record

1. Create the UI Record **CUSTUIS** as a copy of the **CUSTUI** UI Record. Open the VAGen Parts Browser and use mouse button 2 on the **CUSTUI** UI Record and select the *Copy...* menu option.
2. Edit the **CUSTUIS** UI Record to match the structure shown in Figure 74.

Name	Occ	Type	Len	Dec	UI Type	UI Properties	Usage	Description
BUTTONS	2	Char	8	0	Submit	Custom	Nonshared	
BUTTON-EXIT	1	Char	8	0	Submit Bypass	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared	
FNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
LNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
BANKID	1	Char	4	0	Input/Output	Custom	Nonshared	
CANUPDATE	1	Char	1	0	None	None	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 74. CUSTUIS UI Record definition

The following submit and submit bypass definitions need to be refined:

- Define general and submit values of *Find/FIND* and *Update/UPDATE* for the **BUTTONS** array
- Define general and submit values of *Exit/EXIT* for **BUTTON-EXIT**.

Note: State data is going to be stored in the **CANUPDATE** data item, which is defined with a UI Type of None, so that the information about the state is not sent to the browser (the UI Type Hidden is sent, just not displayed). The data will not be seen in the browser or the raw HTML used to compose the browser view.

3. Save the **CUSTUIS** record and close the record editor.

CSTIDUS program and CSTIDUS-MAIN logic

The **CSTIDUS** program is provided in the code base originally loaded to start these exercises.

The **CSTIDUS-MAIN** function logic implements a loop around a CONVERSE statement to solicit user type input and transfer control to the target Web Transaction program. The **CSTIDUS-GET-PROFILE** function is called using the **CUSTUI_ID.USER** data item as a parameter. This function will store the user type (update access) profile in the **STATEWS.UPDATE** data item. This data is passed, as required, to the target Web Transaction program.

The **CSTIDUS-MAIN** function logic relevant for this exercise is shown in Figure 75.

```

IF CUSTUI_ID.BUTTON-VALUE = 'OK' AND
CUSTUI_ID.TARGET-APP = 'CNVUIREC';
STATEWS.UPDATE = CSTIDUS-GET-PROF( CUSTUI_ID.USER); /* Get user profile
DXFR CSTCNS STATEWS; /* Transfer to target pgm - with profile data
END;

```

Figure 75. CSTIDUS-MAIN processing logic (1)

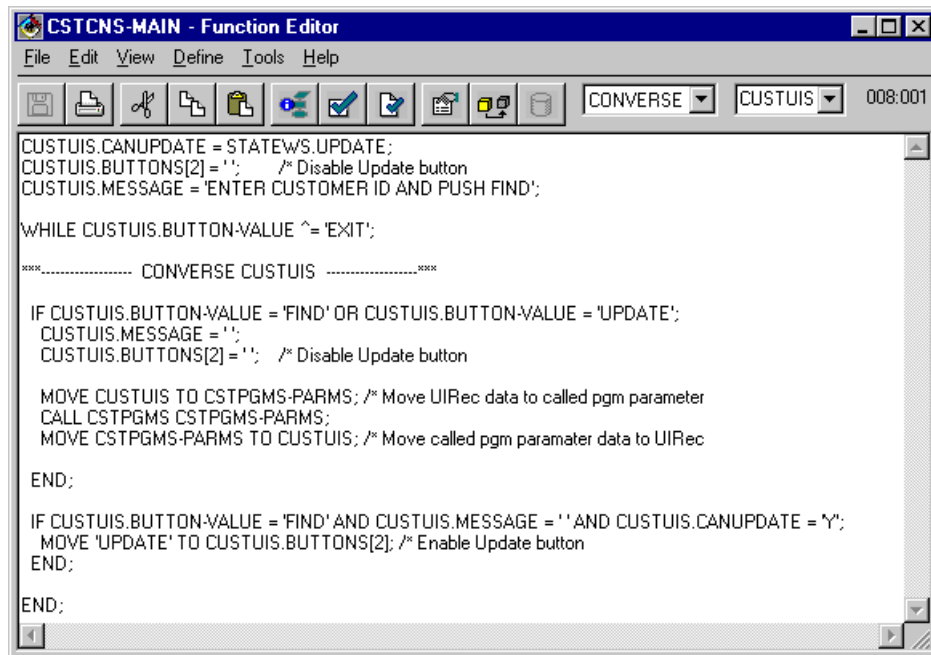
Create Customer Info Web Transaction

1. Define the **CSTCNS** Web Transaction program which will accept the passed state data:
 - Open the VAGen Parts Browser and add a new part.
 - In the New VAGen Part window, enter **CSTCNS** for Part Name, select **Program** for Part Type, and select *vgv4.web.codebase* for the Package. Click on **OK**. This will open the program editor.
 - Select Web Transaction as program type.
2. Add **STATEWS** working storage to *Specifications* list.
 - Use mouse button 2 on the *Specifications* list and select the *Add Working Storage...* menu option. Select **STATEWS** as the part name.
3. Add **CSTPGMS-PARMS** to *Tables and Additional Records* list.
 - Use mouse button 2 on the *Tables and Additional Records* list and select the *Insert Table/Record...* menu option. Select **Record** as *Type* and **CSTPGMS-PARMS** as the part name.
4. Define program main function:
 - Use mouse button 2 on the Structure Diagram and select the *Insert Main Functions...* menu option. Enter **CSTCNS-MAIN** as the part name.
 - Open the **CSTCNS-MAIN** part (select *vgv4.web.codebase* as the target package for the new part).
 - Converse the **CUSTUIS** UI Record (select **CONVERSE** as the I/O option and **CUSTUIS** as the I/O object).
5. Define the function logic required to call the server program as part of a loop around the Converse of the UI Record.

The loop should continue until the Exit submit request is received. The pre-defined server **CUSTPGMS** is called using the **CUSTPGMS-PARMS** working storage record as a parameter. The server program either returns data, updates data, or returns an error message in the **MESSAGE** field

if a successful *Find* was performed and the user has update capability (provided through the **CUSTUIS** UI Record). The **Update** button is now displayed.

The required function logic is shown in Figure 76.



```
CSTCNS-MAIN - Function Editor
File Edit View Define Tools Help
CONVERSE CUSTUIS 008:001

CUSTUIS.CANUPDATE = STATEWS.UPDATE;
CUSTUIS.BUTTONS[2] = ''; /* Disable Update button
CUSTUIS.MESSAGE = 'ENTER CUSTOMER ID AND PUSH FIND';

WHILE CUSTUIS.BUTTON-VALUE ^= 'EXIT';

***----- CONVERSE CUSTUIS -----***

IF CUSTUIS.BUTTON-VALUE = 'FIND' OR CUSTUIS.BUTTON-VALUE = 'UPDATE';
  CUSTUIS.MESSAGE = '';
  CUSTUIS.BUTTONS[2] = ''; /* Disable Update button

  MOVE CUSTUIS TO CSTPGMS-PARMS; /* Move UIRec data to called pgm parameter
  CALL CSTPGMS CSTPGMS-PARMS;
  MOVE CSTPGMS-PARMS TO CUSTUIS; /* Move called pgm parameter data to UIRec

END;

IF CUSTUIS.BUTTON-VALUE = 'FIND' AND CUSTUIS.MESSAGE = '' AND CUSTUIS.CANUPDATE = 'Y';
  MOVE 'UPDATE' TO CUSTUIS.BUTTONS[2]; /* Enable Update button
END;

END;
```

Figure 76. CSTCNS-MAIN processing logic (1)

6. Save the **CSTCNS-MAIN** function and **CSTCNS** program.
7. Use the test facility to test the **CSTIDUS** program. Choosing the **CONVUIREC** option will trigger invocation of the **CSTCNS** program.

Use the test facility to check the UI Record data item input values when interacting with the Web Transaction. Can you find the **CANUPDATE** data item in the browser or HTML? Try both **USER** and **MANAGER** values.

8. Version your code, once it is working.

8.2.1.2 XFER PGM model

In this exercise we will convert the **CSTXP** XFER PGM Web Transaction into one which accepts state information from the **CSTIDUS** Web Transaction that identifies the user type and implements the state management by storing data in the UI Record. The structure of this system is shown in Figure 77.

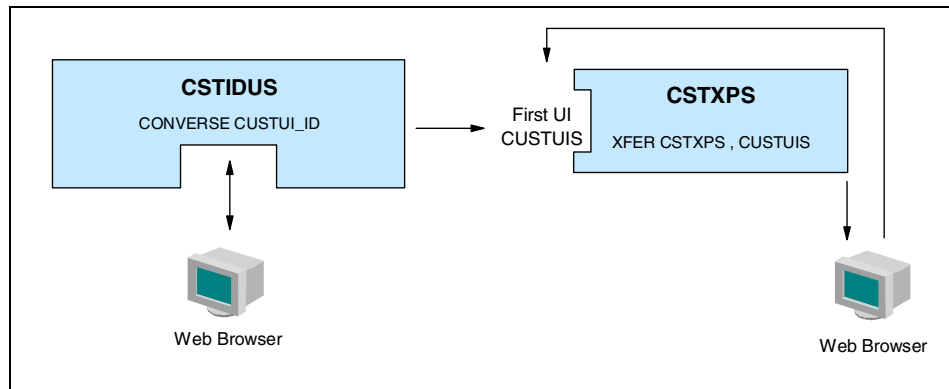


Figure 77. CSTXPS Web Transaction program structure

CSTIDUS-MAIN function logic

To support the **CSTXPS** program, this function will pass the user type (update access) profile directly in the **CANUPDATE** data item in the **CUSTUIS** UI Record. This data is then passed along to the **CSTXPS** Web Transaction program.

The **CSTIDUS-MAIN** function logic relevant for this exercise is shown in Figure 78.

```

IF CUSTUI_ID.BUTTON-VALUE = 'OK' AND CUSTUI_ID.TARGET-APP = 'XPUIREC';
  CUSTUIS.CANUPDATE = CSTIDUS-GET-PROF( CUSTUI_ID.USER); /* Get profile
  CUSTUIS.BUTTONS[2] = '';
  XFER CSTXPS ,CUSTUIS; /* Transfer to target pgm - with profile data
END;
  
```

Figure 78. CSTIDUS-MAIN processing logic (2)

The state data will be stored directly in the **CUSTUIS** UI Record, so that it is not accessible to the user. The data will not be in the HTML data sent to the browser, but stored as part of the session data bean kept for an XFER Pgm Web Transaction.

Create Customer Info Web Transaction

1. Define the **CSTXPS** Web Transaction program:
 - Open the VAGen Parts Browser and add a new part.
 - In the New VAGen Part window, enter **CSTXPS** for Part Name, select **Program** for Part Type, and select *vgv4.web.codebase* for the Package. Click on **OK**. This will open the program editor.
 - Select Web Transaction as program type.
2. Define **CUSTUIS** as the first UI Record for the new program; use mouse button 2 on Specifications and select the *Add First UI Record...* menu option. Select **CUSTUIS** as the first UI record.
3. Add **CSTPGMS-PARMS** to *Tables and Additional Records* list.

Use mouse button 2 on the *Tables and Additional Records* list and select the *Insert Table/Record...* menu option. Select **Record** as *Type* and **CSTPGMS-PARMS** as the part name.
4. Define program main function:
 - Use mouse button 2 on the Structure Diagram and select the *Insert Main Functions...* menu option. Enter **CSTXPS-MAIN** as the part name.
 - Open the **CSTXPS-MAIN** part (select *vgv4.web.codebase* as the target package for the new part).
5. Define required main function logic:
 - Unless the **Exit** button is pushed the pre-defined server **CUSTPGMS** is called using the **CUSTPGMS-PARMS** UI Record as a parameter.
 - The server program either returns data, updates data, or returns an error message in the **MESSAGE** field.
 - If a successful "Find" was performed and the user has update capability (provided through the **CUSTUIS** UI Record) the **Update** button is displayed.
 - To create a loop, the program invokes itself using an XFER Pgm statement.

The required function logic is shown in Figure 79.

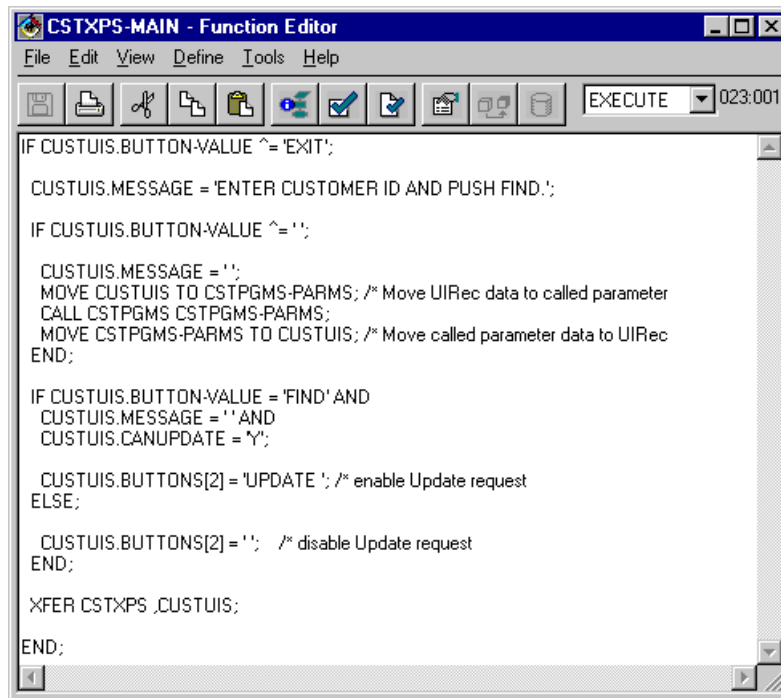


Figure 79. CSTXPS-MAIN processing logic

6. Save the **CSTXPS-MAIN** function and **CSTXPS** program.
7. Use the test facility to test the **CSTIDUS** program. Choosing the **XPUIREC** option will trigger invocation of the **CSTXPS** program.
Use the test facility to check the UI Record data item input values when interacting with the Web Transaction.
8. Version your code, once it is working.

8.2.1.3 XFER'' model

In this exercise we will create an XFER'' Web Transaction that accepts state information from the **CSTIDUS** Web Transaction that identifies the user type and implements the state management by storing data in the UI Record. The structure of this system is shown in Figure 80.

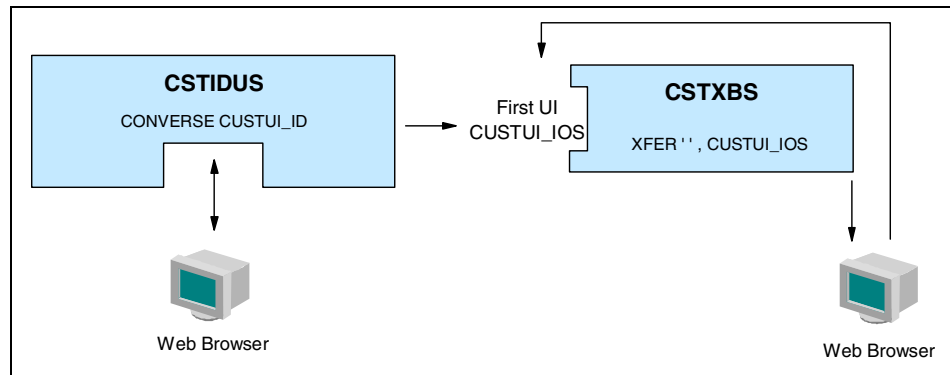


Figure 80. CSTXBS Web Transaction program structure

CSTIDUS-MAIN function logic

To support the **CSTXBS** program, this function will pass the user type (update access) profile directly in the **CANUPDATE** data item in the **CUSTUIS** UI Record. This data is then passed along to the **CSTXPS** Web Transaction program. The **CSTIDUS-MAIN** function logic relevant for this exercise is shown in Figure 81.

```
IF CUSTUI_ID.BUTTON-VALUE = 'OK' AND CUSTUI_ID.TARGET-APP = 'XBUIREC';  
  CUSTUI_IOS.CANUPDATE = CSTIDUS-GET-PROF(CUSTUI_ID.USER); /* Get user profile  
  CUSTUI_IOS.BUTTONS[2] = '';  
  XFER '',CUSTUI_IOS; /* Transfer to target pgm - with profile data  
END;
```

Figure 81. CSTIDUS-MAIN processing logic (3)

Create input/output Customer Info UI Record

1. Create the **CUSTUI_IOS** UI Record as a copy of **CUSTUI_IO**. Open the VAGen Parts Browser and use mouse button 2 on the **CUSTUI_IO** UI Record and select the *Copy...* menu option.
2. Edit the **CUSTUI_IOS** UI Record so that it meets the structure shown in Figure 82.

Name	Occ	Type	Len	Dec	UI Type	UI Properties	Usage	Descripti
FORM_ID	1	Char	171	0	Form	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared	
FNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
LNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
BANKID	1	Char	4	0	Input/Output	Custom	Nonshared	
BUTTON-SET	1	Char	24	0	Output	Default	Nonshared	
BUTTONS	2	Char	8	0	Submit	Custom	Nonshared	
BUTTON-EXIT	1	Char	8	0	Submit Bypass	Custom	Nonshared	
CANUPDATE	1	Char	1	0	None	None	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 82. CUSTUI_IOS UI Record definition

The following submit and submit bypass definitions need to be refined:

- Define general and submit values of *Find/FIND* and *Update/UPDATE* for the **BUTTONS** array.
- Define general and submit values of *Exit/EXIT* for **BUTTON-EXIT**.

Save the **CUSTUI_IOS** UI Record.

Note: By moving the Buttons structure down in the UI Record, we can move the buttons below the input fields in the defined form. Buttons are always at the bottom when a default form is created by VisualAge Generator, but with the defined form used in an XFER ' ' model, you control button placement.

3. Make these customizations to the **CUSTUI_IOS** record:

- On the Program Link tab in the **FORM_IO** UI Properties window define **CSTXBS** as the Program and **CUSTUI_IOS** as the First UI Record.
- In the Link Parameters list insert a line using **CANUPDATE** for Name and Value Item.

This definition will pass the state data forward to the target program. Without this program link parameter, the None UI Type value will not be passed to the target program.

Create Customer Info Web Transaction

1. Define the **CSTXBS** Web Transaction program:
 - Open the VAGen Parts Browser and add a new part.
 - In the New VAGen Part window, enter **CSTXBS** for Part Name, select **Program** for Part Type, and select *vgv4.web.codebase* for the Package. Click on **OK**. This will open the program editor.
 - Select Web Transaction as program type.
2. Define **CUSTUI_IOS** as the first UI Record for the new program; use mouse button 2 on Specifications and select the *Add First UI Record...* menu option. Select **CUSTUI_IOS** as the first UI record.
3. Add **CSTPGMS-PARMS** to *Tables and Additional Records* list.

Use mouse button 2 on the *Tables and Additional Records* list and select the *Insert Table/Record...* menu option. Select **Record** as *Type* and **CSTPGMS-PARMS** as the part name.
4. Define program main function:
 - Use mouse button 2 on the Structure Diagram and select the *Insert Main Functions...* menu option. Enter **CSTXBS-MAIN** as the part name.
 - Open the **CSTXBS-MAIN** part (select *vgv4.web.codebase* as the target package for the new part).
5. Define required main function logic:
 - Unless the **Exit** button is pushed, the pre-defined server **CUSTPGMS** is called using the **CUSTPGMS-PARMS** UI Record as a parameter. The server program either returns data, updates data, or returns an error message in the **MESSAGE** field.
 - If a successful "Find" was performed and the user has update capability (provided through the **CUSTUIS** UI Record), the **Update** button is displayed.
 - To create a loop, the program invokes itself using an XFER ' ' statement.

The required function logic is shown in Figure 83.

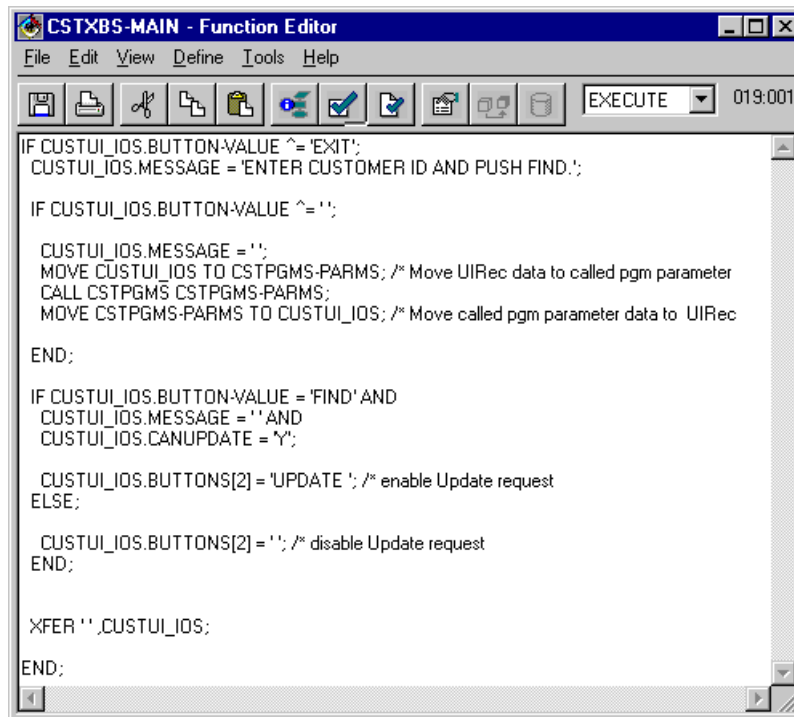


Figure 83. CSTXBS-MAIN processing logic

6. Save the **CSTXBS-MAIN** function and the **CSTXBS** program.
7. Use the test facility to test the **CSTIDUS** program. Choosing the **XBUIREC** option will trigger invocation of the **CSTXBS** program. Set UI Record watch points to monitor data.

Review the HTML sent to the browser. You should be able to find the **CANUPDATE** value, which, while defined as a None UI Type, becomes hidden data in the browser to implement the program link parameter required to pass this state data back to the target program.

If this data is sensitive, or you are concerned that the values might be manipulated, you might need to consider other options for state management when using the XFER '' program structure (see 8.2.3, "Self-managed state implementation (XFER '' model)" on page 168).

8. Version your code, once it is working.

8.2.2 Working storage record-based state management

State data can be passed between Web Transactions using a working storage record, when the transfer command accepts a working storage record parameter (depends on the Web Transaction program structure). By using a working storage record, we avoid sending the data to both the WebSphere Application Server platform and the browser. This may be a requirement either when the data is sensitive, or when the volume of data would delay network transport.

8.2.2.1 CONVERSE model

In this exercise we will convert the **CSTCNS** Web Transaction into one (**CSTCNS2**) that uses a predefined state server program to obtain state data from an application database.

CSTIDUS-MAIN function logic

To support the **CSTCNS2** program, this function will pass the user type (update access) profile directly in the passed working storage record, where it will be kept during subsequent program logic.

Figure 84 shows the **CSTIDUS-MAIN** function logic relevant for this exercise.

```
IF CUSTUI_ID.BUTTON-VALUE = 'OK' AND CUSTUI_ID.TARGET-APP = 'CNVWKST';  
  STATEWS.UPDATE = CSTIDUS-GET-PROF( CUSTUI_ID.USER); /* Get user profile  
  DXFR CSTCNS2 STATEWS;  
END;
```

Figure 84. *CSTIDUS-MAIN* processing logic (4)

Create Customer Info Web Transaction

1. Define the **CSTCNS2** Web Transaction program.:
 - Open the VAGen Parts Browser and add a new part.
 - In the New VAGen Part window, enter **CSTCNS2** for Part Name, select **Program** for Part Type, and select *vgv4.web.codebase* for the Package. Click on **OK**. This will open the program editor.
 - Select Web Transaction as program type.
2. Add **STATEWS** working storage to *Specifications* list.

Use mouse button 2 on the *Specifications* list and select the *Add Working Storage...* menu option. Select **STATEWS** as the part name.

3. Add **CSTPGMS-PARMS** to *Tables and Additional Records* list.

Use mouse button 2 on the *Tables and Additional Records* list and select the *Insert Table/Record...* menu option. Select **Record** as *Type* and **CSTPGMS-PARMS** as the part name.

4. Define program main function:

- Add **CSTCNS2-MAIN** as the main function for the program.
- Open the **CSTCNS2-MAIN** part (select *vgv4.web.codebase* as the target package for the new part).
- Converse the **CUSTUIS** UI Record (select **CONVERSE** as the I/O option and **CUSTUIS** as the I/O object).

5. Define required main function logic:

- The function logic should call the server program as part of a loop around the Converse of the UI Record.
- The loop should continue until the Exit submit request is received.
- The pre-defined server **CUSTPGMS** is called using the **CUSTPGMS-PARMS** UI Record as a parameter.
- The server program either returns data, updates data, or returns an error message in the **MESSAGE** field.
- If a successful "Find" was performed and the user has update capability (provided through the **STATEWS** WS Record) the **Update** button is displayed.

The required function logic is shown in Figure 85.

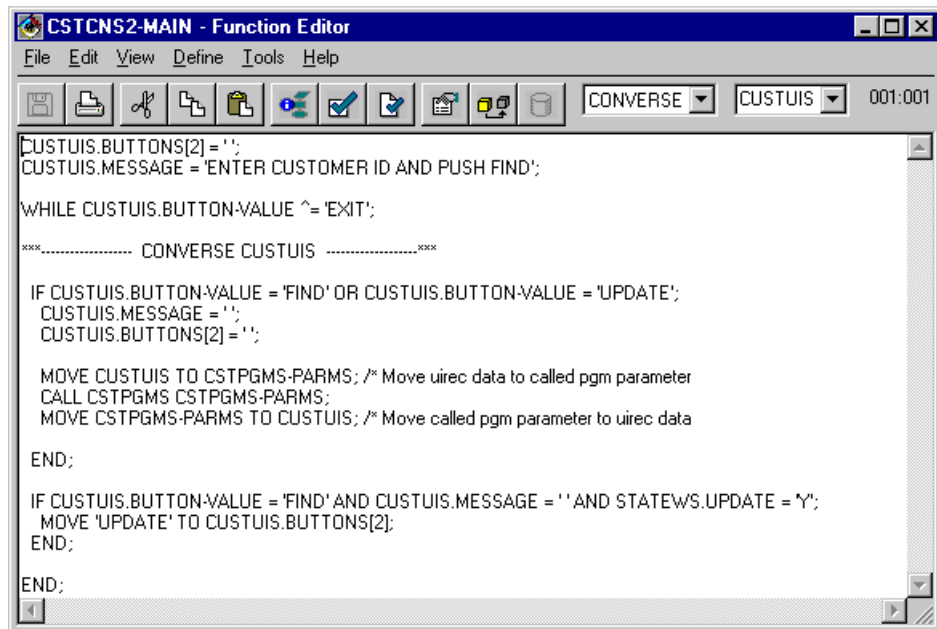


Figure 85. CSTCNS2-MAIN processing logic

6. Save the **CSTCNS2-MAIN** function and **CSTCNS2** program.
7. Use the test facility to test the **CSTIDUS** program. Choosing the **CNVWKST** option will trigger invocation of the **CSTCNS2** program.
 - Use the test facility to check the UI Record data item input values when interacting with the Web Transaction.
 - Note that the update profile information is no longer shown in the HTML code. Therefore, this is a very secure solution.
8. Version your code, once it is working.

8.2.2.2 XFER PGM model

In this exercise we will convert the **CSTXPS** Web Transaction into one (**CSTXPS2**) that accepts state information from the **CSTIDUS** Web Transaction and stores the state data in a working storage record.

CSTIDUS-MAIN function logic

To support the **CSTXP2** program, this function will pass the user type (update access) profile directly in the passed working storage record, where it will be kept during subsequent program logic.

Figure 86 shows the **CSTIDUS-MAIN** function logic relevant for this exercise.

```

IF CUSTUI_ID.BUTTON-VALUE = 'OK' AND CUSTUI_ID.TARGET-APP = 'XPWKST';
STATEWS.UPDATE = CSTIDUS-GET-PROF( CUSTUI_ID.USER); /* Get profile
CUSTUIS.BUTTONS[2] = '';
XFER CSTXPS2 STATEWS,CUSTUIS;
END;

```

Figure 86. CSTIDUS-MAIN processing logic (5)

Create Customer Info Web Transaction

1. Define the **CSTXPS2** Web Transaction program.:
 - Open the VAGen Parts Browser and add a new part.
 - In the New VAGen Part window, enter **CSTXPS2** for Part Name, select **Program** for Part Type, and select *vgv4.web.codebase* for the Package. Click on **OK**. This will open the program editor.
 - Select Web Transaction as program type.
2. Define **CUSTUIS** as the first UI Record for the new program; use mouse button 2 on Specifications and select the *Add First UI Record...* menu option. Select **CUSTUIS** as the first UI record.
3. Add **STATEWS** working storage to *Specifications* list.
Use mouse button 2 on the *Specifications* list and select the *Add Working Storage...* menu option. Select **STATEWS** as the part name.
4. Add **CSTPGMS-PARMS** to *Tables and Additional Records* list.
Use mouse button 2 on the *Tables and Additional Records* list and select the *Insert Table/Record...* menu option. Select **Record** as *Type* and **CSTPGMS-PARMS** as the part name.
5. Define program main function:
 - Use mouse button 2 on the Structure Diagram and select the *Insert Main Functions...* menu option. Enter **CSTCXP2-MAIN** as the part name.
 - Open the **CSTXPS2-MAIN** part (select *vgv4.web.codebase* as the target package for the new part).
6. Define required main function logic:
 - Unless the **Exit** button is pushed, the pre-defined server **CUSTPGMS** is called using the **CUSTPGMS-PARMS** UI Record as a parameter.
 - The server program either returns data, updates data, or returns an error message in the **MESSAGE** field.

- If a successful "Find" was performed and the user has update capability (provided through the **STATEWS WS Record**), the **Update** button is displayed.
- To create a loop, the program invokes itself using an XFER Pgm statement.

The required function logic is shown in Figure 87.

```

CSTXPS2-MAIN - Function Editor
File Edit View Define Tools Help
EXECUTE 001:001
IF CUSTUIS.BUTTON-VALUE ^= 'EXIT';
CUSTUIS.MESSAGE = 'ENTER CUSTOMER ID AND PUSH FIND.';
IF CUSTUIS.BUTTON-VALUE ^= '';
CUSTUIS.MESSAGE = '';
MOVE CUSTUIS TO CSTPGMS-PARMS; /* Move UIRec data to called parameter
CALL CSTPGMS CSTPGMS-PARMS;
MOVE CSTPGMS-PARMS TO CUSTUIS; /* Move called parameter data to UIRec
END;
IF CUSTUIS.BUTTON-VALUE = 'FIND' AND
CUSTUIS.MESSAGE = '' AND
STATEWS.UPDATE = 'Y';
CUSTUIS.BUTTONS[2] = 'UPDATE '; /* enable Update request
ELSE;
CUSTUIS.BUTTONS[2] = ''; /* disable Update request
END;
XFER CSTXPS2 STATEWS,CUSTUIS;
END;

```

Figure 87. CSTXPS2-MAIN processing logic

7. Save the **CSTXPS2-MAIN** function and the **CSTXPS2** program.
8. Use the test facility to test the **CSTIDUS** program. Choosing the **XPWKST** option will trigger invocation of the **CSTXPS2** program.
 - Use the test facility to check the UI Record data item input values when interacting with the Web Transaction.
 - Note that the update profile information is no longer shown in the HTML code. Therefore, this is a very secure solution.
9. Version your code, once it is working.

8.2.3 Self-managed state implementation (XFER ' ' model)

The XFER ' ' model does not support the use of a working storage record on the transfer request. This means that with the default behavior of VisualAge Generator XFER ' ' Web Transactions, the only way state data can be passed is in the UI Record (where it will be visible in the HTML sent to the browser).

This may not satisfy some requirements either when the data is sensitive, or when the volume of data to be stored (when hidden in the HTML) would delay network transport.

Secure state data support in an XFER ' ' model Web Transaction must be implemented in the program logic (see 5.3, "Implementing self-managed state support for XFER ' ' programs" on page 94 for theoretical and design considerations regarding self-managed state programming).

In this exercise we will convert the **CSTXBS** Web Transaction into one (**CSTXBS2**) that obtains state information from an application database, where it has been stored by the **CSTIDUS** Web Transaction.

CSTIDUS-MAI'N function and WTSPGM state management program

To support the **CSTXB2** program, this function will use the pre-defined program **WTSPGM** to store the state information an application database.

The **WT_STATE_WS** working storage record is passed to the **WTSPGM** program with the 'SAVE STATE' option stored in its **WTS_ACTION** data item. Access to the state data is obtained by passing the public part of the key in the data item **CUSTUI_IOS2.BLIND-KEY**. This will allow the target program (**CSTXB2**) to call **WTSPGM** to read the saved state data.

Figure 88 shows the **CSTIDUS-MAIN** function logic relevant for this exercise.


```

IF CUSTUI_ID.BUTTON-VALUE = 'OK'
  AND CUSTUI_ID.TARGET-APP = 'XBDB';

STATEWS.UPDATE = CSTIDUS-GET-PROF( CUSTUI_ID.USER); /* Get user profile
MOVE STATEWS TO WT_STATE_WS; /* Move state to server parm

WT_STATE_WS.WTS_ACTION = 'SAVE';
CALL WTSPGM WT_STATE_WS; /* save state

IF WT_STATE_WS.WTS_KEY_TMSTMP ^= ' ';
  MOVE WT_STATE_WS.WTS_KEY_TMSTMP TO CUSTUI_IOS2.BLIND-KEY;
  CUSTUI_IOS2.BUTTONS[2] = ' ';
  XFER ' ',CUSTUI_IOS2;
END;
MOVE "Problem saving state data" TO CUSTUI_ID.MESSAGE;

END;

```

Figure 88. CSTIDUS-MAIN processing logic (6)

Create input/output Customer Info UI Record

1. Create the UI Record **CUSTUI_IOS2** as a copy of the **CUSTUI_IOS** UI Record. Open the VAGen Parts Browser and use mouse button 2 on the **CUSTUI_IOS** UI Record and select the *Copy...* menu option.
2. Edit the **CUSTUI_IOS2** UI Record so that it meets the structure shown in Figure 89 (you will have to remove **CANUPDATE** from the program link parameters for the **FORM_IO** data item before you can save the record).

Name	Occ	Type	Len	Dec	UI Type	UI Properties	Usage	Descr
FORM_ID	1	Char	200	0	Form	Custom	Nonshared	
BUTTON-VALUE	1	Char	8	0	None	None	Nonshared	
CUSTID	1	Char	4	0	Input/Output	Custom	Nonshared	
FNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
LNAME	1	Char	30	0	Input/Output	Custom	Nonshared	
BANKID	1	Char	4	0	Input/Output	Custom	Nonshared	
BUTTON-SET	1	Char	24	0	Output	Default	Nonshared	
BUTTONS	2	Char	8	0	Submit	Custom	Nonshared	
BUTTON-EXIT	1	Char	8	0	Submit Bypass	Custom	Nonshared	
BLIND-KEY	1	Char	30	0	None	None	Nonshared	
MESSAGE	1	Char	70	0	Output	Default	Nonshared	

Figure 89. CUSTUI_IOS2 UI Record definition

3. Make these customizations to the **CUSTUI_IOS2** record:

- On the Program Link tab in the **FORM_IO** UI Properties window, define **CSTXBS2** as the Program and **CUSTUI_IOS2** as the First UI Record.
- In the Link Parameters list, insert a line using **BLIND-KEY** for Name and Value Item.

The passed value acts as a session identifier for the saved state data. This is safe to keep in the HTML, as it is not the actual state data.

Create Customer Info Web Transaction

1. Define the **CSTXBS2** Web Transaction program:

- Open the VAGen Parts Browser and add a new part.
- In the New VAGen Part window, enter **CSTXBS2** for Part Name, select **Program** for Part Type, and select *vgv4.web.codebase* for the Package. Click on **OK**. This will open the program editor.
- Select Web Transaction as program type.

2. Define **CUSTUI_IOS2** as the first UI Record for the **CSTXBS2** program; use mouse button 2 on Specifications and select the *Add First UI Record...* menu option. Select **CUSTUI_IOS2** as the first UI record.

3. Add **WT_STATE_WS** working storage to *Specifications* list.

Use mouse button 2 on the *Specifications* list and select the *Add Working Storage...* menu option. Select **WT_STATE_WS** as the part name.

4. Add **CSTPGMS-PARMS** to *Tables and Additional Records* list.

Use mouse button 2 on the *Tables and Additional Records* list and select the *Insert Table/Record...* menu option. Select **Record** as *Type* and **CSTPGMS-PARMS** as the part name.

5. Define program main function:

- Use mouse button 2 on the Structure Diagram and select the *Insert Main Functions...* menu option. Enter **CSTXBS2-MAIN** as the part name.
- Open the **CSTXBS2-MAIN** part (select *vgv4.web.codebase* as the target package for the new part).

6. Define required main function logic:

- Unless the **Exit** button is pushed the pre-defined server **CUSTPGMS** is called using the **CUSTPGMS-PARMS** UI Record as a parameter.
- The server program either returns data, updates data, or returns an error message in the **MESSAGE** field.
- If a successful "Find" was performed and the user has update capability, the **Update** button is displayed.
- The update capability is checked through the **WTSPGM** program, using the given **WT_STATE_WS** WS Record as a parameter. The 'READ STATE' option is now stored in the **WT_STATE_WS.WTS_ACTION** data item; also, the **BLIND-KEY** data item is used to identify the session.
- To create a loop, the program invokes itself using an XFER ' ' statement.

The required function logic is shown in Figure 90.

```

CSTXBS2-MAIN - Function Editor
File Edit View Define Tools Help
EXECUTE 001:001

IF CUSTUI_IOS2.BUTTON-VALUE ^= 'EXIT';
CUSTUI_IOS2.BUTTONS[2] = ''; /* disable Update request

IF CUSTUI_IOS2.BLIND-KEY = '';
CUSTUI_IOS2.MESSAGE = "Error setting user profile";
END;

IF CUSTUI_IOS2.BUTTON-VALUE ^= '';
CUSTUI_IOS2.MESSAGE = '';
MOVE CUSTUI_IOS2 TO CSTPGMS-PARMS; /* Move UIRec data to called pgm parameter
CALL CSTPGMS CSTPGMS-PARMS;
MOVE CSTPGMS-PARMS TO CUSTUI_IOS2; /* Move called pgm parameter data to UIRec
END;

WT_STATE_WS.WTS_ACTION = 'READ'; /* Get state data request
MOVE CUSTUI_IOS2.BLIND-KEY TO WT_STATE_WS.WTS_KEY_TMSTMP; /* Load blind key value
CALL WTSPGM WT_STATE_WS;

IF WT_STATE_WS.WTS_KEY_TMSTMP ^= '';
IF CUSTUI_IOS2.BUTTON-VALUE = 'FIND' AND
CUSTUI_IOS2.MESSAGE = '' AND
WT_STATE_WS.WTS_DATA = 'Y';
CUSTUI_IOS2.BUTTONS[2] = 'UPDATE '; /* enable Update request

ELSE;
CUSTUI_IOS2.BUTTONS[2] = ''; /* disable Update request
END;

ELSE;
CUSTUI_IOS2.MESSAGE = "Error reading user profile";
END;

XFER '' ,CUSTUI_IOS2;

END;

```

Figure 90. CSTXBS2-MAIN processing logic

7. Save the **CSTXBS2-MAIN** function and the **CSTXBS2** program.
8. Use the test facility to test the **CSTIDUS** program. Choosing the **XBDB** option will trigger invocation of the **CSTXBS2** program.
9. Version your code, once it is working.

Note: Really, our exit logic should call the **WTSPGM** state management program with a *DELETE* request, but we will leave this up to you to add. In the meantime, the data left in the database will help you analyze your test runs. One option for implementing delete processing is to have the active Web Transaction delete all expired data, in order to have a self-cleaning system.

Chapter 9. VisualAge Generator Templates Web Transactions

This chapter contains a scripted exercise that will help you understand how a Web-based application system can be implemented with VisualAge Generator Templates.

The VisualAge Generator Templates support in VisualAge Generator V4 includes a new system generator for Web Transaction systems. During this skill building exercise you will define, generate, test, and then enhance a VisualAge Generator Templates generated Web Transaction system. The following specific tasks are included:

- Import relational database definitions into the VisualAge Generator Templates information model.
- Define VisualAge Generator Templates information model entity definitions.
- Customize entity definition generation parameters to control the target package used during generation processing.
- Generate Workspace and entity definitions to create a Java Application System.
- Use the test facility to run a generated Web Transaction program and see the user interface implemented by VisualAge Generator Templates in a Web browser.

9.1 Preparing the workspace

Systems generated by VisualAge Generator Templates use the same architecture, and many of the same parts, for all system types. We need to remove some projects and create a new project for this lab exercise:

1. Before performing the VisualAge Generator Templates lab steps, any projects in the workspace that contain code generated by VisualAge Generator Templates must be removed.

If projects with VisualAge Generator Templates generated code are in the workspace, there will be processing errors when the common code parts are created during generation.

2. Stop, and then start, the VisualAge for Java workbench. We need to reset database connections to avoid DB2 access conflicts with VisualAge Generator Templates.

3. Create a new project and package for the VisualAge Generator Templates definitions:
Project — VGV4 VAGT Defs JGUI System
Package — vgv4.vagt.defs.jgui

9.2 Relational table definition using database import

We begin by acting as the DBA who is responsible for defining the VisualAge Generator Templates Relational Table entity definitions for the ITSOBANK database.

Note: For information on the ITSOBANK database, see Appendix A.3, “Database” on page 367.

1. Open a VisualAge Generator Templates browser and create a workspace:
 - Use the Workbench menu option *Workspace -> VAG Templates Browser*.
 - Select the *Create a new workspace* radio button.
 - Define a workspace name of *VGTJGUIWS*.
 - Select the package *vgv4.vagt.defs.jgui*.
 - Click on OK.

Note: We could use a separate package and even project, but to keep things simple, we will use the same package for all VisualAge Generator Templates definitions.

2. Import only the Bank and Customer tables from the ITSOBANK database:
 - Use the *VAG Templates Browser* menu option *Tools -> Import from Database...*
 - Define Search Criteria — Qualifier: VGDBA.
Note: You may need to use another qualifier if the tables were created under another ID. Check with the instructor.
 - Define *vgv4.vagt.defs.jgui* as the target package.
 - Click on *Build List*.
 - Move the Bank and Customer tables to the *Selected tables* list.
 - Click on *Import*.

Two relational table entities and eight data element entities have been defined.

9.3 Business Object definition

We now act as the programmer responsible for business logic implementation in the server domain. VisualAge Generator Templates Business Objects generate both server programs and reusable parts that exist in the client side domain (non-visual resource beans). These parts provide access to common business logic and the server programs.

1. Define the Business Object *BankBO*, using the *Bank* Relational Table, all data elements, with no extract criteria, and the data element *Bankid* as the sort criteria:
 - Use the menu option *Instance -> New...* to open the new VAGT Instance dialog.
 - Define a Business Object name of *BankBO*, an Instance type of *Business Object*, and a Package of *vgv4.vagt.defs.jgui*.
 - Using the *Tables and Fields* entry — add the *Bank* Relational Table and select all columns as fields for the business object.
 - Using the *Key and Criteria* entry — add *Bankid* as the sort criteria.
 - Close the *BankBO* business object definition.
2. Define the Business Object *CustomerBO*, using the *Customer* Relational Table, all data elements, with the data element *Bankid* as the extract criteria, and the data element *Custid* as the sort criteria:
 - Create a new Business Object instance named *CustomerBO* in the package *vgv4.vagt.defs.jgui*.
 - Using the *Tables and Fields* entry — add the *Customer* Relational Table and select all columns as fields for the business object.
 - Using the *Key and Criteria* entry — add *Bankid* as the extract criteria (greater than or equal) and *Custid* as the sort criteria.
 - Close the *CustomerBO* business object definition.

9.4 Interface Unit definition

We now act as the programmer responsible for definition of the system interaction and flow. VisualAge Generator Templates Interface Units define data content and linkage between Interface Units. When generated, visual parts are created to implement the desired processing. These parts use the non-visual components generated from Business Objects.

1. Define the Interface Unit *CDCustDtIU* as a simple IU that implements a detail view on the *CustomerBO* Business Object:
 - Create a new Interface Unit instance named *CDCustDtIU* in the package *vgv4.vagt.defs.jgui*.
 - Using the *Business Objects* entry — add the *CustomerBO* Business Object with a Layout Type of *detail*.
 - Close the *CDCustDtIU* Interface Unit definition.
2. Define the Interface Unit *CLCustLstIU* as a simple IU that implements a list view on the *CustomerBO* Business Object and targets the *CDCustDtIU* Interface Unit:
 - Create a new Interface Unit instance named *CLCustLstIU* in the package *vgv4.vagt.defs.jgui*.
 - Using the *Business Objects* entry — add the *CustomerBO* Business Object with a Layout Type of *list*.
 - Using the *Target Interface Units* entry — add the *CDCustDtIU* Interface Unit.
 - Close the *CLCustLstIU* Interface Unit definition.
3. Define the Interface Unit *BLBankLstIU* as a simple IU that implements a list view on the *BankBO* Business Object and targets the *CLCustLstIU* Interface Unit:
 - Create a new Interface Unit instance named *BLBankLstIU* in the package *vgv4.vagt.defs.jgui*.
 - Using the *General* entry — define the type as *root*.
 - Using the *Business Objects* entry — add the *BankBO* Business Object with a Layout Type of *list*.
 - Using the *Target Interface Units* entry — add the *CLCustLstIU* Interface Unit.
 - Close the *BLBankLstIU* Interface Unit definition.

9.5 Generation Option definition and system generation

We now act as the project leader responsible for definition of the processing options and target considerations that will control VisualAge Generator Templates generation. How code is generated to implement the required interface unit, business object, and relational table entity processing is defined using workspace and entity level generation parameters.

1. Define workspace generation parameters:
 - Use the *VAG Templates Browser* menu option *Workspace -> Definition* to open the workspace definition dialog for the *VGTJGUIWS* workspace.
 - Using the *Target Packages* entry — define the Target Project as *VGv4 VAGT JGUI System*.
 - Click on OK to close the workspace definition dialog.
2. Define Data Element Entity Default Generation Parameters:
 - Select the Data Element category in the *VAG Templates Browser*.
 - Use the *VAG Templates Browser* menu option *Entity -> Default Generation Parameters* to open the Default Generation Parameters dialog for the Date Element entity.
 - Using the *Target Packages* entry — define *vgv4.vagt.cmddata* as the Target Package for all package types.

If you wish, you can complete the definition of alternate package names at the workspace and entity level.

Regardless of your target package name choices, all packages will be created in the target project (defined as a workspace generation parameter).

3. Generate complete system with reusable code components:
 - Select the *BLBankLstIU* Interface Unit definition in the *VAG Templates Browser*.
 - Use the *VAG Templates Browser* menu option *Instance -> Generate...* to open the Generate Interface Units dialog for the *BLBankLstIU* Interface Unit.

- Select a generator to use for the *BLBankLstIU* Interface Unit:
 - *Interface Unit Java Visitor* — to build a Java Application system
 - *Interface Unit Web Visitor* — to build a Web Transaction system
 - *Interface Unit TUI Visitor* — to build a Text map system

You need to select the *Web Visitor* for the purposes of this redbook exercise. The text map system is useful for validating a VisualAge Generator runtime environment (the TUI Visitor should be available in Fixpak 1).

- Define these settings:
 - Store Options — *Normal*
 - Generation Scope — *With associates and predefined beans*
 - Client/Server — *Visuals, Client, and Server*

Notes:

- The Java Application system can take much more time to generate (there are many more components). Only choose this option, in addition to the others, if you can start the generation request and leave the machine alone (or if you have a very fast processor).
- You can always skip generation and load the generated system (see below for instructions).
- Click on the Generate push button to start generation.

Note: 180+ builders will run (each generates a small part of the system). This can take time on slow machines (slow chip/low memory).

Follow these steps to bypass the VisualAge Generator Templates generation processing delay and directly load the generated system:

- In the Workbench, choose the *Selected -> Import...* Workbench menu option. Select *Repository* as the import source and click on the *Next >* push button.
- Choose Local repository as the import option and use the Browse push button to find this file:

`<localdrive>:\VAGen\VAGTWeb.dat`

- Select the *Projects* radio button and use the *Details...* push button to import:

VG4 VAGT JGUI System — Base Gen 1.1

Note: Select the *Add most recent project edition to workspace* toggle to automatically load the imported project. If you did not do this, use the *Selected -> Add -> Project...* Workbench menu option to add the imported project version to the workspace.

9.6 Test the generated system

We can now test the Web Transaction system generated by VisualAge Generator Templates.

The entry point to the system is the root interface unit *BLBankLstIU*.

1. Find the Web Transaction programs in the VAGen Parts Browser.
2. Test the **BLBANWE** program:
 - When the Bank List has been displayed, click on the *CLCustLstIU* hyperlink.
 - When the Customer List has been displayed, click on one of the Cust ID hyperlinks.

The hyperlink starts the Web Transaction program created for the *CLCustLstIU* target interface unit. From this page, you can open up customer detail views using the hyperlinks available for each key value.

3. If you generated the VisualAge Generator Templates system, complete the required library management tasks for the VisualAge Generator Templates entity definitions and generated source code:
 - Version the **VGV4 VAGT Defs JGUI System** project with the version name **Base JDefs 1.0**.
 - Version the **VGV4 VAGT JGUI System** project with the version name **Base JDefs 1.0**.

9.7 Customization

By design, the Customer List should be restricted to a specific bank. To implement this processing, we need to enhance the Bank List UI Record:

1. Open the UI Record part **BLBANRW-UI-PAGE**, the Bank List UI Record.
2. Review the program link attributes for the **CLCUSWE-LINK / CLCUSWE** data item.
3. Open the **CLCUSWE** program definition and identify the first record that is used.
4. Adjust the **BANKBO-INSTANCE / BANKBO-BANKID** data item so that it implements a program link to the same program as the **CLCUSWE-LINK / CLCUSWE** data item (**CLCUSWE**).
5. Add to this program link definition the ability to pass the BankID value to the first record used by the **CLCUSWE** program.

The customized **BANKBO-BANKID** data item program link settings are shown Figure 91.

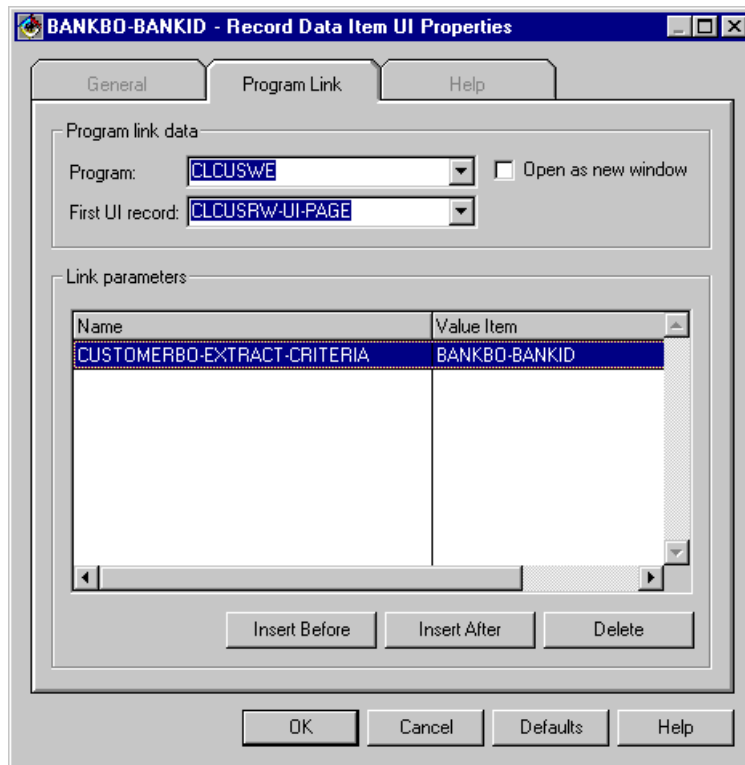


Figure 91. Program link settings for BANKID

6. Test the **BLBANWE** program.

When the Bank List has been displayed, click on one of the Bank ID hyperlinks.

7. Adjust the program link definition to open a new window.

8. Test the **BLBANWE** program again.

Note: The Customer List (*CustomerBO*) business object extract criteria are defined as greater than or equal (see 9.3, “Business Object definition” on page 175). This allows the system to function before the customized link has been defined. To be sure you can tell if your program link is functioning, choose a BankID value that is high in the sort order (such as REDB). When all the listed customers have BankID values equal to or greater than the selected BankID, then the program link is functioning.

Chapter 10. Demonstration system

The demonstration system discussed in this section was developed to support ITF and runtime testing of the available Web Transaction program structures, transfer processing options, and UI Record definition options.

The following Web Transaction programming concepts are covered:

- Transfer processing and data management
- State management
- Input validation

The demonstration system does not access a database or call server programs. You may find it useful as you evaluate the system processing implications of different Web Transaction implementation options.

Notes:

- Load version *Final 4.25.a*, or *Final 4.25.b* of the *z.vgv4.web.tests* package from the *DemoSys.dat* file (see A.1, “VisualAge Generator code” on page 365) to use the demonstration system code referenced in this section.
- You can add this package to the *VG4 Redbook WebTran Programs* project if you performed the exercises in Chapter 8, “Developing Web Transaction programming skills” on page 127.

10.1 Components

The major components of the demonstration system are shown in Figure 92.

Icon	Part Name	Description <
	CONV_UI_RECORD	Converse Model UI Rec - Active Beans
	CONVMOD	Converse Model WebTran
	FRST_FRM_RECV_UI	Single Segment - XFER ", UIRec - First Record (input)
	FRST_FRM_UI_RECORD	Single Segment - XFER ", UIRec - Output UI Rec
	FRSTFRM	Single Segment - XFER ", UIRec Model
	FRST_PGM_UI_RECORD	Single Segment - XFER PGM WS, UIRec - Active Beans
	FRSTPGM	Single Segment - XFER PGM WS, UIRec - Active Program
	FRST_PLK_UI_RECORD	Single Segment - Implicit XFER PGM - UIRec for Input
	FRSTPLK	Single Segment - Implicit XFER PGM with Params to UIRec

Figure 92. Demonstration system: Web Transactions and UI Records

There are two versions of the FRSTFRM program: One uses different UI Records for input and output (*Final 4.25.b*).

10.2 Processing overview

Each Web Transaction program in the demonstration system implements loop processing and can transfer control to the other Web Transaction programs (see Figure 93).

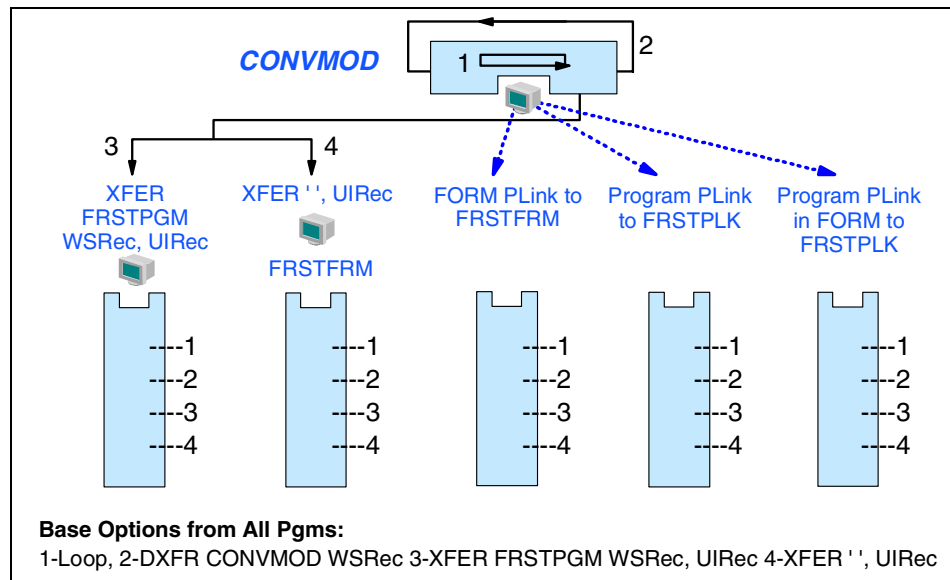


Figure 93. Demonstration system: transfer control

The following transfer processing statements are used:

```
DXFR CONVMOD WEB-COMM-WRK;
XFER FRSTPGM WEB-COMM-WRK, FRST_PGM_UI_RECORD;
XFER ' ', FRST_FRM_UI_RECORD;
```

These additional techniques, based on definitions in the Converse model UI Record (CONV_UI_RECORD), are used:

- Program link defined to start FRSTPLK program
- Form defined to start FRSTFRM program
- Program link defined in Form to start FRSTPLK program

A program link to FRSTPLK is also defined in FRST_FRM_UI_RECORD and FRST_PLK_UI_RECORD.

Note: Version *Final 4.25.a* of the demonstration system uses a single UI Record for input and output processing for the XFER ' ' sample program. Version *Final 4.25.b* uses two different UI Records.

10.3 Transfer processing and data management

Transfer processing and data management considerations are discussed in this section.

Transfer processing — Each program in the demonstration system provides the ability to loop or jump to one of the other sample programs. In addition, some UI Records are defined with program links that will directly invoke the FRSTPLK program (sometimes in a new window).

Data management — Each program passes data forward using either a passed working storage record, moves to a UI Record, or program link parameters. The technique used depends on the type of transfer or program invocation. A common data item is passed through to almost all target Web Transaction programs using different techniques.

Figure 94 shows an overview of the transfer processing and data management as implemented in the demonstration system. Included in almost all transfers is access to the same data values (common data) in all Web Transaction programs.

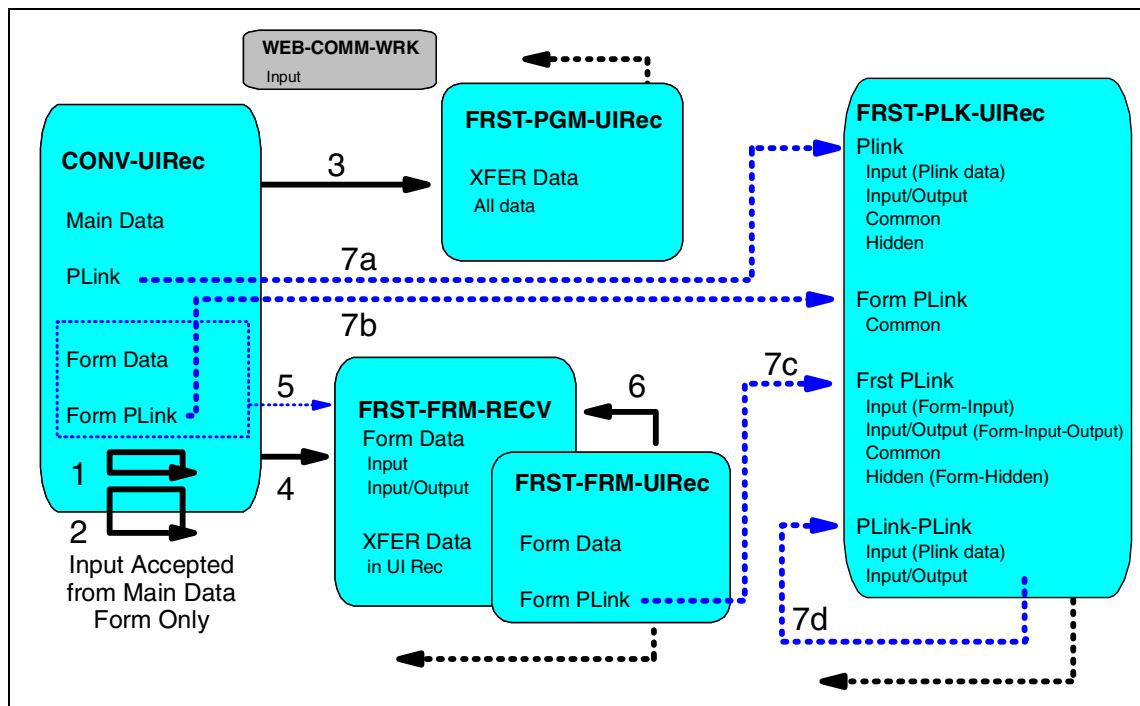


Figure 94. Demonstration system: transfer processing and data management

Data transfer rules for each processing option shown in Figure 94 are described below:

1. Internal loop to CONVERSE.
2. DXFR CONVMOD WEB-COMM-WRK;
Data to be passed is stored in the working storage record.
3. XFER FRSTPGM WEB-COMM-WRK, FRST_PGM_UI_RECORD;
Data to be passed is stored in the UI Record and working storage record.
4. XFER ' ' , FRST_FRM_UI_RECORD;
Data to be passed is stored in the UI Record.
5. CONV_UI_RECORD defined form for FRSTFRM program:
Data to be passed is first mapped to the UI Record for the target program (by name for input and input/output data) and referenced in program link parameters (COMMON and FORM-NONE data items).
6. Defined form in FRST_FRM_UI_RECORD:
Data to be passed is first mapped to the UI Record for the target program (by name for input and input/output data) and referenced in program link parameters (COMMON data item).
7. Program Links for FRSTPLK:
 - a. In CONV_UI_RECORD — opens new browser
 - b. In defined form in CONV_UI_RECORD — uses same browser
 - c. In FRST_FRM_UI_RECORD — opens new browser
 - d. In FRST_PLK_UI_RECORD — opens new browser

Each program link passes different parameters (shown in Figure 94).

10.4 State management

The state management options explored in the demonstration system are discussed in Table 6.

Table 6. State management in the demonstration system

Program	State management considerations
CONVMOD	All data is saved, all UI Record data item types are available after browser interaction. (All UI Record fields contain data when control returns to the program because of the active beans).
FRSTPGM	Only data passed in the UI Record and working storage record are saved. (All UI Record fields contain data when control returns to the program because of the active beans).

Program	State management considerations
FRSTFRM	<p>Only input and input/output data passed in the UI Record and sent to the browser is returned to the program (no bean is saved, so UI Types Output, Hidden, and None do not return to the program unless specified as program link parameters).</p> <p>When the FRSTFRM program is invoked by the defined form in the CONV_UI_RECORD UI Record, the data items COMMON and FORM-NONE are passed as program link parameters. The defined form in FRST_FRM_UI_RECORD only passes the COMMON data item as a program link parameter.</p>
FRSTPLK	<p>Only data identified in the program link is passed to the first UI Record for the target program. Data entered by the end user is not passed. When UI Record fields are defined as part of a program link, the values are hardcoded when the HTML is sent to the browser.</p>

10.5 Input validation

Several types of defined and logic edits are implemented in the demonstration system to show how editing works for the different Web Transaction types. The UI Record edits defined for data items used in the demonstration system are identified in Table 7.

Table 7. UI Record data item edits in demonstration system

UI Record	Data Item	Edits
CONV_UI_RECORD	INPUT (Non-shared data item)	CONV-REC-INPT-EDT edit function. Checks input value, and can then either set up an error message or modify visible or non-visible data values in UI Record.
	INPUT_OUTPUT (Non-shared data item)	Minimum input (14 characters). Data value is preloaded with 13 characters, but edit will only fire if data content changes value.
FRST_FRM_UI_RECORD FRST_FRM_RECV_UI	FORM_INPUT (shared data item)	FRST-FRM-INPT-EDTW edit function. Runs on Web. Checks input value and can then either set up an error message or modify visible/non-visible data values in UI Record.
	FORM_INPUT_OUTPUT (shared data item)	FRST-FRM-INPT-EDTH edit function Runs on host. Checks input value and can then either set up an error message or modify visible or non-visible data values in UI Record.

The edits defined demonstrate how UI Record edit processing is implemented and how different program structures can impact the triggering of edits and the resulting re-display of the UI Record in the browser.

10.6 Testing path

The following scripted testing exercise for the demonstration system will allow you to experience and further investigate the Web Transaction programming options that exist.

1. Start testing the CONVMOD program using these transfer request options:

Continue Loop Program loop returns to current UI Record display

Complete State DXFR CONVMOD WEB-COMM-WRK

These transfer options are implemented by program logic. The end user request is translated into the appropriate transfer command.

Note that the test facility has paused on the CONVERSE for CONVMOD during each browser interaction.

On the target runtime platform, all program data would be saved during the CONVERSE and the program would be terminated. End user interaction restarts the program, with all data, at the point of the active CONVERSE statement.

2. Enter data in the two input fields (Input and Form Input), then select either **Continue Loop** or **Complete State** and the **Process / Enter** push button.

Only the data in the input field that is part of the CONV_UI_RECORD generated form is passed back to the CONVMOD program.

Data entered in the form input field that is part of the defined form structure does not get passed back to the CONVMOD program.

(This would be a good time to review the CONV_UI_RECORD UI Record definition.)

3. Return to the CONVMOD (complete state) program and reset the data values. This is done by selecting the toggle named **Boolean Edit Creates Toggle** which will trigger a data reset in the CONVMOD program logic.

4. Starting at the CONVMOD program CONVERSE, select this transfer request option:

Controlled State XFER FRSTPGM WS FRST_PGM_UI_RECORD

This transfer option is also implemented by program logic. The end user request is translated into the appropriate transfer command.

Note that once the next UI Record has been sent to the browser, the test facility has paused on the First UI Record for FRSTPGM with a pending restart (continuation) of the pre-scheduled program.

The target program starts after an XFER Pgm transfer and is in a wait state, similar to the one that occurs during a CONVERSE, when the UI Record is sent to the browser. On the target runtime platform, only the passed working storage record (if any) would be saved during the browser interaction, and the currently active program (FRSTPGM) would be terminated. End user interaction restarts the FRSTPGM program, with the passed working storage record data and the input from the First UI Record.

5. Return to the CONVMOD (complete state) program (**Complete State** option) and, if required, reset the data values (select the **Boolean Edit Creates Toggle**).
6. Starting at the CONVMOD program CONVERSE select this transfer request option:

No State XFER ' ' ,FRST_FRM_UI_RECORD

This transfer option is also implemented by program logic. The end user request is translated into the appropriate transfer command.

Note that once the next UI Record has been sent to the browser, the test facility is empty (test complete). After an XFER ' ' transfer request has been issued and the UI Record has been sent to the browser, the test facility (and the current program) are done. There is no prescheduled program waiting to start. The program that should start next is identified in defined form in the FRST_FRM_UI_RECORD UI Record (see Figure 95).

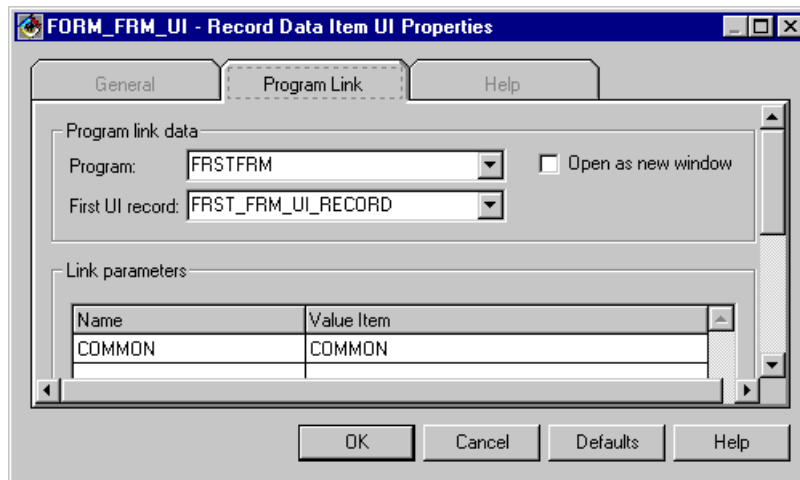


Figure 95. FRST_FRM_UI_RECORD defined form properties

The defined form identifies the target program and First UI Record along with any required program link parameters to pass data in addition to the input and input/output data items that exist in the defined form structure. In this example, the only program link parameter is the common data value.

On the target runtime platform, no program data would be saved, and the program would be terminated. The target program has run to completion, and the only data available has been sent to the browser in the UI Record.

7. Return to the CONVMOD program (**Complete State** option) and reset the data values (select the **Boolean Edit Creates Toggle**). You will need to tell the test facility to continue twice to reach the CONVERSE.
8. Enter some data in the CONV_UI_RECORD defined form fields (FORM-xxx fields in the lower portion of the browser) and then start the FRSTFRM program using the **Go Action** push button (tell the test facility to continue).

This transfer option is implemented by the defined form in the UI Record CONV_UI_RECORD (see Figure 96).

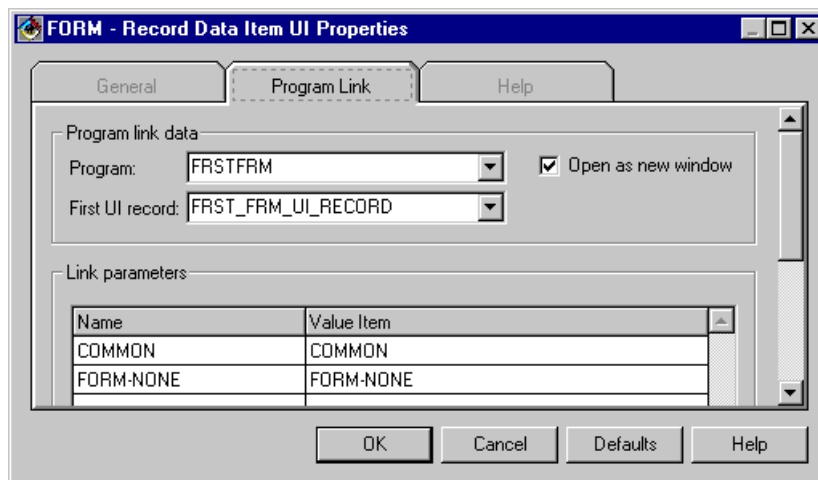


Figure 96. CONV_UI_RECORD defined form properties

The defined form identifies the target program and First UI Record along with any required program link parameters to pass data in addition to the input and input/output data items that exist in the defined form structure. In this example the defined form includes the option to start the target program in a new browser.

The new browser includes the data you entered in the defined form fields and the common data passed as part of the program link parameters.

Note that a new instance of the test facility is started to support the new program thread (*Open as new window* setting in Figure 96). Once the program has run to completion (XFER ' ' statement), the test facility instance is unused.

If you return to the original browser view (active CONVERSE) and review the page source (you may have to save the page to a file), you can find the defined form and see the following common data reference in the program link:

```
<FORM METHOD="POST"
ACTION="http://localhost:8080/hptGateway?hptAppId=FRSTFRM&execute=1&record=FRST_FRM_
UI_RECORD" TARGET="_blank">
<INPUT TYPE=HIDDEN NAME="COMMON" VALUE="zCommon Data">
  <INPUT TYPE=HIDDEN NAME="FORM-NONE" VALUE="zForm-None">
  <b>This starts the defined form. <br>Form Output Field: </b> zForm-Output
  <br>
  <INPUT TYPE=HIDDEN NAME="FORM-HIDDEN" VALUE="zForm-Hidden  ">
  <b>Form Input :</b> Enter <b>Hello</b> or <b>Hello Dolly</b> to trigger Edits
</b><INPUT TYPE=TEXT NAME="FORM-INPUT" SIZE=40 MAXLENGTH=40 VALUE="zForm-Input">
  <br>
  <b>Form Input/Output </b><INPUT TYPE=TEXT NAME="FORM-INPUT-OUTPUT" SIZE=25
MAXLENGTH=25 VALUE="zForm-Input-Output ">
  <br>
  <A
  HREF="http://localhost:8080/hptGateway?hptAppId=FRSTPLK&execute=1&record=FRST_PLK_UI
_RECORD&COMMON=zCommon+Data">Form Plink- w/Common Data passed as Pgm Link Parm</A>
  <br>
  <INPUT TYPE=SUBMIT NAME="FORM-SUBMIT" VALUE="Go Action"><INPUT TYPE=SUBMIT
NAME="FORM-SUBMIT-BYPASS" VALUE="End/Cancel Action">
  <br>

</FORM>
```

On the target runtime platform, no program data would be saved, and the program would be terminated. The target program has run to completion and the only data available has been sent to the browser in the UI Record.

- Return to the CONVMOD (complete state) program in the first browser, enter some data in the input-output field, and select this hyperlink transfer request option:

UIRec Pgmink... Starts FRSTPLK, input in FRST_PLK_UI_RECORD

Remember to tell the test facility to start (required on all defined form or program link based transfers).

This program link starts the program FRSTPLK, with hardcoded values as input, and with a new browser for the displayed UI Record. The program link definition is shown in see Figure 97.

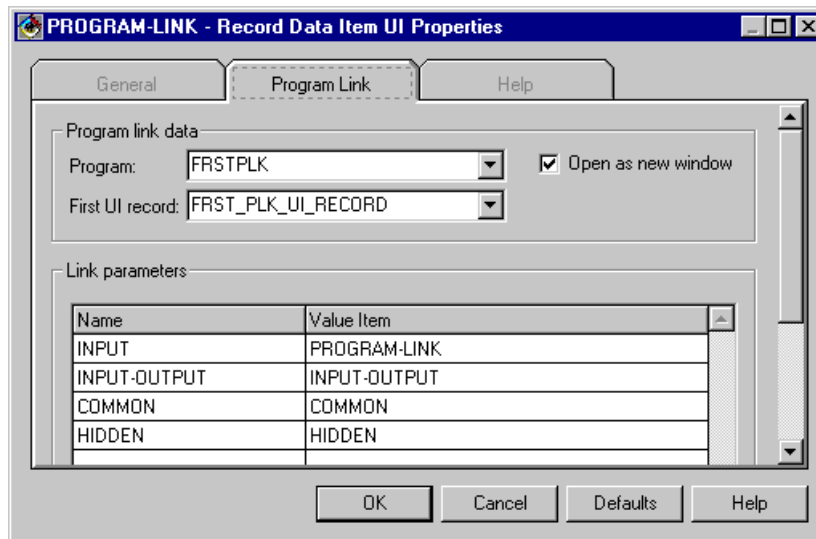


Figure 97. CONV_UI_RECORD program link definition (1)

The data you entered in the input-output field is not passed, even though the program link parameters specific the field. With a program link, only the values present in the HTML when it is first sent to the browser can be returned to the target program. If you enter the data and loop through the CONVMOD program once first, the data values will then be sent forward to the target program FRSTPLK.

10. Close all browsers except the one with the active CONVERSE.
11. Return to the CONVMOD (complete state) program in the first browser and select this hyperlink transfer request option:

Form Plink... Starts FRSTPLK, input in FRST_PLK_UI_RECORD

This program link starts the program FRSTPLK, with only the common data passed forward. The same browser is reused in this program link example (see Figure 98).

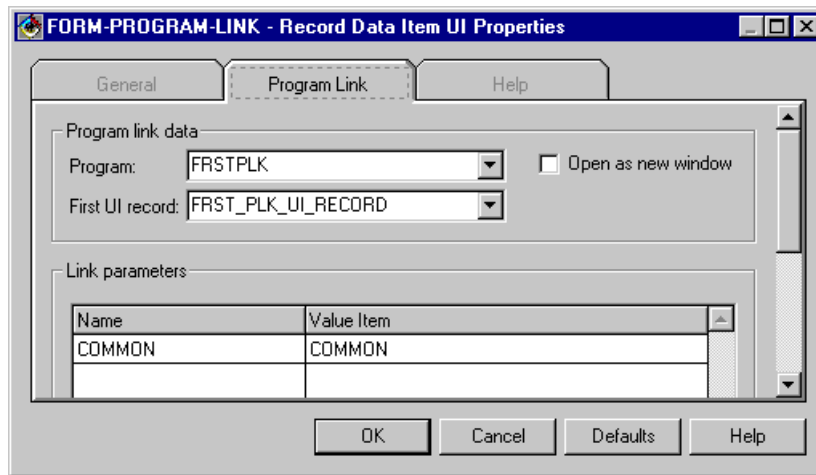


Figure 98. CONV_UI_RECORD program link definition (2)

There should now be only one active browser, but two open test facility monitors! The program link opened a new test facility, which replicates the scheduling of a new program, but the settings forced the reuse of the existing browser. Where do you think that active program waiting on a CONVERSE is now? Stranded would be one answer (see the discussion related to data lost after program link during CONVERSE, as shown in Figure 29 on page 91, for additional details).

Note: You can get back to the active CONVERSE using the back button in the browser followed by a refresh. This works because a conversed UI Record has active beans in session data.

The FRSTPLK program presents a UI Record that only uses program link options to start the next Web Transaction. Some of these program links do not return the common data to the next Web Transaction. Use the program data view in the test facility to find out which one loses the common data value. (You could also just inspect the HTML in the browser; learning to read this will help you debug your own Web Transaction programs.)

Review the FRST_PLK_UI_RECORD definition to see where the decision not to pass data forward was made (missing program link parameter).

12. Return to the CONVMOD (complete state) program and reset the data values (select the **Boolean Edit Creates Toggle**).

13. Delete the last character in the Input/Output field and replace it with the same character. Then select either **Continue Loop** or **Complete State** and select the **Process / Enter** push button.

Nothing happens, right? Nothing should. Even though a minimum input edit is defined for the INPUT-OUTPUT field, the edit will not trigger as the data value has not changed.

14. Delete the last character in the Input/Output field and replace it with a different character. Then select either **Continue Loop** or **Complete State** and select the **Process / Enter** push button.

The defined minimum input edit will now trigger. as the data value has changed. Add another character to the field and the edit will pass.

Note: See 5.4.2, “UI Record edits” on page 100 for a complete description of edit processing.

15. Test the Input field edit function using this process:

- Enter the text *Hello* in the Input field.
- Click on the stop icon in the test facility so you can see what happens when the program starts executing again.
- Select either **Continue Loop** or **Complete State** and select the **Process / Enter** push button.

Because we have modified the input field data, the edit function CONV-REC-INPT-EDT will run after the CONVERSE, but before the main line logic. Because the field is marked as in-error as a result of the EZEUIERR error message, this edit function will be triggered every time (until an EZEUIERR statement is not processed).

16. Test the use of an edit function that does not trigger an error:

- Enter text *Hello Dolly* in the input field.
- Click on the stop icon in the test facility so you can see what happens when the program starts executing again.
- Select either **Continue Loop** or **Complete State** and select the **Process / Enter** push button.

Because we have modified the input field data, the edit function will run. The field is not marked in-error, so the post-CONVERSE logic will run after the edit function. The edit function will not be triggered after the next browser response (unless the data is modified) because an EZEUIERR statement was not processed. This is how it works for UI Records that have active beans (CONVERSE or XFER Pgm Web Transactions).

17. To see how edit processing works for an XFER ' ' Web Transaction, select the **Go Action** push button that is part of the defined form in the CONV_UI_RECORD.

A new test facility instance will start and two edit functions will triggered:

FRST-FRM-INPT-EDTW Defined for FORM-INPUT data item in First UI Record. Run on Web attribute selected.

FRST-FRM-INPT-EDTH Defined for FORM-INPUT-OUTPUT data item in First UI Record. Run on Web attribute not selected.

Edits for an XFER ' ' Web Transaction structure are based on the definition of the First UI Record associated to the target program (not the UI Record referenced on the XFER ' ' statement or the data item definitions for fields that are in the defined form).

The defined edits and edit functions will always run, regardless of whether the field has been modified or not. This is because there are no active beans to use when comparing new and old data values. All edits must run in this situation.

Note: See Table 4 on page 102 for additional information.

18. Use this process to demonstrate some of the capabilities of edit functions in an XFER ' ' Web Transaction:

- Enter text *Hello* in the form input field of the new browser display.
- Select the **Go Action** push button.

The test facility start and edit processing will be triggered. The value *Hello* triggers certain logic in the Web-side edit routine. The fact that the Web-side edit function has run is displayed in an output field in the browser.

- Enter text *Hello Dolly* in the form input field and select **Go Action**

The value *Hello Dolly* triggers certain logic in both edit routines (see the results in the output field in the browser).

Additional capabilities exist in the demonstration system. Use your imagination during test cycles and check both the program data and HTML source in the browser to see what else you can discover.

Note: While this exercise was written for the VisualAge Generator test facility, some options are better demonstrated in a runtime system (for instance, the EZEUSR, EZEUSRID and EZELTERM values are not very meaningful, or do not exist, when using the test facility). You may wish to follow the exercise again after you have created a runtime version of the demonstration system.

Chapter 11. Front-end customization techniques

While the generated default JSPs function, they probably do not represent the final presentation view that you want for your system. Changes to the generated default JSPs are permitted (if not expected). Any valid HTML tags or scripting languages, such as client-side JavaScript, may be added to the JSPs, as well as any valid 1.0 JSP syntax.

Basic changes can be used to improve the visual display. Other common enhancements include the implementation of help processing or support for dynamic protection of FORM elements in the JSP logic.

11.1 Level 1: What's a Web Transaction developer to do?

In the level 1 development approach, the Web Transaction developer(s) must work alone, having been given the responsibility of developing and generating a working system as well as tweaking the resulting front end to create a finished Web-based application system.

We will use the simple Customer Info system implemented in a previous exercise (see 8.1.2, "Converse model programming" on page 128) to demonstrate how basic modifications can be made to the generated default JSP. To follow this exercise, you must:

- Generate and implement a runtime environment for the CSTCNV Web Transaction and CUSTPGM server program (see Chapter 15, "Web Transaction generation" on page 335, and Chapter 13, "VisualAge Generator Web Transaction runtime setup" on page 257).
- Configure the WebSphere Test Environment in VisualAge for Java (see 14.4, "VisualAge for Java WebSphere test environment" on page 329).
- Have access to a simple text editor (the examples will use Notepad for Windows). Keep in mind that any text editor can be used, as long as it does not add any unnecessary tabs and spaces, and it maintains the original text file integrity.

11.1.1 Correcting the generated default JSP

These items may not be required, depending on which fixpak level you are running. If required, edit the generated JSPs to correct invalid syntax:

- Change any "imports =" directives to "import =" (drop the s).
- Remove any create="false" clauses inside useBean tags.

The create clause was only found in the Vagen1EntryPage.jsp file.

A JSP can be read and rendered by tools such as Netscape (browse for the JSP using file `://C:/` and open the JSP file) and on the preview page in WebSphere Studio Page Designer. Sometimes a complex JSP does not render that well, and the syntax remains as text (see Figure 99).

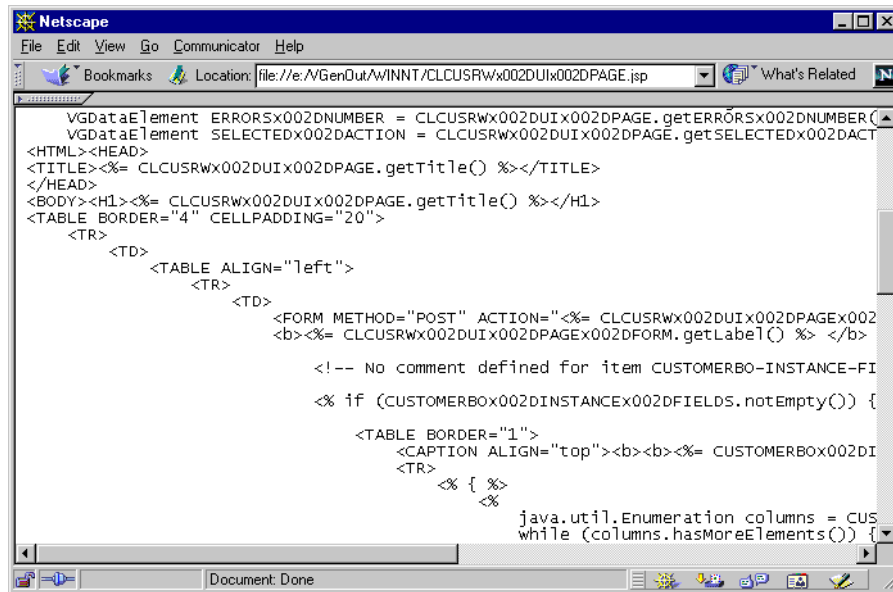


Figure 99. Enhancing JSP rendering: Before

Edits to the generated JSP can be used to enhance the rendering of the JSP in some development tools. Tools can better interpret the JSPs if the `<HTML><HEAD>` is moved above the `<jsp:useBean` tag:

```
<%@ page import = "com.ibm.vgj.uibean.VGDataElement" %>
<HTML><HEAD>
<jsp:useBean id="CONV_UI_RECORD" scope="request"
class="z.vgv4.web.tests.genout.CONV_UI_RECORDBean" />
<!--
This is JAVA code that gets the individual data elements
from the UI Bean that are to be used by this page to
access all dynamic data. -->
<%
    VGDataElement OUTPUT = CONV_UI_RECORD.getOUTPUT();
    ...
    VGDataElement FORMx002DSUBMITx002DBYPASS =
CONV_UI_RECORD.getFORMx002DSUBMITx002DBYPASS(); %>
<TITLE><%= CONV_UI_RECORD.getTitle() %></TITLE>
```

This change to the order of the syntax results in formatting of the JSP into pseudo presentation mode (see Figure 100).

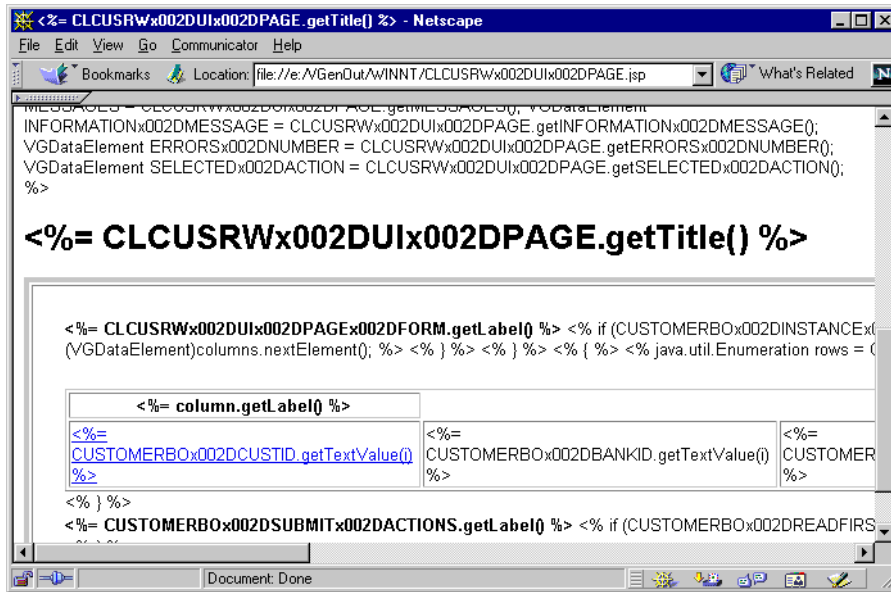


Figure 100. Enhancing JSP rendering: After

Generation processing for default JSPs may be modified in future releases of Generator to move the `<HTML><HEAD>` tags to the beginning of the file.

This movement of the `<HTML><HEAD>` tags also impacts how the JSP is shown in the WebSphere Page Designer.

Update Information

As of this writing, Fixpak 1 for VisualAge Generator V4 has included processing during generation which better positions the `<HTML>` and `<HEAD>` tags in the generated default JSP. You will still have to make a similar change to the VAGen prefixed JSPs, even with Fixpak 1.

11.1.2 Easy elements

The Web Transaction developer should be able to take the default JSPs and re-order and/or rearrange the elements according to project requirements. When a .jsp source file is opened in a text editor (such as Notepad) or in an HTML editor that supports JSP development (such as WebSphere Studio's Page Designer), we can make edits to the ordering and arranging of the elements. For anyone who has worked with HTML before, there are not many surprises, except for the addition of Java code. For those who are not familiar

with HTML, please refer to Chapter 3, “HTML and UI Record definition” on page 55. You may also wish to find additional HTML documentation to study.

Once we can discern the basic structure of an HTML document, we can determine the types of customization that are desirable for our particular application. The default JSPs (and most well-formed HTML documents in general) contain the basic markup tags to form a minimal HTML page. These tags, which are: <HTML>, <HEAD>, <TITLE>, and <BODY>, form the framework of an HTML page, and are laid out in the following fashion:

```
<HTML>
<HEAD>
<TITLE>Page title goes here</TITLE>
</HEAD>

<BODY>

... HTML code of the page ...

</BODY>
</HTML>
```

Important Note

Keep in mind that any and all changes made to a generated default JSP will be **LOST** if the page is later re-generated! VisualAge Generator has nothing built into it to maintain any changes made to the generated JSP source during generation processing. If you are going to re-generate a UIRecord JSP, keep a backup copy of your existing file until you are sure you want the new generated page.

11.1.3 Implementing help

VisualAge Generator allows you to fill in help information for individual data items and for the whole UI Record. However, nothing is actually done with this information, other than to make it available in the data bean and accessible through the interface bean.

An example of implementing help is shown below. We use client side JavaScript in combination with JSP tags to open another browser window to display the help information for the UI Record CUSTUI when the user clicks on the button labelled Help.

Note: You may have to first add help text for the UI Record and the data items contained in the record, and then regenerate, before you have generated beans that contain the associated help text.

To implement this:

- Inside the HTML header tag in the generated JSP, add JavaScript code similar to that shown in Figure 101.
- Inside the FORM tag in the generated JSP, add JSP tags similar to that shown in Figure 102.

```
<SCRIPT LANGUAGE="javascript">
function showHelp()
{ newWindow = window.open("", "Help", "scrollbars=1,resizable=1,height=200,width=200")
newWindow.document.write("<HTML><HEAD><TITLE>Help for
CUSTUI</TITLE></HEAD>")
newWindow.document.write("<BODY><%= CUSTUI.getHelpText()
%></BODY></HTML>")
newWindow.document.close()}
</SCRIPT>
```

Figure 101. Implementing help — JavaScript

```
<INPUT TYPE="button" NAME="helpbutton" VALUE="Help" onClick="showHelp()">
```

Figure 102. Implementing help — JSP tags

We are making use of the events available for FORM fields, which are part of the specification for HTML and JavaScript; they are entirely outside the scope of Generator development. The INPUT of TYPE "button" has an onClick event which we can use to trigger a JavaScript function.

The modified UI Record view (the JSP in the browser) and the new window with the help text defined in the UI Record is shown in Figure 103.

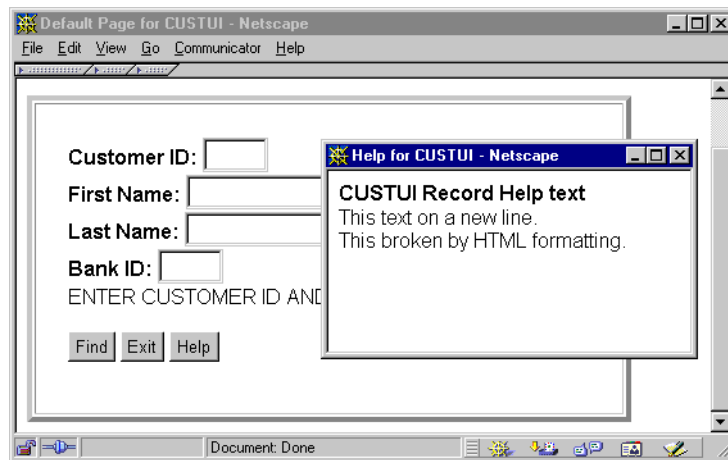


Figure 103. Customized JSP with help button and help window

Note: To get the text shown in the help window in Figure 103 we entered the following as a **single** line of text in the help page properties dialog for the CUSTUI UI Record:

```
<b>CustUI Record Help Text</b><br>This text on a new line.<br>This  
broken by HTML formatting.
```

If you use the Enter key to make a new line when adding help text, the `getHelpText()` method will fail. The `\n` added to the string in the generated Java bean source (`CUSTUIBean.java`) invalidates the method. We added some HTML formatting control tags (``, ``, and `
`) to further customize the help text view in the help window.

Other HTML INPUT TYPEs such as TEXT have similar events, for example, you could trigger data item help for a data item named `TXTFLD` of UI type INPUT or INPUT/OUTPUT to display in the browser status bar when the user tabs into, or focuses on a field, and remove the help when they tab out or focus on another field, by adding:

```
onFocus="window.status='<%= TXTFLD.getHelpText() %>'; return true"  
onBlur="window.status=' ' ; return true"
```

to the HTML INPUT tag, so the full tag in the JSP appears as shown below:

```
<!-- TXTFLD -->  
<b><%= TXTFLD.getLabel() %> </b>  
<INPUT TYPE=TEXT NAME="TXTFLD" SIZE=30 MAXLENGTH=30  
VALUE="<%= TXTFLD.getTextValue() %>"  
onFocus="window.status='<%= TXTFLD.getHelpText() %>'; return true"  
onBlur="window.status=' ' ; return true">  
<% if (TXTFLD.hasInputError()) { %>  
<BR><FONT COLOR="#FF0000"><%= TXTFLD.getErrorMessage() %></FONT>  
<% } %>  
<br>
```

11.1.4 Protecting FORM fields

You first need to define a 1-byte CHA data item in the UI Record for each data item you wish to protect. You may define these fields as UI type NONE, since they do not need to appear in the HTML.

You are only likely to see problems using NONE if you XFER ' ' ,UI Record with the UI Record containing the NONE data items and you have a FORM data item where the receiving UI Record is the same UI Record. In this case, you need to ask to pass through the NONE fields as link parameters when you specify the UI properties of the FORM data item, so the values are not lost. This of course turns the NONE field into a HIDDEN field so it can be referenced in the program link, and HIDDEN fields can be seen in the HTML source sent to the browser.

You need to set the flags as appropriate inside your Web Transaction code before the CONVERSE or XFER which displays the UI Record in the browser.

In the generated JSP, add code to look up the flag. For example, for a data item of type INPUT or INPUT/OUTPUT named TXTFLD, you could add:

```
<% if (FLAGFORTXTFLD.getTextValue().equals("Y")) {  
    out.print("READONLY"); } %>
```

to the INPUT field, so the full tag appears in the JSP as shown below:

```
<%= TXTFLD.getLabel() %> </b><INPUT TYPE=TEXT NAME="TXTFLD" SIZE=10  
MAXLENGTH=10 VALUE="<%= TXTFLD.getTextValue() %>"  
<% if (FLAGFORTXTFLD.getTextValue().equals("Y")) {  
    out.print("READONLY"); } %>
```

11.1.5 Making the default JSP look better

The UI Record provides the ability to control data content and basic presentation (input, output, lists, drop downs). Other presentation decisions are defined in the default JSP generated from the UI Record definition.

Figure 104 shows an example of the default page as generated by VisualAge Generator (we have returned to the original default JSP).



The screenshot shows a web browser window with the title "Customer Info". Inside the browser, there is a form with the following elements:

- Customer ID:
- First Name:
- Last Name:
- Bank ID:
- ENTER CUSTOMER ID AND PUSH FIND.
- Find Exit

Figure 104. Default Customer Info JSP

While the view in Figure 104 is functional, there can be no doubt that it could look better if we made a few changes. The easiest elements to modify will be the TABLE and FORM elements.

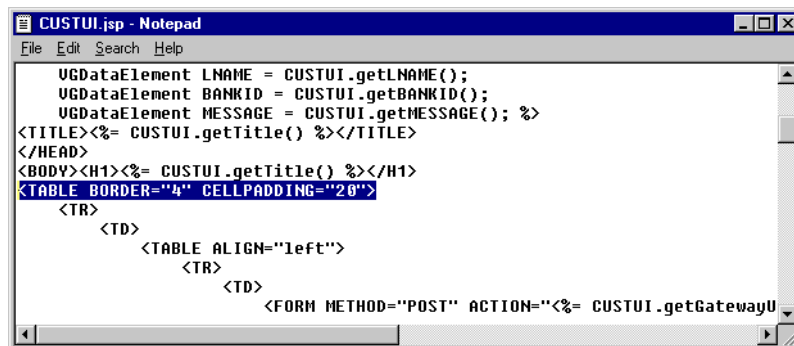
Modifying the TABLE

The primary element in the default JSP layout is the use of `TABLE`. A `TABLE` has many attributes that can be modified easily, namely `HEIGHT`, `WIDTH`, `BORDER`, `CELLSPACING`, and `CELLPADDING`.

The `TABLE` is set up like a grid with rows `<TR>` and cells `<TD>`. `HEIGHT` controls how much of the screen the table takes up vertically, and the value can be either pixels or percentage. `WIDTH` controls how much of the screen the table takes up horizontally, and can also be either pixel or percentage. `BORDER` controls the width of the border of the `TABLE` — this can be any number from 0 to N (N being a reasonable number greater than 0 — this is one of the recommended changes to make to the Generator defaults).

`CELLSPACING` controls how much space the browser will put between individual data cells, and is a pixel value. `CELLPADDING` represents the amount of padding the browser will render around content inside the data cells and it is also a pixel value.

So, if we want to create a `TABLE` that has a `BORDER` width of 1, only takes up 90% of the horizontal space of the browser, has no space in *between* cells, but has 5 pixels of padding around cell content, we could take the default code (see Figure 105) and make the desired changes (see Figure 106).



```
CUStUI.jsp - Notepad
File Edit Search Help
UGDataElement LNAME = CUSTUI.getLNAME();
UGDataElement BANKID = CUSTUI.getBANKID();
UGDataElement MESSAGE = CUSTUI.getMESSAGE(); %
<TITLE><%= CUSTUI.getTitle() %></TITLE>
</HEAD>
<BODY><H1><%= CUSTUI.getTitle() %></H1>
<TABLE BORDER="1" CELLPADDING="20">
  <TR>
    <TD>
      <TABLE ALIGN="left">
        <TR>
          <TD>
            <FORM METHOD="POST" ACTION="<%= CUSTUI.getGatewayU
```

Figure 105. Default code for HTML table

```

CUSTUI.jsp - Notepad
File Edit Search Help
UGDataElement LNAME = CUSTUI.getLNAME();
UGDataElement BANKID = CUSTUI.getBANKID();
UGDataElement MESSAGE = CUSTUI.getMESSAGE(); %>
<TITLE><%= CUSTUI.getTitle() %></TITLE>
</HEAD>
<BODY><H1><%= CUSTUI.getTitle() %></H1>
<TABLE BORDER="1" CELLPADDING="5" CELLSPACING="0" WIDTH="90%">
  <TR>
    <TD>
      <TABLE ALIGN="left">
        <TR>
          <TD>
            <FORM METHOD="POST" ACTION="<%= CUSTUI.getGatewayU

```

Figure 106. Modified code for HTML table

Figure 107 shows the JSP after modification of the TABLE definition.

Figure 107. JSP with TABLE modifications

Modifying the FORM

It is important to note that it really doesn't matter how the FORM elements are ordered as far as HTML is concerned — all INPUT elements of a FORM are submitted at the same time with their respective name/value pairs.

When we open up the source file in Notepad, we can easily identify the FORM elements (in this case there are only two types: INPUT and SELECT). To rearrange the individual elements, all you have to do is cut and paste the FORM element into its new position.

We have already seen the default source. To put the FORM elements in a different order, we simply highlight the element in question (see Figure 108).

```

CUSTUI.jsp - Notepad
File Edit Search Help

<!-- No comment defined for item FNAME -->
<b><%= FNAME.getLabel() %> </b><INPUT TYPE=TEXT
<% if (FNAME.hasInputError()) { %>
  <BR><FONT COLOR="#FF0000"><%= FNAME.getErr
<% } %>
<br>

<!-- No comment defined for item LNAME -->
<b><%= LNAME.getLabel() %> </b><INPUT TYPE=TEXT
<% if (LNAME.hasInputError()) { %>
  <BR><FONT COLOR="#FF0000"><%= LNAME.getErr
<% } %>
<br>

<!-- No comment defined for item BANKID -->
<b><%= BANKID.getLabel() %> </b><INPUT TYPE=TEXT
<% if (BANKID.hasInputError()) { %>
  <BR><FONT COLOR="#FF0000"><%= BANKID.getEr
<% } %>
<br>

```

Figure 108. Highlighting the FORM element to move

And then cut it (either with the **Edit** menu option or by pressing Ctrl+X), and paste it into its new position (again, using the **Edit** menu option or using Ctrl+V) (see Figure 109).

```

CUSTUI.jsp - Notepad
File Edit Search Help

<!-- No comment defined for item BANKID -->
<b><%= BANKID.getLabel() %> </b><INPUT TYPE=TEXT
<% if (BANKID.hasInputError()) { %>
  <BR><FONT COLOR="#FF0000"><%= BANKID.getEr
<% } %>
<br>

<!-- No comment defined for item FNAME -->
<b><%= FNAME.getLabel() %> </b><INPUT TYPE=TEXT
<% if (FNAME.hasInputError()) { %>
  <BR><FONT COLOR="#FF0000"><%= FNAME.getErr
<% } %>
<br>

<!-- No comment defined for item LNAME -->
<b><%= LNAME.getLabel() %> </b><INPUT TYPE=TEXT
<% if (LNAME.hasInputError()) { %>
  <BR><FONT COLOR="#FF0000"><%= LNAME.getErr
<% } %>
<br>

```

Figure 109. JSP source after modification

The appearance of the page has now been altered independent of the UIRecord definition (see Figure 110).

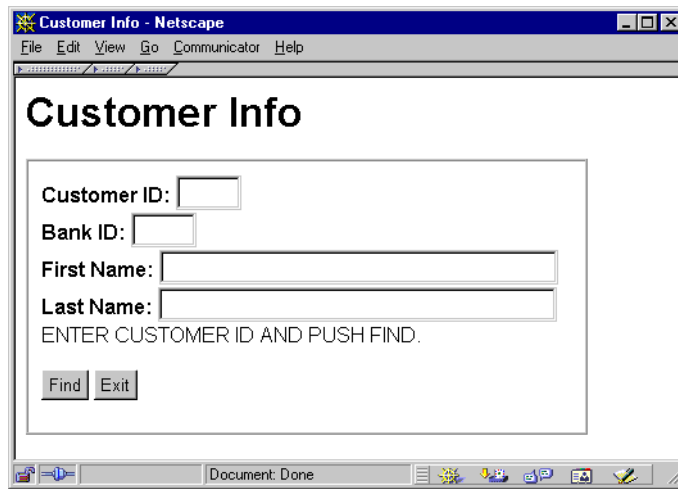


Figure 110. Modified JSP rendered in browser

Important Note

It is important at this point to fully comprehend the concept that these JSPs are completely and fully modifiable! There are many developers (especially those who have most of their experience in back-end development) who have been trained to never modify anything that VisualAge Generator produces or risk causing the system to fail. From a JSP perspective, this is untrue! As long as the changes you make to the JSP after it has been generated are purely cosmetic (HTML layout), are client-side JavaScript functionality, or are otherwise not involved in changing the **program code** that make the JSP work, you can edit completely non-destructively.

11.2 Level 2: Enter the JSP developer

The JSP developer is the second role that Level 2 development brings to the project. This is basically a support role for Phase 2 of a Web Transaction project. The Web Transaction developer is now able to pass along the Web site responsibilities to someone more knowledgeable. At this level, we can expect the JSP developer to have a firmer grasp on HTML knowledge and concepts than our Level 1 Web Transaction developer.

11.2.1 Advanced JSP customization

The JSP developer skill allows for both rapid and more advanced modification to the default JSP. A person filling this skill role should be able to simply go through the finished (generated default JSP) system and make the desired changes. These changes would be based on the level of HTML knowledge the JSP developer has, including modification of page layout, addition of simple menus, modification of tables, modification of forms, and color decisions.

If the JSP developer has the knowledge, Cascading Style Sheets (CSS), following the specification outlined by the World Wide Web Consortium at <http://www.w3c.org>, can be implemented in the default pages to make the task of modifying the pages a lot easier (this is demonstrated in 11.2.3, “Modification Using WebSphere Studio” on page 216).

Rather than spend time outlining how many changes a developer who is very experienced with HTML could make (since they are infinite and this is not an HTML instructional manual), a few possibilities and examples will be presented.

TABLE modification — Rows and Cells

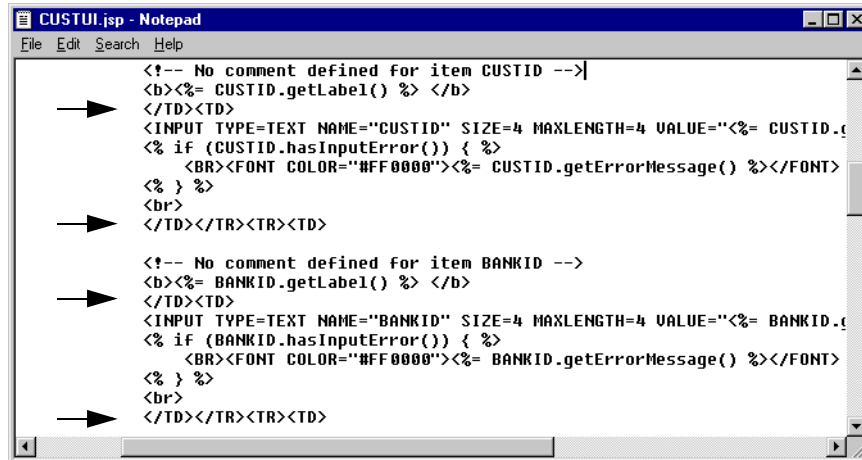
There are additional modifications we can make to our example TABLE to make it look much better. We can actually take advantage of the row and cell TABLE structure and make the elements of the FORM line up to provide a visually pleasing presentation. First, we need to break up the FORM elements into separate row and cell containers. This is done by inserting a `<TR>` wherever we need a new row, and a `<TD>` wherever we need a new cell. The number of columns *MUST* be consistent, so keep track of your numbers.

A TABLE that has a grid of rows and cells has the following layout:

```
<TABLE>
<TR>
<TD>Cell 1</TD>
<TD>Cell 2</TD>
</TR>
<TR>
<TD>Cell 3</TD>
<TD>Cell 4</TD>
</TR>
</TABLE>
```

In short, you need to rearrange the content to fit into these cells, either by cut and paste, or by inserting the `<TR>` and `<TD>` tags around your content.

An example of inserting the necessary tags is shown in Figure 111.



```
<!-- No comment defined for item CUSTID -->
<b><%= CUSTID.getLabel() %> </b>
</TD><TD>
<INPUT TYPE=TEXT NAME="CUSTID" SIZE=4 MAXLENGTH=4 VALUE="<%= CUSTID.g
<% if (CUSTID.hasInputError()) { %>
<BR><FONT COLOR="#FF0000"><%= CUSTID.getErrorMessage() %></FONT>
<% } %>
<br>
</TD></TR><TR><TD>

<!-- No comment defined for item BANKID -->
<b><%= BANKID.getLabel() %> </b>
</TD><TD>
<INPUT TYPE=TEXT NAME="BANKID" SIZE=4 MAXLENGTH=4 VALUE="<%= BANKID.g
<% if (BANKID.hasInputError()) { %>
<BR><FONT COLOR="#FF0000"><%= BANKID.getErrorMessage() %></FONT>
<% } %>
<br>
</TD></TR><TR><TD>
```

Figure 111. TABLE code with rows and cells

The revised visual view, with label and input field alignment, is shown in Figure 112.

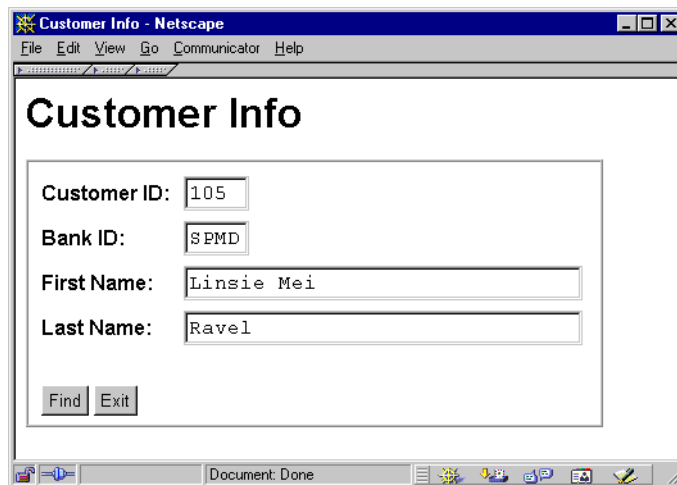


Figure 112. TABLE display after rows and cells

Note: If the generated default JSP file is modified with the simple insertion of `</TD><TD>` and `</TD></TR><TR><TD>` tags around the data item label and value (input field or output data), the JSP will function and look like the view shown in Figure 112.

This simple modification will result in a set of interleaved TABLE and FORM tags with the following layout:

```
<TABLE>
  <TR> <TD>
    <TABLE>
      <TR> <TD>
        <FORM>
          Cell 1</TD>
          <TD>Cell 2</TD>
        </TR> <TR>
          <TD>Cell 3</TD>
          <TD>Cell 4</TD>
        </FORM>
      </TD> </TR>
    </TABLE>
  </TD> </TR>
</TABLE>
```

The problem is that the interleaved syntax shown above will not be accepted by some tools, such as WebSphere Page Designer, whose parsing of the syntax is very strict.

The WebSphere Page Designer will issue a *Corrected the errors* message and modify the syntax by moving the </FORM> tag up in the source file to a point right after the first data item label. This will result in a nonfunctional form (all input fields are outside the form). To avoid this problem, reformat the tags with the following layout:

```
<TABLE>
  <TR> <TD>
    <FORM>
      <TABLE>
        <TR> <TD>
          Cell 1</TD>
          <TD>Cell 2</TD>
        </TR> <TR>
          <TD>Cell 3</TD>
          <TD>Cell 4</TD>
        </TD> </TR>
      </TABLE>
    </FORM>
  </TD> </TR>
</TABLE>
```

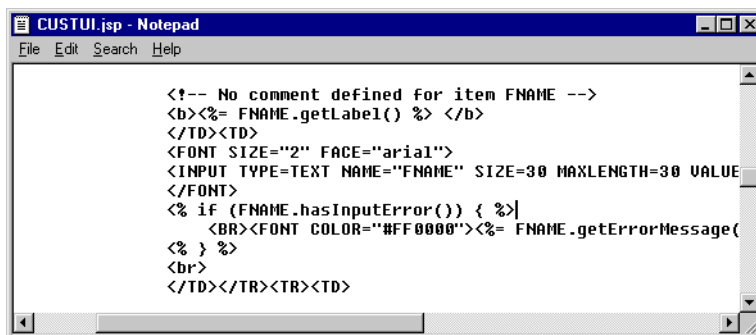

FORM modification—FONT: SIZE, FACE, and COLOR

There are more advanced changes at this level you can make to the appearance of the FORM itself. For example, if you've worked with HTML, you know that `` tags have a `SIZE` attribute and a `FACE` attribute associated with them.

You can actually resize and reformat the text fields by surrounding them with tags that have these attributes. If you examine the generated default JSP source, you see that there is no font information defined for both the text field labels and the text inside the input fields. The presentation will use the default font defined in the browser.

However, by modifying the JSP source, we can add tags that will change the visual appearance.

To change the appearance of a text field, simply add a `` tag before the `INPUT` element, including the appropriate `SIZE="2"` `FACE="arial"` attributes, and add the closing `` tag after the `INPUT` element. This change is shown in Figure 113.



```
CUSTUI.jsp - Notepad
File Edit Search Help

<!-- No comment defined for item FNAME -->
<b><%= FNAME.getLabel() %> </b>
</TD><TD>
<FONT SIZE="2" FACE="arial">
<INPUT TYPE=TEXT NAME="FNAME" SIZE=30 MAXLENGTH=30 VALUE
</FONT>
<% if (FNAME.hasInputError()) { %>
  <BR><FONT COLOR="#FF0000"><%= FNAME.getErrorMessage(
<% } %>
<br>
</TD></TR><TR><TD>
```

Figure 113. FONT attributes given to FORM element

Now, if we take a look at the presentation view of the modified JSP (see Figure 113), we can see that anything that is displayed in the First Name field has the FONT properties that were assigned using the font tag set. See Figure 114.

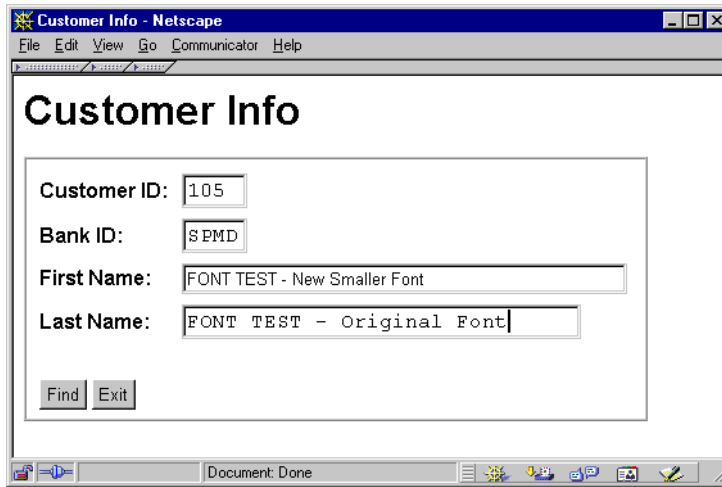


Figure 114. FORM field with modified FONT properties

It should also be noted that Generator provides us with color information for our error messages, as shown in Figure 115.

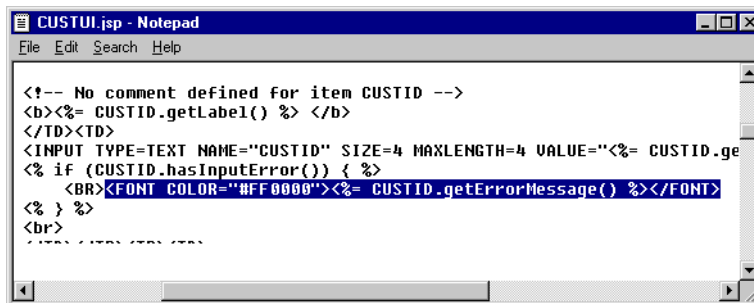


Figure 115. Color information for FORM error message

The color code is a hexadecimal number based on three pairs of numbers that stand for amounts of red, green, and blue color, respectively. HTML syntax also allows us to just use the word "red" as well.

11.2.2 WebSphere Studio

Some of the basic concepts that are key to understanding how WebSphere Studio can be used are introduced in this section.

Setup and configuration

Before we can use WebSphere Studio to enhance the formatting and presentation of our JSP pages, we must first install and configure WebSphere Studio (use V3.02).

Note: If you do not have an appropriate version of the Microsoft Internet Explorer installed, WebSphere Studio will complain during the installation. The WebSphere Page Designer uses Microsoft Internet Explorer components to support rendering of the preview page. The preview page will not be available if you do not have Microsoft Internet Explorer V4 or V5 installed.

Configuration requirements will depend on your planned use of WebSphere Studio. You can add links between WebSphere Studio and the VisualAge for Java WebSphere Test Environment as well as the WebSphere Application Server runtime environment (see Figure 116).

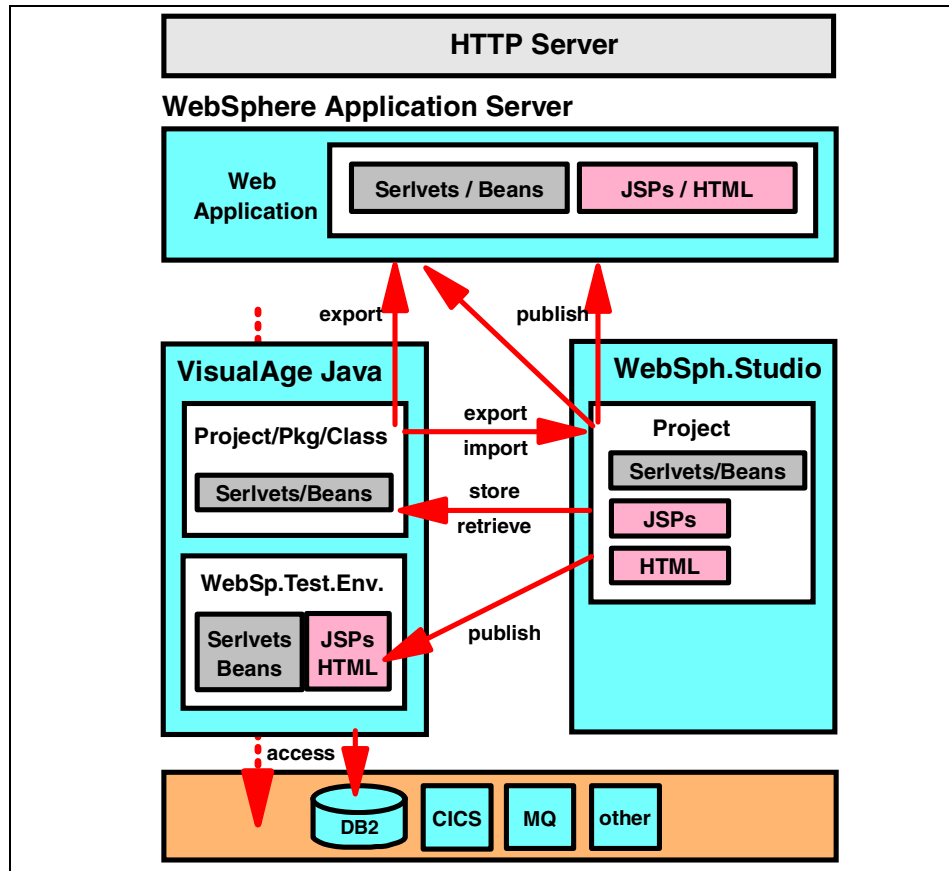


Figure 116. WebSphere Studio, VisualAge for Java, and WebSphere Application Server

Note: See *WebSphere Studio and VisualAge for Java — Servlet and JSP Programming*, SG24-5755 for additional details on the use of WebSphere Studio and VisualAge for Java in an integrated environment.

Publishing from WebSphere Studio

WebSphere Studio uses a *publishing* motif. The source in WebSphere Studio is published to a target server defined for a publishing stage. By default, two publishing stages (Test and Production) are preconfigured on your workstation when WebSphere Studio is installed (see Figure 117).

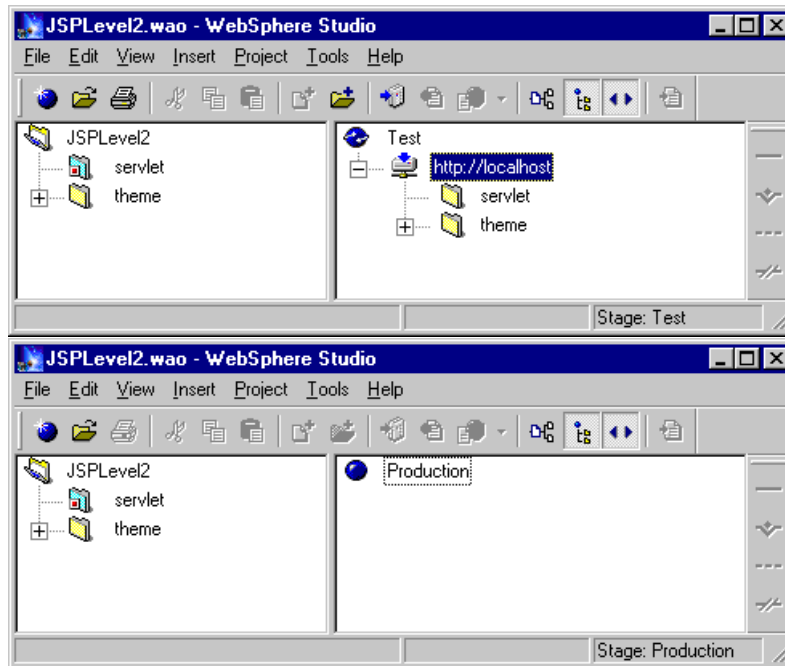


Figure 117. Default publishing stages and servers after WebSphere Studio installation

Depending on the software available on your workstation, one or more of these publishing stages may be precustomized with named targets for where the files go when you publish (see Figure 118).

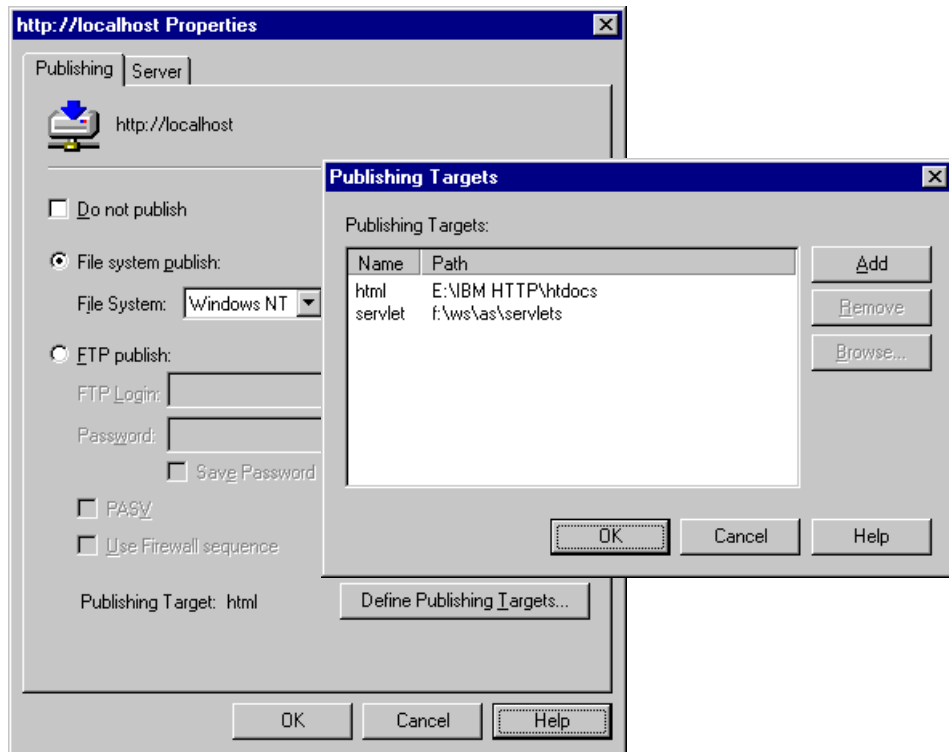


Figure 118. Default properties for target server

We have to define a publishing server so our files will be saved to our target directory. First, right-click on the server on the right side (should be either Test or Production, it does not matter) and choose **Insert > Server...**

Integration into Web Transaction development environment

To effectively use WebSphere Studio we must connect it to the development environment and define the flow of source and generated code between the tools.

We will focus on a simplified environment where we use WebSphere Studio to build HTML and customize the generated default JSPs, with support for publishing the front end components to the document root for either the WebSphere Test Environment or the WebSphere Application Server WebApp. Figure 119 shows development with WebSphere Studio, VisualAge for Java, and WebSphere Application Server.

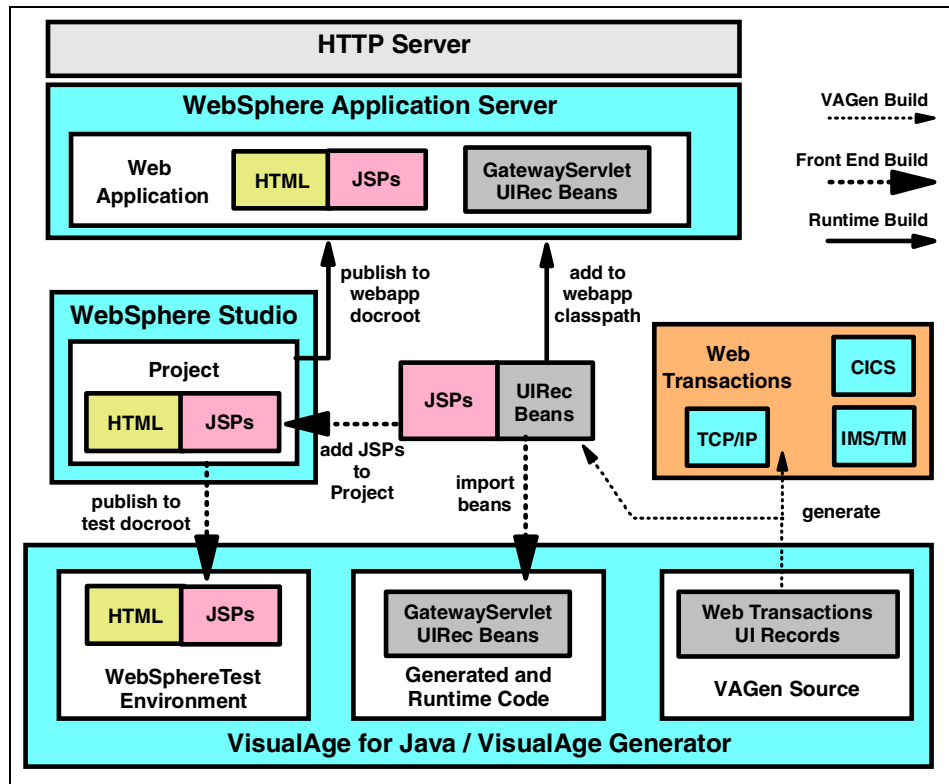


Figure 119. Development with WebSphere Studio, VisualAge for Java, and WebSphere Application Server

The basic process, as shown in Figure 119, is:

VisualAge Generator system build:

- Develop and test Web Transactions and UI Records
- Generate Web Transactions and UI Records

This is the act of developing the Web Transaction programs in VisualAge Generator and using the test facility to test the 4GL source in the browser. Once complete, or when the UI Record interface has stabilized, the Web Transactions and UI Records can be generated to produce the default generated JSP which can be further customized.

See Chapter 13, “VisualAge Generator Web Transaction runtime setup” on page 257 and Chapter 15, “Web Transaction generation” on page 335 for more information on setting up and building a VisualAge Generator Web Transaction system.

Front end system build:

- Import JavaBeans generated for UI Records into VisualAge for Java
- Add GatewayServlet to VisualAge for Java and configure in WebSphere Test Environment
- Add generated default JSPs to WebSphere Studio project
- Enhance generated default JSPs, adding HTML as required
- Publish front end code to docroot defined for WebSphere Test Environment
- Test front end using Web Transaction runtime system

This is the act of customizing the front end of the system. The task may require the development of an HTML-based Web site and the customization of the generated default JSPs.

WebSphere Studio can be used to support this activity. Once complete, or as part of an iterative development process, the front end system can be published to the WebSphere Test Environment to support dynamic testing of front end logic which directly calls the generated Web Transaction programs in the runtime environment.

See 14.4, “VisualAge for Java WebSphere test environment” on page 329 for more information on setting up the WebSphere Test Environment.

Note: We have not added the Gateway Servlet or the JavaBeans generated from the UI Record to the WebSphere Studio environment. There did not seem to be any value in doing so at this time. If the generated JSPs change, then there might be a time when adding these additional files to the WebSphere Studio project would make sense.

Runtime system build:

- Configure GatewayServlet in WebSphere Application Server
- Add generated JavaBeans to WebSphere Application Server Web application
- Publish front end code to docroot defined for WebSphere Application Server Web application

This is the act of implementing a full runtime system. The front end system is published to the Web application defined in the WebSphere Application Server. The generated JavaBeans are also added to the Web application, either as class files or packaged in a JAR file.

See Chapter 14, “WebSphere Application Server setup” on page 301 and Chapter 16, “Running Web Transactions” on page 341 for more information on setting up the Web Transaction runtime system.

11.2.3 Modification Using WebSphere Studio

We can use WebSphere Studio to enhance the formatting and presentation of our JSP pages. If you have reviewed the Level 1 approach, you are aware of some of the changes to the HTML code that can be made by customizing the generated JSPs. WebSphere Studio will support making the same kind of changes, but with more powerful tooling for extended modification capability.

In this section we will repeat the previous JSP customization exercises using WebSphere Studio and then extend all the JSPs in the current project using cascading style sheets (CSS).

Creating a WebSphere Studio project

After installing WebSphere Studio the first thing you need to do is create a new project using the new project dialog (see Figure 120). You can call the new project *JSPLevel2* or something similar. We will not use a template for this project.

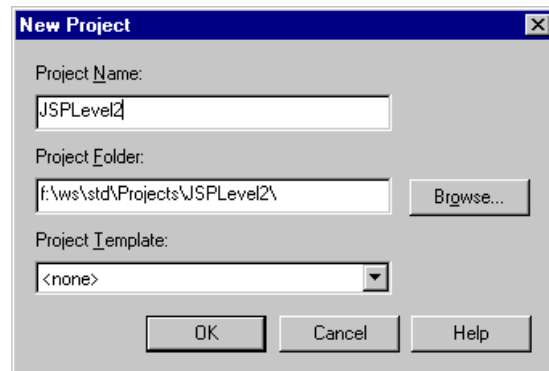


Figure 120. Creating the StyleDemo project in WebSphere Studio

Adding publishing support for WebSphere Test Environment

We have to define and/or customize a publishing stage so our files will be sent to the appropriate target directory. We want to publish the JSP files to the docroot defined when the WebSphere Test Environment was customized. For this we will customize the test publishing stage to send the files to the WebSphere Test Environment.

Note: During the WebSphere Studio installation, the test publishing stage may have already defined a server customized to any existing Web server and WebSphere Application Server installation on your workstation. If this is the case, you may wish to add a new publishing stage that you customize specifically for the WebSphere Test Environment.

To customize the server we need to set the publishing target to a value that matches the docroot defined in the WebSphere Test Environment SERunner.properties file:

```
#####
# SERunner.properties                                     #
#                                                         #
# VisualAge for Java WebSphere Test Environment Properties #
#                                                         #
# docRoot - location the server expects to find html, jsp and #
#           various other resources.                       #
#                                                         #
# httpPort - The port the server listens on for HTTP requests. #
#                                                         #
#                                                         #
# Note: form the path using either single forward slash "/" or #
#       or double backslashes "\\"                       #
#####

httpPort=8181
docRoot=e:\\Genout\\WSTEnv
serverRoot=g:\\j3\\ide\\project_resources\\IBM WebSphere Test Environment
```

To do this, select the server and customize the properties. Click on the Define Publishing Targets push button and adjust the HTML and servlet targets to identify the same docroot using the standard directory name format (no double backslashes; see Figure 121).

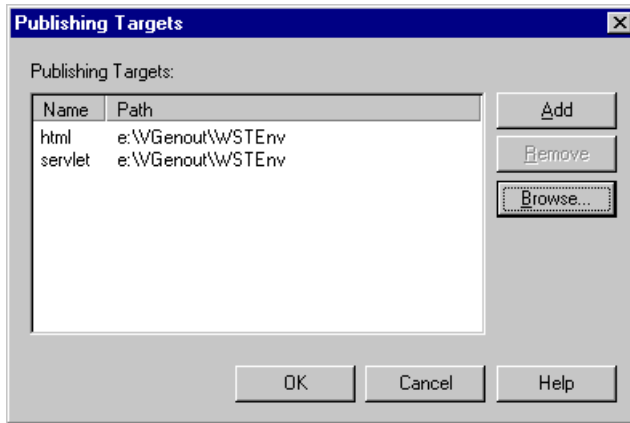


Figure 121. Publishing targets properties for project

Add VisualAge Generator JSPs to project

The JSPs used to support logon processing, the entry page, and error display, as provided by VisualAge Generator, need to be added to the base project to provide complete publishing support. You could also manually copy these files to the active docroot directory and customize them using Notepad.

All the JSPs used in a Web Transaction system must be in the same docroot directory. This is because the Gateway Servlet is programmed to look in the root directory (active docroot).

To add the VisualAge Generator JSPs to a folder for the project:

- Select the project icon in the left side of the WebSphere Studio workbench
- Select Insert -> File from the menu bar
- Find and select the following JSPs and GIFs from the directory where VisualAge Generator Server was installed (you can select them all at once):

```
CSOERRORUIR.jsp
Vagen1EntryPage.jsp
Vagen1ErrorPage.jsp
Vagen1LogonPage.jsp
vawcg-wp.gif
visage.gif
```

You can also add the JSP customized previously (CUSTUI.jsp, see 11.1, “Level 1: What’s a Web Transaction developer to do?” on page 195) to provide access to a Web Transaction.

Note: If you had previously copied and customized these files, you may want to add them to WebSphere Studio from your alternative location.

The files that have been added to the project get mirrored in the publishing server (see Figure 122).

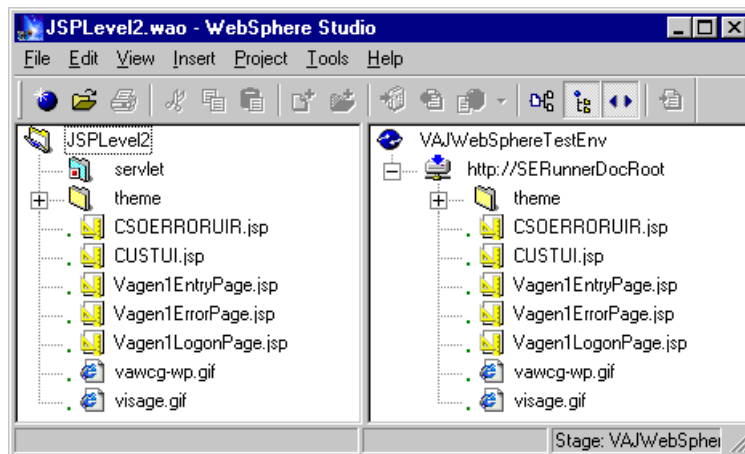


Figure 122. Publishing targets properties for VAGenFiles folder

- Select each JSP and modify the properties so that the *Use Parser* option is not selected.

If used, the parser will place an unexpected / in the ACTION value for the form definition as shown below:

```
<FORM METHOD=POST ACTION="/<%= hptGatewayURL %>">
```

The added / will generate the invalid destination for form interaction shown below:

Error 400

```
An error has occurred while processing
request:http://127.0.0.1:8181/http://127.0.0.1:8181/GatewayServlet
```

So be sure you change the properties before you first edit the JSP using WebSphere Studio. Otherwise you may have to manually correct the source or add the generated JSP to project again.

This completes the base setup of a Web Transaction front end in WebSphere Studio. To create a recovery point you may wish to export the current WebSphere Studio project to an archive file (use menu option File -> Save as Archive...).

You can now test the publishing process by selecting the JSPLevel2 project in the left pane of the WebSphere Studio workbench and choosing the menu option File -> Publish whole Project. The publishing process may ask you to create directories (if the targets do not exist) or overwrite files (if your target has old versions of the files in place).

A portion of the report generated after publishing the current project is shown in Figure 123.

The screenshot shows a report window titled "Files Published and Verified" with the text "Published to Server: http://SERunnerDocRoot". Below this is a table with two columns: "Destination on Server" and "Source".

Destination on Server	Source
http://SERunnerDocRoot/CSOERRORUIR.jsp	\CSOERRORUIR.jsp
http://SERunnerDocRoot/vawcg-wp.gif	\vawcg-wp.gif
http://SERunnerDocRoot/visage.gif	\visage.gif
http://SERunnerDocRoot/Vagen1ErrorPage.jsp	\Vagen1ErrorPage.jsp
http://SERunnerDocRoot/Vagen1LogonPage.jsp	\Vagen1LogonPage.jsp
http://SERunnerDocRoot/Vagen1EntryPage.jsp	\Vagen1EntryPage.jsp
http://SERunnerDocRoot/CUSTUI.jsp	\CUSTUI.jsp

Figure 123. Publishing targets properties for VAGenFiles folder

The directory view for the target docroot directory is shown in Figure 124.

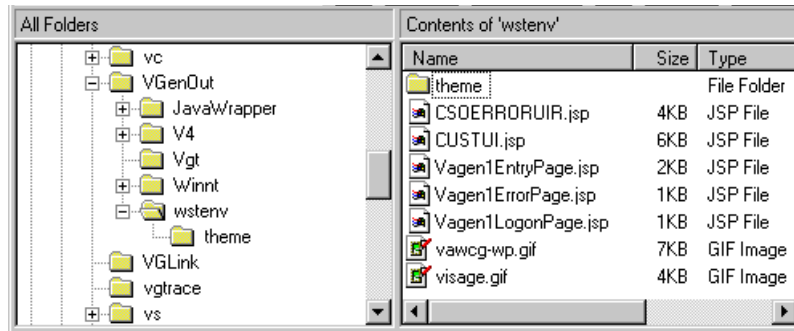


Figure 124. Publishing targets properties for VAGenFiles folder

At this point you can start the WebSphere Test Environment and get the initial VisualAge Generator JSP sent by the Gateway Servlet. If you have generated and set up the Web Transaction used by the CUSTUI.jsp, you can also test this program.

Testing the current setup will validate the WebSphere Studio to WebSphere Test Environment publishing process (for WebSphere Studio, VisualAge for Java, and WebSphere Application Server) which was originally outlined in Figure 116 on page 211.

Using folders in WebSphere Studio

We would like to organize files in folders using WebSphere Studio and then use customized publishing properties so that no actual directory structure is used during publishing for the folders defined in WebSphere Studio. However, this does not function as desired.

If we attempt to use a folder by following these steps:

- Select the project icon in the left side of the WebSphere Studio workbench
- Select Insert -> Folder from the menu bar
- Add a folder named VAGenFiles
- Select the VAGenFiles folder
- Select Insert -> File from the menu bar
- Find and select the required VisualAge Generator Server JSPs and GIFs

And then customize the publishing process by doing the following:

- Select the VAGenFiles folder in the publishing domain (right side of WebSphere Studio workbench view when the publishing view is active) and using mouse button 2, select the properties option.
- To ensure that the WebSphere Studio folder organization does not force a similar directory structure during publishing, modify the publishing page for the VAGenFiles folder as shown in Figure 125.

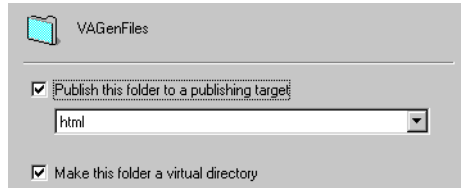


Figure 125. Publishing targets properties for VAGenFiles folder

Note: If you do not select the *Make this folder a virtual directory* option, the folder will not be created during publishing; but references, such as those to the GIF files, will include the VAGenFiles directory name.

The problem is that the mix of the required deselection of the *Use Parser* option for JSP file properties and the publishing target customization results in a JSP that looks like this:

```
<HEAD>
  <TITLE>VisualAge Generator System Entry</TITLE>
</HEAD>
<BODY background="/VAGenFiles/vawcg-wp.gif">
<img SRC="/VAGenFiles/visage.gif" >
<FONT SIZE=6>VisualAge Generator System Entry</FONT>
<img SRC="/VAGenFiles/visage.gif" >
<BR>
<BR>
<FORM METHOD=POST ACTION="<%= hptGatewayURL %>">
```

If you keep the referenced GIF files in the root directory, the VisualAge Generator supplied JSPs will resolve the name in the flat publishing directory.

The use of folders may allow for better JSP file management in a team environment when using WebSphere Studio. Just be sure that the WebSphere Studio organization results in a published structure where all references resolve.

Formatting using cascading style sheets

A CSS is simply a central location for style information to which you connect all the pages in your Web site. This means that if, for example, you want to change the font size of your pages, you could simply change the font definition in the style sheet one time rather than make the exact same changes to every instance of a font definition in every page.

Note: Not all versions of the browsers available support the use of CSS references in HTML/JSP files. We used both Netscape Navigator 4.5, 4.6, and 4.7 as well as Microsoft Internet Explorer 4.0 in our testing for this exercise. Many of the CSS defined options for control of formatting and presentation was only completely visible when using Microsoft Internet Explorer.

The base rendering of the default generated JSP for the simple Customer Info system (CUSTUI.jsp), for both Netscape Navigator and Microsoft Internet Explorer, is shown in Figure 126.

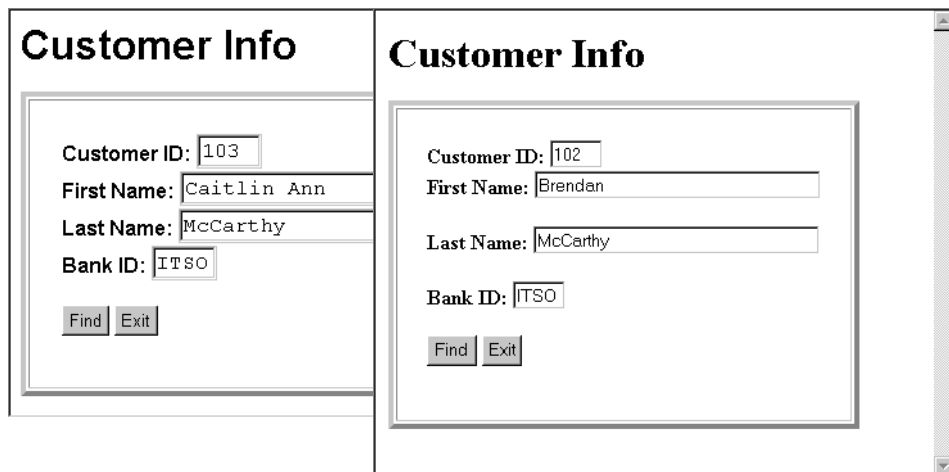


Figure 126. Base rendering of CUSTUI.jsp in Navigator (l) and Internet Explorer (r)

References to the Master.css file that exists in the active WebSphere Studio project are automatically included when you create JSPs using a WebSphere Studio Wizard (such as the JavaBean Wizard). An example of this reference is shown below:

```
<LINK href="file:///d:/ws/std/Projects/JSPLLevel2/theme/Master.css"
rel="stylesheet" type="text/css">
```

Using WebSphere Studio you can add the generated default JSPs to a project and apply a style sheet to them for more flexible control of their appearance.

To add support for the CSS definition in the project, a link relationship must be defined in the JSP file. This is done by adding this statement just before the `</HEAD>` tag in the JSP file:

```
<LINK REL="stylesheet" TYPE="text/css" HREF="theme/Master.css">
```

Note: To edit our `CUSTUI.jsp`, we can simply double-click on it to open it in Page Designer. If this does not happen automatically, go to the **Tools > Tools Registration...** menu and register **Page Designer** as the default editor for "html" extensions.

The revised rendering of the default generated JSP, when combined with the default `Master.css` file in the WebSphere Studio project, is shown in Figure 127. Slight changes in the fonts used for the title and input field labels are visible.

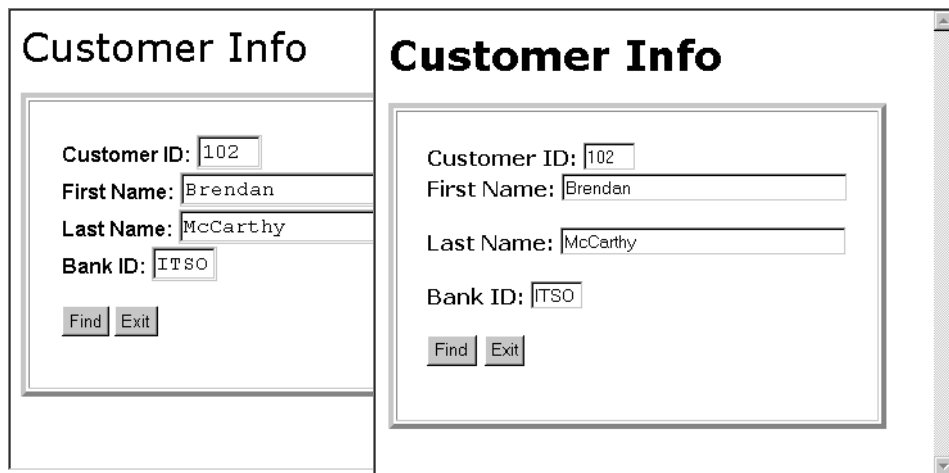


Figure 127. Rendering with default `Master.CSS` in Navigator (l) and Internet Explorer (r)

If we change the `Master.css` file we can further control the presentation used for the generated default JSP. To show this, we modify the fonts used in the `Master.css` using WebSphere Page Designer. The "before" and "after" settings are shown in Figure 128.

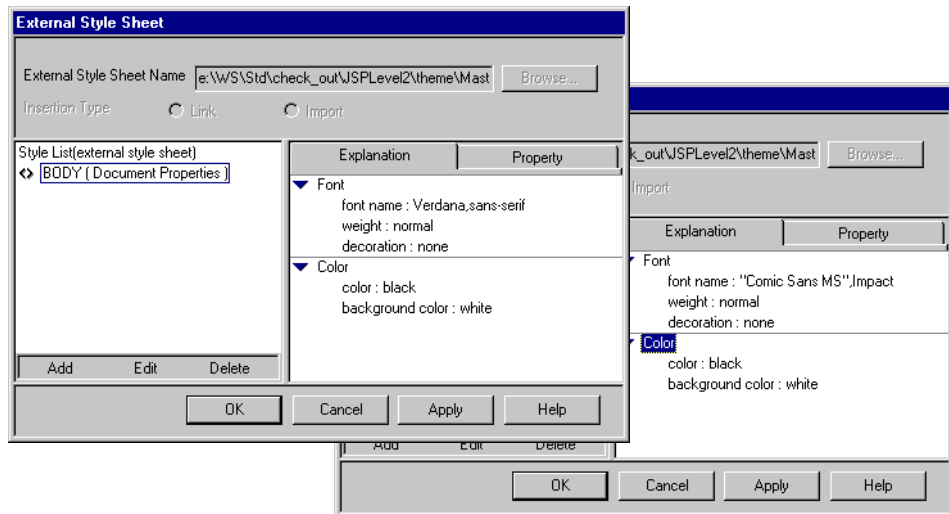


Figure 128. Master.css font modification

The revised rendering of the default generated JSP, when combined with initial font modification to the Master.css file in the WebSphere Studio project, is shown in Figure 129. Additional changes in the fonts used for the title and input field labels are visible.

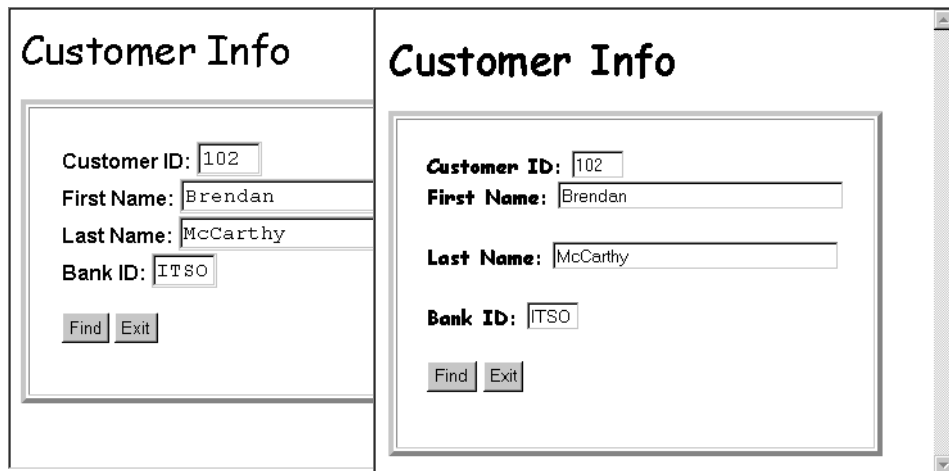


Figure 129. Rendering with modified fonts in Navigator (l) and Internet Explorer (r)

If we continue to change the Master.css file we can demonstrate more vivid control of the presentation used for the generated default JSP. This is shown by adding additional changes to the Master.css (see Figure 130).

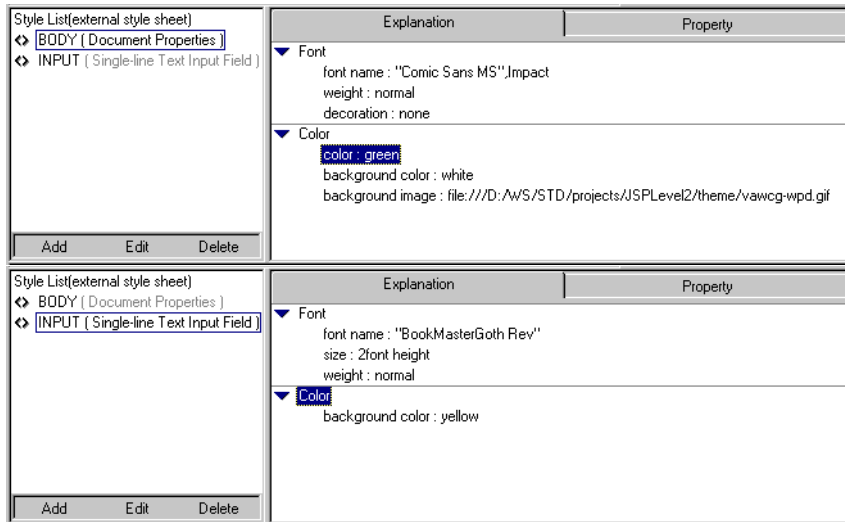


Figure 130. Master.css font modification

These modifications work with the default generated JSP and result in the presentation shown in Figure 131.

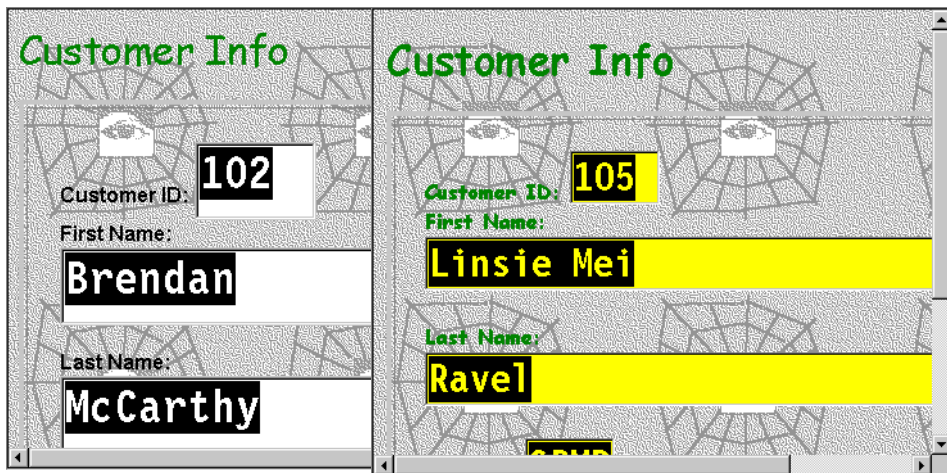


Figure 131. Rendering with modified fonts in Navigator (l) and Internet Explorer (r)

Master.css changes are reflected directly in the WebSphere Page Designer when editing a JSP that includes the reference (see Figure 132).

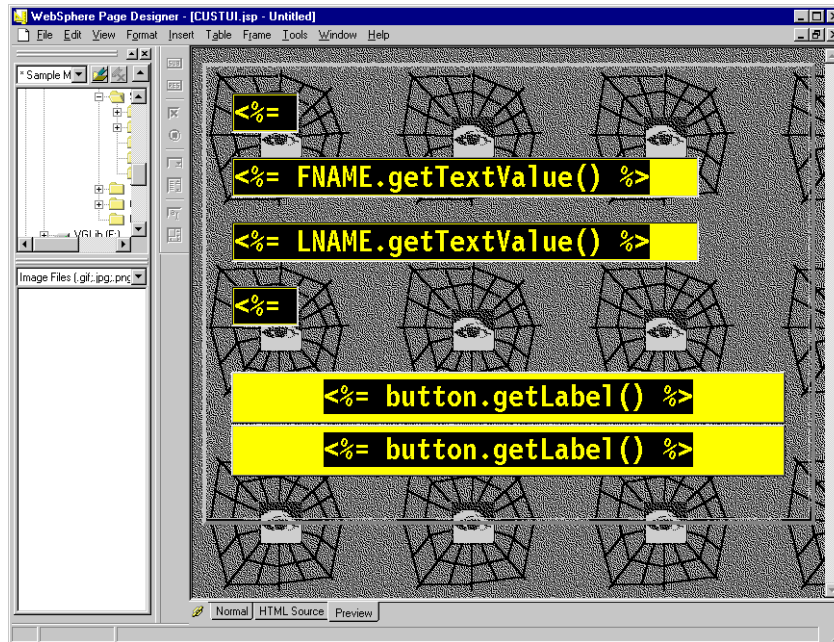


Figure 132. Editing CUSTUI.jsp in WebSphere Page Designer

11.3 Level 3: Integrating Web Transactions into a Web site

The third role that is added in the Level 3 development scenario is that of an HTML designer. As it has been mentioned before, the HTML designer will not do anything outside of the ordinary as far as what type of Web site development is being done, but this third role does significantly alter the project planning parameters and the responsibilities of the JSP developer.

In fact, it is the JSP developer that will be responsible for most of the work required to integrate Web Transactions into a completed Web Site.

11.3.1 Explanation of simultaneous development

The whole concept of simultaneous development is unique to Level 3 development. In Figure 40 on page 113, we can see that in the two-phase development process, the first phase contains tasks for both the Web Transaction developer and the HTML designer. The thing that makes this simultaneous development model possible is the fact that the development of presentation and functional components are separate processes, until Phase 2, when the JSP developer takes over.

Since they are two separate processes, it is not necessary (and not recommended) for the project to move in a linear fashion from Web Transaction development to JSP development to HTML design. Attempting to complete a project this way would significantly increase total project hours, elapsed time, and cause unnecessary overlap between the different roles.

11.3.2 Web site planning issues

There are a few issues that must be given a lot of thought during the planning stage of a Web Transaction project. This planning will be different than planning for a regular Web site deployment due to the nature of the default JSPs and JavaBeans generated by VisualAge Generator, and the behavior of Web browsers, HTTP, and HTML, and JSP/Servlet runtime environments such as WebSphere Application Server.

Web site complexity

First in importance is the complexity of the Web site design itself. It is common practice to create Web sites that are FRAMES-based or have FRAME elements in them. While not impossible to accommodate, these types of Web sites have numerous issues associated with them that will make the actual deployment of the system slightly more difficult.

Scope and context

The scope of a page in a Web site is determined not by its placement, but by its place within the hierarchy of FRAMES. In a client-side programming language (such as JavaScript), the top-level browser window is an object of type **window**. However, each FRAME inside that window is an entity unto itself, also of type **window**. So, you can have an infinite number of windows within this window, each of which has its own HTTP Request, source HTML, and name.

Use of session data

As described above, a page contains other pages in the form of FRAMES. However, the data beans live in the scope of an HTTP request unless you explicitly program your JSP to place the bean in session data. As you will see, this is required to support effective integration of a Web Transaction in a FRAMES-based Web site.

Other client-side technologies

For the most part, client-side technologies such as JavaScript can interact non-destructively with a Web Transaction system and its JSPs. You are permitted to add function to the generated default JSP, as required. This may increase the complexity of the development process when changes to the UI Record definition have to be matched in the customized JSPs.

11.3.3 Development steps

Web Transaction developer

The Web Transaction developer creates, tests, and generates a fully functional Web Transaction system.

HTML designer creates mock-up Web site

During Phase 1, while the Web Transaction developer is building and testing the Web Transaction system, the HTML designer creates the Web site in its entirety. The use of static text data in place of the dynamic data that might be provided by the completed JSP can help create a finished look to the mock-up site.

JSP developer integrates Web Transactions into Web site

Once the front and back end components are complete, the JSP developer can integrate the Web site with the Gateway Servlet and the runtime implementation of the Web Transaction programs.

11.3.4 Front end Web site development

Instead of developing a Web site from scratch we choose to use a simple FRAMES-based site created from a WebSphere Studio template.

WebSphere Studio and the default templates

WebSphere Studio is a part of the WebSphere suite of products. The WebSphere Studio set of tools allow developers to integrate HTML, JSPs, servlets, and other Web site elements into one environment called a *project*. In this project, pages may be modified, elements moved, and other elements added to create a final finished project which is then *published* to local or remote directories.

Included in WebSphere Studio is the ability to generate a Web site from a template when creating a new project. While you may decide not to use the provided templates for your Web site development, we found that the template gave us a reasonable implementation of a finished Web site that needed to be integrated with existing Web Transactions.

In WebSphere Studio version 3.02, there are 2 template choices: Corporate1 and Corporate2. Our example will use the Corporate1 template due to its simpler structure and ease of modification.

To begin, open WebSphere Studio. If you have not used the product before, or if you do not have an active project, a dialog box will ask you if you want to create a new project or open an existing project. Select the *Create a new project* option (see Figure 133).



Figure 133. Opening WebSphere Studio and creating a new project

If you are taken directly into WebSphere Studio on an active project, use the **File -> New Project...** menu option to start the **New Project** dialog.

In the **New Project** dialog provide a name for the new project (the directory structure is filled in automatically for you as you type the project name) and choose a Project Template of **Corporate1** (see Figure 134).

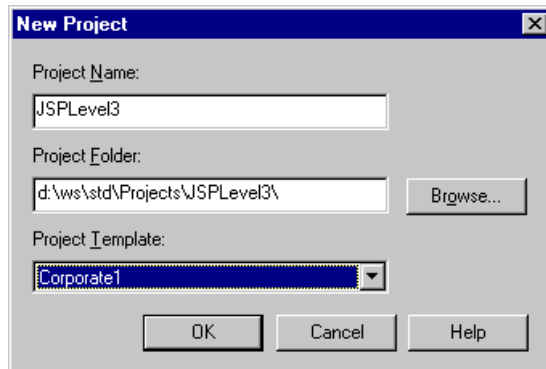


Figure 134. Selecting the Corporate1 template

When you click on **OK** WebSphere Studio will create a new project from the template.

Set up publishing support

The next task was to repeat the steps described earlier (see “Adding publishing support for WebSphere Test Environment” on page 216) to customize the publishing environment.

Figure 135 shows the WebSphere Studio workbench view after creating the project from the template and customizing publishing to support the WebSphere Test Environment. Different parts of the Web site are shown in each side of the WebSphere Studio workbench view.

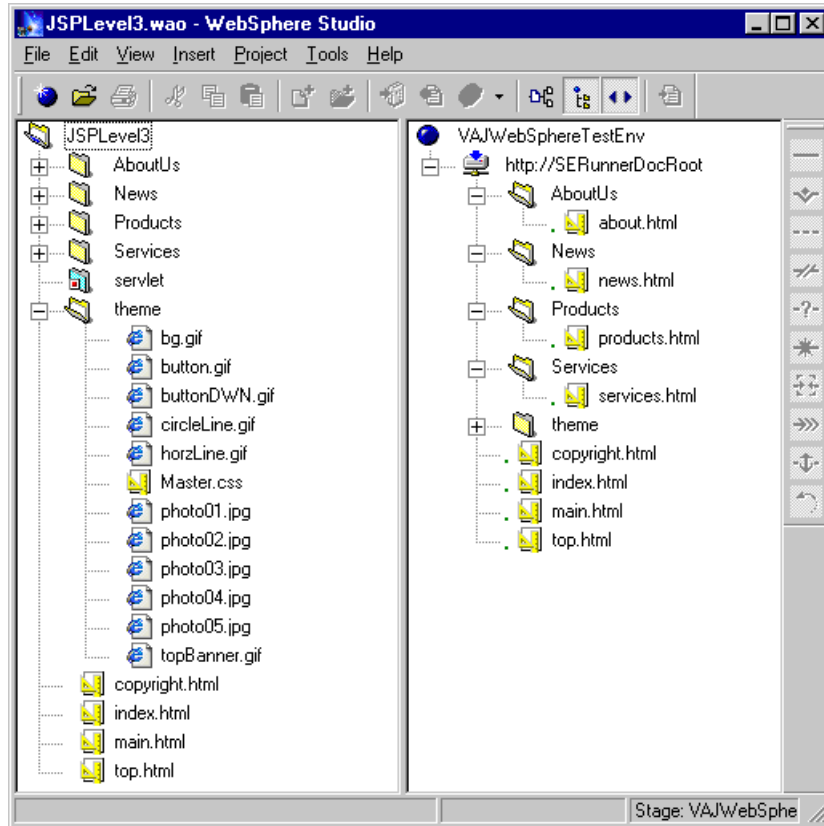


Figure 135. Web site from Corporate1 template in WebSphere Studio

You can test the Web site by publishing the project and using the WebSphere Test Environment to supply the HTML to a browser. The Web site entry page is shown in Figure 136.



Figure 136. Entry page for Corporate1 template Web site

Modify Web site: Pretty it up

The Web site created from the template does not format well in all browsers, so the first task is to change a few settings to improve the rendering.

The page we will be modifying most heavily is the page called **top.html**. Double-click on this page to open it up in WebSphere Page Designer. The initial view is the WYSIWYG view for the WebSphere Page Designer editor (see Figure 137).

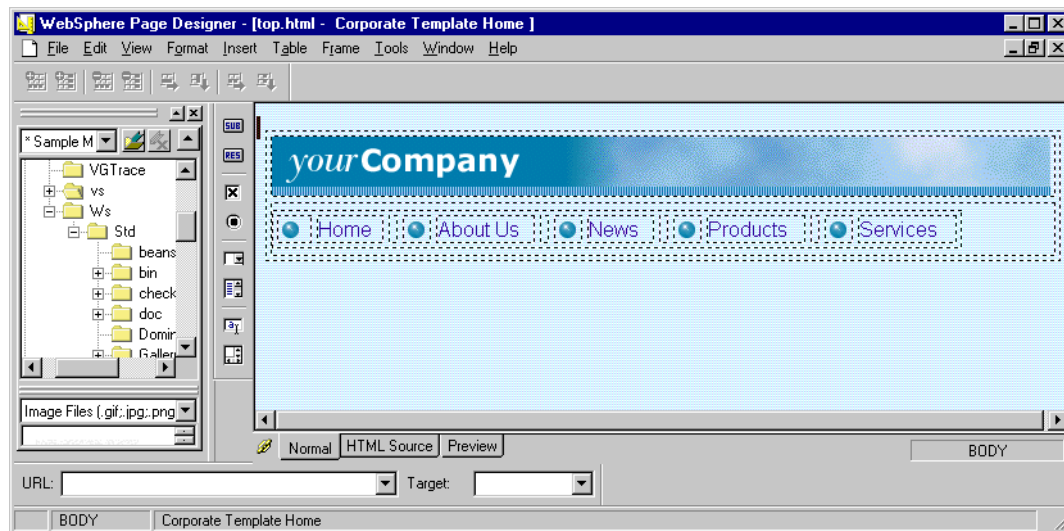
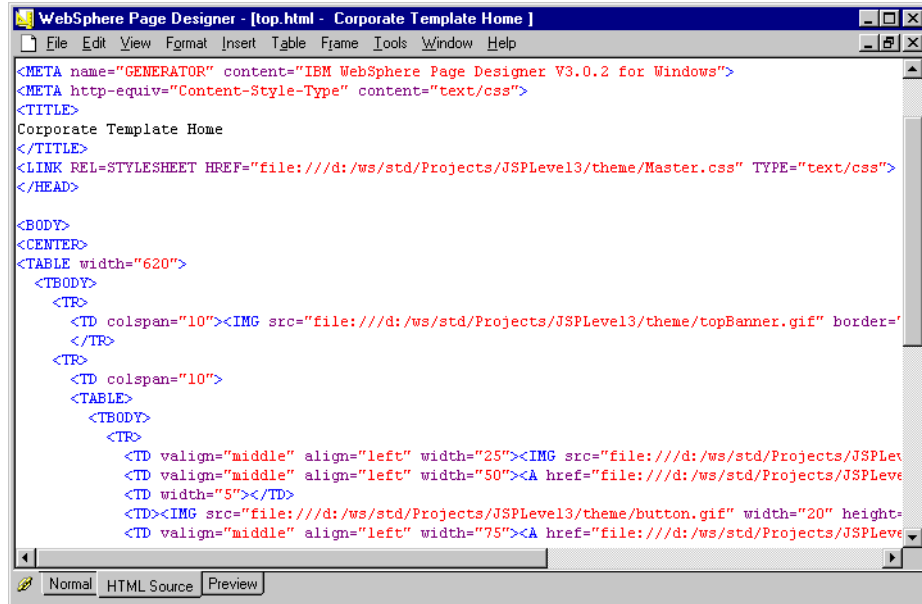


Figure 137. WebSphere Page Designer WYSIWYG view for top.html

Note: If this does not automatically happen, go to **Tools > Tools Registration...** to register **Page Designer** as the default editor for files with "html" extensions).

If we click on the *HTML Source* tab, the HTML tags will appear, each color-coded to identify with its function in the layout (see Figure 138).



```
WebSphere Page Designer - [top.html - Corporate Template Home ]
File Edit View Format Insert Table Frame Tools Window Help
<META name="GENERATOR" content="IBM WebSphere Page Designer V3.0.2 for Windows">
<META http-equiv="Content-Style-Type" content="text/css">
<TITLE>
Corporate Template Home
</TITLE>
<LINK REL=STYLESHEET HREF="file:///d:/ws/std/Projects/JSPLevel3/theme/Master.css" TYPE="text/css">
</HEAD>
<BODY>
<CENTER>
<TABLE width="620">
  <TBODY>
    <TR>
      <TD colspan="10"><IMG src="file:///d:/ws/std/Projects/JSPLevel3/theme/topBanner.gif" border="1">
    </TR>
    <TR>
      <TD colspan="10">
        <TABLE>
          <TBODY>
            <TR>
              <TD valign="middle" align="left" width="25"><IMG src="file:///d:/ws/std/Projects/JSPLevel3/theme/button.gif" width="20" height="20">
              <TD valign="middle" align="left" width="50"><A href="file:///d:/ws/std/Projects/JSPLevel3/theme/button.gif" width="50">
            </TR>
            <TR>
              <TD colspan="10"><IMG src="file:///d:/ws/std/Projects/JSPLevel3/theme/button.gif" width="20" height="20">
            </TR>
          </TBODY>
        </TABLE>
      </TD>
    </TR>
  </TBODY>
</TABLE>
Normal HTML Source Preview
```

Figure 138. WebSphere Page Designer HTML view for top.html

The first thing we want to do is go through and remove all hard-coded WIDTH attributes in the TABLE that holds the menu, but leave the TD tags with WIDTH="5" attributes alone; they are needed for the spacing.

Simply highlight all the WIDTH attributes in the TABLE and TD tags and delete them. DO NOT delete the tags, just the WIDTH attributes.

After you have finished doing this to every instance of the menu TABLE, save the file and close Page Designer.

If your menu items are not fully visible or are being cut off edit **index.html** with WebSphere Page Designer and click on the **Frame HTML Source** tab to see the FRAME source code (see Figure 139).


```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<META name="GENERATOR" content="IBM WebSphere Page Designer V3.0.2 for Windows">
<META http-equiv="Content-Style-Type" content="text/css">
<TITLE>ITSO Web Tran Demo System</TITLE>
</HEAD>
<FRAMESET rows="80,*" framespacing="0" frameborder="no" border="0">
<FRAME src="file:///d:/ws/std/Projects/JSPLevel3/top.html" name="upper" marginheight="2" marginwidth=
<FRAME src="file:///d:/ws/std/Projects/JSPLevel3/main.html" name="thecontent" scrolling="AUTO" frameb
<FRAME src="file:///d:/ws/std/Projects/JSPLevel3/copyright.html" name="lower" scrolling="NO" framebor
</FRAMESET>
<BODY>
<NOFRAMES>
<BODY>You must use a browser that supports frames
to view this site.</BODY>
</NOFRAMES>
</BODY>
</HTML>
```

Figure 139. WebSphere Page Designer Frame HTML source view for index.html

Modify the first value in the "rows" attribute of the FRAMESET to a larger number (such as 120).

We continued to make other cosmetic and content changes as we learned more about WebSphere Studio.

To help you master WebSphere Studio you may wish to order *WebSphere Studio and VisualAge for Java—Servlet and JSP Programming*, SG24-5755.

If we check-in all the modified files and republish the Web site, we will see the modified HTML site with an improved rendering for the top.html menu (see Figure 140).

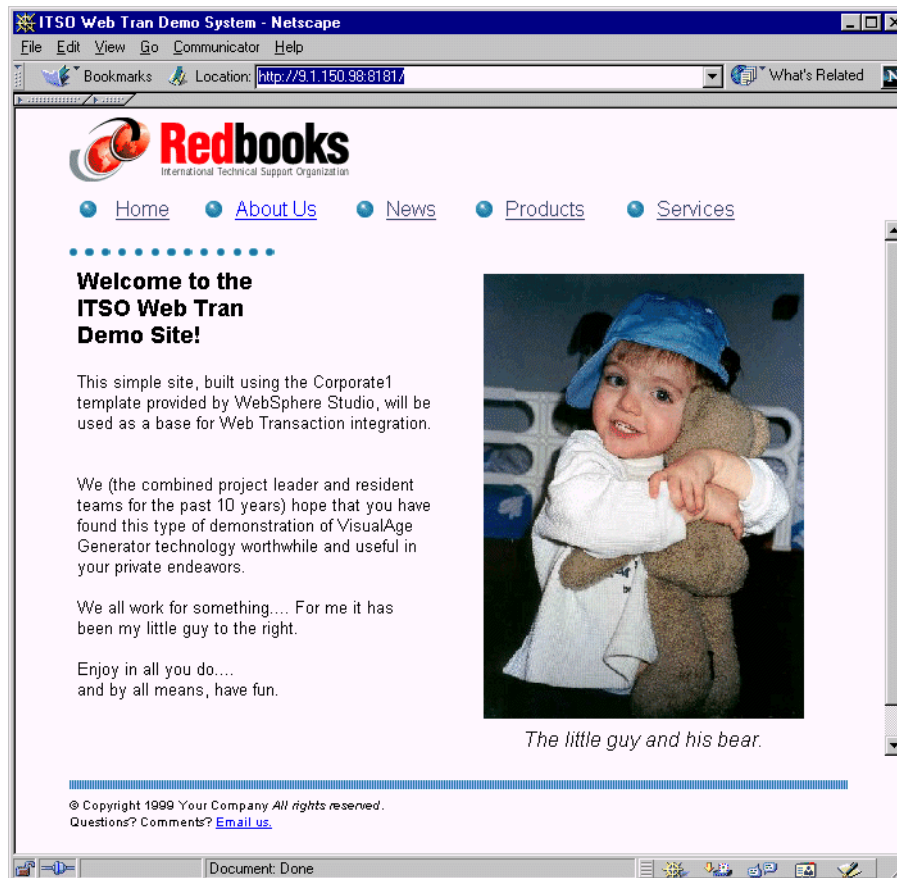


Figure 140. Enhanced entry page for Corporate1 template Web site

11.3.5 Bringing the two sides (front and back) together

The process of integrating Web Transaction programs into an existing FRAMES-based Web site is reviewed in this section.

There are five basic connection approaches to consider:

- Connecting Web site to Gateway Servlet
- Connecting Web site to specific Web Transactions
- Connecting Gateway Servlet to Web site
- Connecting Gateway Servlet to JSP-based Web site
- Adding dynamic Gateway Servlet name resolution to JSP-based Web site

Each approach allows for different levels of integration and support for dynamic references to the Gateway Servlet home (URL/WebApp).

Add VisualAge Generator components

We begin by adding the demonstration pages that we want to put in the context of this pre-designed Web site (remember, we are working with an HTML designer that provided the raw site while Web Transaction integration is being done by the JSP developer). This will allow us to publish the complete site from WebSphere Studio.

The Web Transactions that will be integrated into this Web site include components from the Customer Info Web Transaction (see Chapter 8, “Developing Web Transaction programming skills” on page 127) and the Web Transaction structure demonstration (see Chapter 10, “Demonstration system” on page 181).

We will also add the VisualAge Generator supplied JSPs and GIFs to support alternate entry point and logon processing options. The process for the VisualAge Generator parts was outlined earlier (see “Add VisualAge Generator JSPs to project” on page 217).

To insert the JSPs for the sample Web Transaction programs, select the project folder in the file view, choose the **Insert > File...** menu option and select the **Use Existing** notebook page. We can select multiple files in the **Open** dialog box by holding the **Ctrl** key while selecting the files.

To support the sample Web Transaction programs we added these files:

```
FRST_PLK_UI_RECORD.jsp
FRST_FRM_RECV_UI.jsp
FRST_FRM_UI_RECORD.jsp
FRST_PGM_UI_RECORD.jsp
CONV_UI_RECORD.jsp
CUSTUI.jsp
```

Remember to adjust the JSP properties (deselect *Use Parser* option) for each JSP file provided by or generated by VisualAge Generator.

Connecting Web site to Gateway Servlet

The HTML in the Web site can include a menu option that will start the Gateway Servlet which will then provide access to the Web Transactions that need to be available.

The HTML can be connected to the Gateway Servlet by adjusting the HREF link in a menu option found in the **top.html** code. The revised menu option will point directly to the Gateway Servlet using a hardcoded value that represents the name defined for the Gateway Servlet in the WebSphere Test

Environment or WebSphere Application Server. The HTML source for this connection is shown in Figure 141.

```
<TD valign="middle" align="left" width="20"><IMG src="/theme/button.gif" width="20"
height="20" border="0"></TD>
<TD valign="middle" align="left" width="20"><A href="/GatewayServlet"
target="thecontent">Gateway Servlet</A></TD>
```

Figure 141. Source for HTML navigation to Gateway Servlet

This pointer allows the HTML Web site to invoke the Gateway Servlet entry point. See the status area value ([http://127.0.0.1:8181/Gateway Servlet](http://127.0.0.1:8181/GatewayServlet)) for the link named Gateway Servlet in Figure 142.



Figure 142. Runtime view of HTML navigation to Gateway Servlet

When selected, this link will invoke the Gateway Servlet which will use the active parameters to determine what page to show next. Several parameter settings control the process:

- hptLogonPage
- hptEntryPage
- hptEntryApp.

In reality the process works like this:

```
if hptLogonPage ^= ''
    display hptLogonPage
end
if hptEntryPage ^= ''
    display hptLogonPage
else
    if hptEntryApp ^= ''
        start hptEntryApp Web Transaction
    end
end
end
```

For now we will ignore the use of the VisualAge Generator Web Transaction logon (hptLogonPage).

Figure 141 shows what happens with an active Gateway Servlet setting of hptEntryPage=Vagen1EntryPage.jsp and the navigation pointer shown in Figure 143 where the response is targeted to the frame *thecontent*.

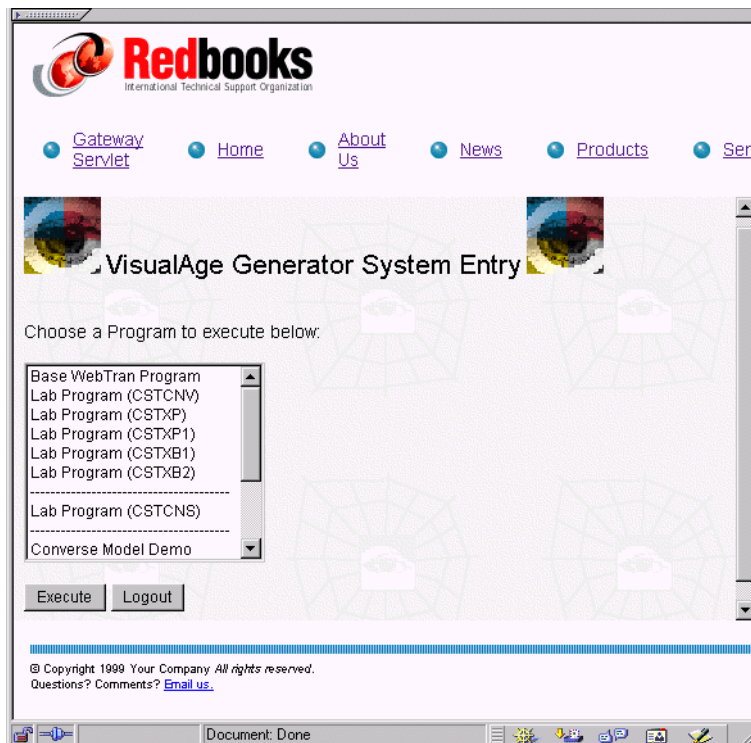


Figure 143. Gateway Servlet as target of HTML navigation

Web Transactions selected from the Vagen1EntryPage run in the content frame (see Figure 144).

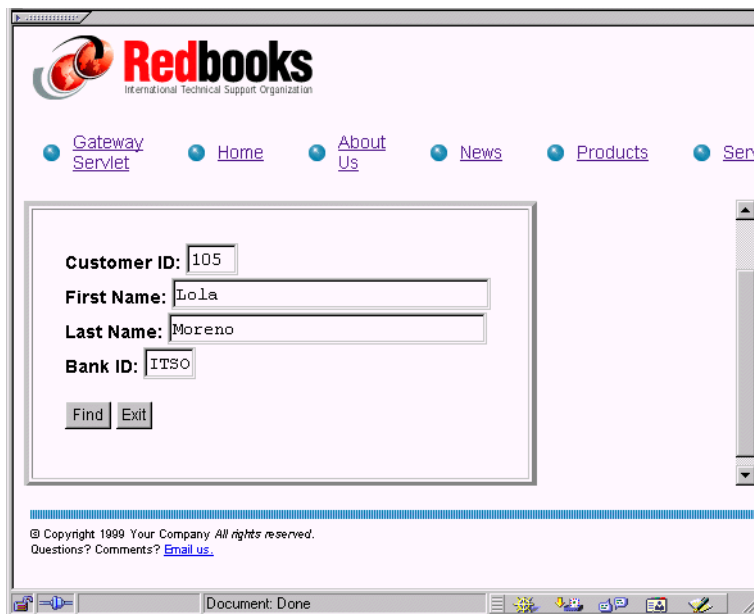


Figure 144. Customer Info Web Transaction running in HTML content frame

When you exit the Web Transaction, control returns to the Gateway Servlet entry point.

The problem with this approach (HTML to Gateway Servlet) is that only one HTML navigation option can be used to start a Web Transaction. If the UI Records defined for the Web Transaction system include appropriate navigation control, this may be sufficient. You should also be cautious of hardcoding the Gateway Servlet name.

Connecting Web site to specific Web Transactions

Different menu options in the HTML for the Web site can start the Gateway Servlet with parameters for a specific Web Transaction.

The adjusted HREF links in the **top.html** code that support distinct Web Transactions are shown in Figure 145.

Note: hptExec=Y for runtime, in the VisualAge Generator test facility you see hptExec=1.

```

<A href="/GatewayServlet?hptAppId=CSTCNV&hptExec=Y"
target="thecontent">Customer Info</A></TD>
<A href="/GatewayServlet?hptAppId=CONVMOD&hptExec=Y"
target="thecontent">Converse
<A href="/GatewayServlet?hptAppId=FRSTPGM&hptExec=Y"
target="thecontent">XFER PGM Model</A></TD>
<A href="/GatewayServlet?hptAppId=FRSTFRM&hptExec=Y"
target="thecontent">XFER ' ' Model</A></TD>

```

Figure 145. Source for HTML navigation to defined Web Transactions

These pointers allow the HTML Web site to use the Gateway Servlet to invoke specific Web Transactions. See the status area value for the link that started the Customer Info Web Transaction in Figure 146.

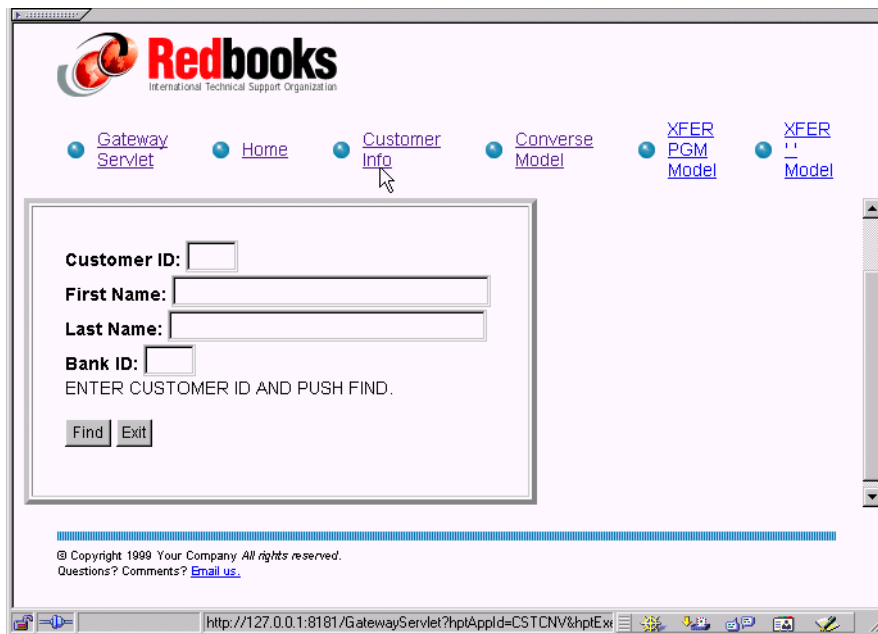


Figure 146. Direct invocation of Customer Info Web Transaction in content frame

When you exit the Web Transaction, control returns to the Gateway Servlet entry point, which will return the configured entry page.

The problem with this approach (HTML to specific Web Transaction) is that the termination of a Web Transaction may result in an unacceptable response (entry page list). You should also be cautious of hardcoding the Gateway Servlet name.

Connecting Gateway Servlet to Web site

The previous options had the HTML invoke the Gateway Servlet. With this approach we use the Gateway Servlet configuration to target the Web site. Instead of having the Gateway Servlet start the Vagen1EntryPage.jsp, we configure the Web site as the Gateway Servlet entry point.

To try this approach we will configure the Gateway Servlet to target the Web site entry point, the **index.html** file, as the entry page (hptEntryPage=index.html).

If there is a login page that has been configured for the Gateway Servlet (hptLogonPage=Vagen1LogonPage.jsp), or when a Web Transaction terminates, control is passed from the last JSP referenced by the Gateway Servlet servlet and the configured index.html entry point, which results in a failed response (see Figure 147).

Note: If a login page is not configured, the Gateway Servlet to Web site connection will function initially. Failure occurs when the Web Transaction terminates and control is passed back to the entry page.

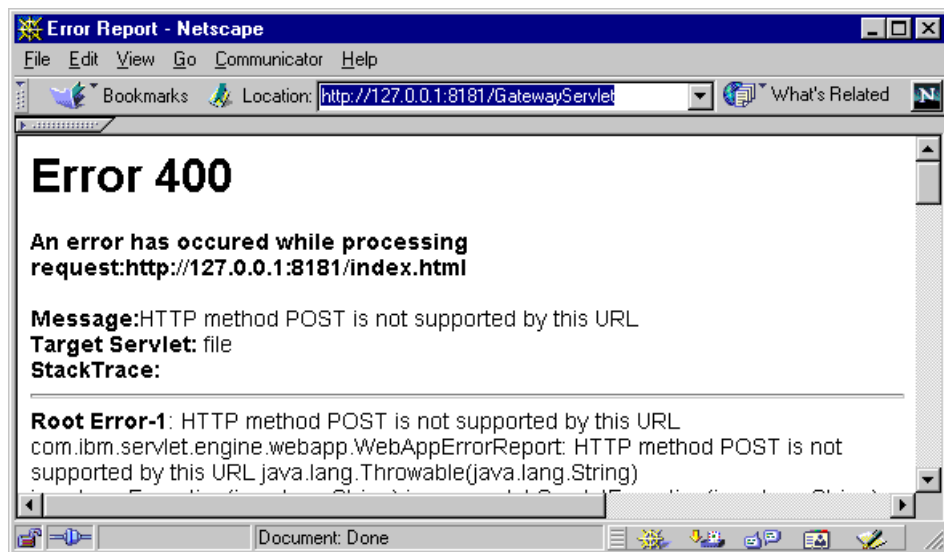


Figure 147. Response failure when using index.html as Gateway Servlet entry page

This failure is because the Gateway Servlet is programmed to work with JSPs and we are attempting to insert HTML files in place of the expected JSPs. The JSPs act like servlets and therefore can support the POST method.

Connecting Gateway Servlet to JSP-based Web site

The previous options had the Gateway Servlet target the HTML for the Web site, but this did not function in all situations. Instead of having the Gateway Servlet start the HTML, we will rework the Web site into a JSP that can be integrated with the Gateway Servlet.

The simplest approach to this requires the following steps:

- Rename index.html to index.jsp
- Configure Gateway Servlet with hptEntryPage=index.jsp

Yes, that is all that it takes!

This works fine at first, but when you exit a Web Transaction it returns to the Gateway Servlet entry point, which loads the index.jsp in the content frame. This results in a recursive Web site (see Figure 148).



Figure 148. Recursive failure when using index.jsp as Gateway Servlet entry page

This can be corrected by teaching the index.jsp file to navigate the frame set. Adding the JavaScript code shown in Figure 149 to the index.jsp will correct the recursive loading.

```
<HEAD>
<META name="GENERATOR" content="IBM WebSphere Page Designer V3.0.2 for
Windows">
<META http-equiv="Content-Style-Type" content="text/css">

<!-- Return from a WebTran will send entry point,
      this code makes sure that the site does not go recursive.
      That is, a site inside the content frame for the site. -->
<SCRIPT language="javascript">
    if (location != top.location)
    { top.location = location; }
</SCRIPT>

<TITLE>ITSO Web Tran Demo System</TITLE>
</HEAD>
```

Figure 149. Correcting the recursive loading for index.jsp as Gateway Servlet entry page

When you exit the Web Transaction, control returns to the index.jsp, which will then reset the Web site to the top level of the frame set.

The one possible problem with this approach (Gateway Servlet to JSP-based Web site) is that the Gateway Servlet name is hardcoded in the top.html file.

Adding dynamic Gateway Servlet name resolution

By adding some additional code to the JSP-based Web site, we can support multiple WebSphere Test Environment and/or WebSphere Application Server Gateway Servlet configurations without changing the Web site source. The name of the Gateway Servlet is defined in the WebSphere Test Environment or WebSphere Application Server configuration. Dynamic resolution allows for different names to be used without having to update the Web site.

The approach we will use mimics the dynamic Gateway Servlet name resolution found in the Gateway Servlet JSPs provided by VisualAge Generator (see Figure 150).

```

:
:
<%@ page errorPage="Vagen1ErrorPage.jsp" %>
<jsp:useBean id="hptGatewayURL" class="java.lang.String" scope="request" />
:
:
<FORM METHOD=POST ACTION="<%= hptGatewayURL %>">
:
:

```

Figure 150. Dynamic Gateway Servlet resolution in Vagen1EntryPage.jsp

The `jsp:useBean` entry references the object sent in the HTTP response produced by the Gateway Servlet. The action defined for the form uses the content of the bean to dynamically identify the address for the Gateway Servlet.

To use this logic in the JSP-based Web site we must make several changes:

- Rename `top.html` to `top.jsp`
- Rework `index.jsp` to:
 - Identify the `top.jsp` as the source for the navigation frame
 - Capture the `hptGatewayURL` object from the HTTP request and store it in session data
- Rework `top.jsp` to:
 - Obtain `hptGatewayURL` object from session data
 - Use `hptGatewayURL` object value for dynamic Gateway Servlet name in Web Transaction navigation

The key areas of the revised `index.jsp` are shown in Figure 151.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML> <HEAD>
<!-- web site entry point grabs URL bean from request -->
<%@ page errorPage="Vagen1ErrorPage.jsp" %>
<jsp:useBean id="hptGatewayURL" class="java.lang.String" scope="request" />
<!-- URL Bean is then saved again, in the active session,
so it can be found by other frame requests -->
<% session = request.getSession(true);
session.putValue("hptGatewayURL", hptGatewayURL); %>

<!-- If you wanted a new session you would do this -->
<!-- HttpSession mySession = request.getSession(true);
MySession.putValue("hptGatewayURL", hptGatewayURL); -->

<SCRIPT language="javascript">
if (location != top.location)
{ top.location = location; }
</SCRIPT>
<TITLE>ITSO Web Tran Demo System</TITLE>
</HEAD>

```

Figure 151. Revised index.jsp for dynamic Gateway Servlet resolution

The key areas of the revised top.jsp are shown in Figure 152.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML> <HEAD>
<%@ page errorPage="Vagen1ErrorPage.jsp" %>
<jsp:useBean id="hptGatewayURL" class="java.lang.String" scope="session" />
<TITLE>Corporate Template Home</TITLE>
</HEAD>

```

Figure 152. Revised top.jsp for dynamic Gateway Servlet resolution

These changes are required because we need the hptGatewayURL object in the top.jsp, but because the Gateway Servlet sends it as a request object, the index.jsp *consumes* the object and it is not available in the subsequent request sent to satisfy the navigation frame (top.jsp).

The recursive fix is still required in this implementation option.

The source for the final WebSphere Studio project discussed in this chapter is available; see Appendix A, "Sample code and other materials" on page 365 for details.

Part 4. Environment configuration and system implementation

Chapter 12. Runtime environment scenario implementation

During the residency we implemented IBM WebSphere Application Server on Windows NT with VisualAge Generator Web Transactions on three different runtime platforms: Windows NT, CICS on Windows NT, and CICS/ESA.

12.1 Windows NT Web Transactions

Our runtime implementation is shown in Figure 153.

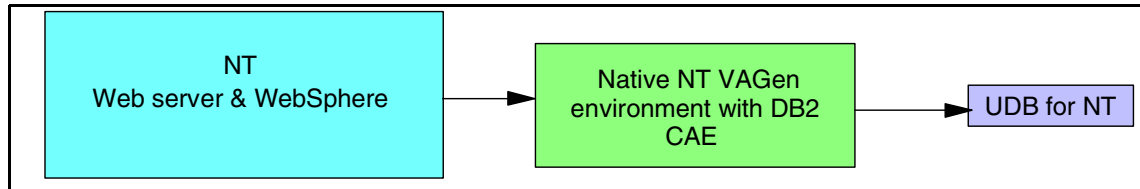


Figure 153. Native NT and UDB scenario

12.1.1 Software requirements

The software components used for the Windows NT-based runtime configuration are shown in Figure 154.

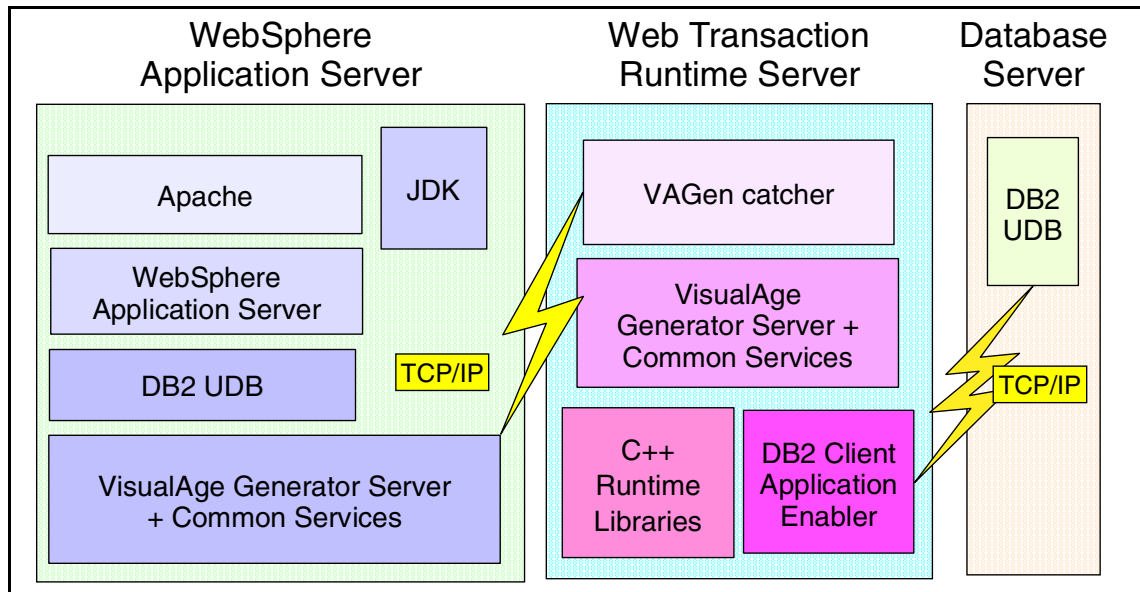


Figure 154. Native NT and UDB implementation

The prerequisite software to run VisualAge Generator Web Transactions is as follows:

WebSphere Application Server

- Windows NT V4 with fixpack 4.
- TCP/IP.
- A servlet engine which supports JSDK 2.1 and JSP 1.0.

We chose to use IBM WebSphere Application Server, so for Web Transaction support we must use V3.0. It is intended with a fixpack, to allow Web Transactions to be used with servlet engines which only support JSDK 2.0 and JSP 0.91, such as WebSphere Application Server V2, but this is not available with GA.

We chose to use WebSphere Application Server Advanced V3.

- IBM WebSphere Application Server V3 runs as a plug-in to a Web server. The Web servers supported for NT include:
 - Apache V1.3.6 or later
 - IBM HTTP server V1.3.6 or later
 - Lotus Domino Go V4.6.2.5 or later
 - Domino V5.0 or later
 - Netscape Enterprise V3.51 or later
 - Microsoft IIS V4.0 or later

We build configurations using both the Apache and IBM HTTP. web servers.

- IBM WebSphere Application Server Advanced requires:
 1. One of the following database products to support the implementation of the *entity* flavor of Enterprise JavaBeans and the storage of configuration data which is managed via EJB:
 - IBM DB2 Universal Database Version 5.2 with fixpack 10 or later.

Note that this is either a full database server on the Web server machine or a Client Application Enabler provided connection to a remote DB2.

The fixpack is VERY important; if you do not apply it, the IBM WebSphere Application Server will not come up and will throw lots of Java exceptions in EJB and JDBC classes.
 - Oracle V8.05 with Driver Manager JDBC-Thin/100% Java for JDK1.1.x.

We chose to use UDB.

2. Java Development Kit V1.1.7B_003.

- VisualAge Generator Server and Common Services

They provide access to the Power Server API, which acts as a layer over the top of other software and communication protocols to allow this environment to invoke a VisualAge Generator server module. They also provide the Gateway Servlet and other associated runtime modules.

- According to the VisualAge Generator install documentation, VisualAge Generator Server and Common Services require one of the following C++ runtime libraries:
 - Microsoft Visual C++ V5
 - Microsoft Visual C++ V6
 - VisualAge for C++ V3.5
 - VisualAge for C++ V3.6

We chose to use VisualAge for C++ V3.5.3.

Note: You may not actually need the C++ runtime libraries on a platform that will only implement Gateway Servlet processing in a WebSphere Application Server environment.

Web Transaction runtime server

- Windows NT V4 with fixpack 4.
- TCP/IP.
- VisualAge Generator Server and Common Services.
- We chose to use VisualAge for C++ V3.5.3 to satisfy the C++ compiler and runtime libraries requirement for building Web Transaction programs that will run in Windows NT with VisualAge Generator Server.
- For our database access we connected to a separate database server machine, so we installed DB2 Client Application Enabler V5.2 with fixpack 8 on the VisualAge Generator server machine.

Database server

- Windows NT V4 with fixpack 4.
- DB2 Universal Database V5.2 with fixpack 8.

12.1.2 Implementation tasks

To implement a Windows NT-based Web Transaction runtime environment, you need to:

- Implement a VisualAge Generator runtime environment with a CSOGW.properties configuration that uses the TCP/IP Web Transaction listener (see Chapter 13, “VisualAge Generator Web Transaction runtime setup” on page 257).
- Install and configure a Gateway Servlet in a WebSphere Application Server environment (see Chapter 14, “WebSphere Application Server setup” on page 301).
- Generate a Web Transaction program (see Chapter 15, “Web Transaction generation” on page 335).
- Implement the generated code in the runtime environment (see Chapter 16, “Running Web Transactions” on page 341).

12.2 CICS for NT Web Transactions

Our runtime implementation is shown in Figure 155.

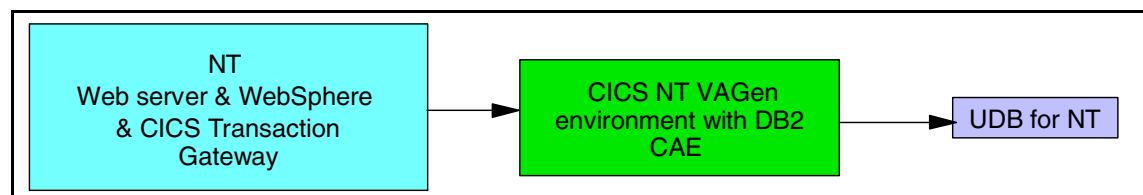


Figure 155. NT CICS and UDB scenario

12.2.1 Software requirements

The software components used for the CICS on Windows NT-based runtime configuration are shown in Figure 156.

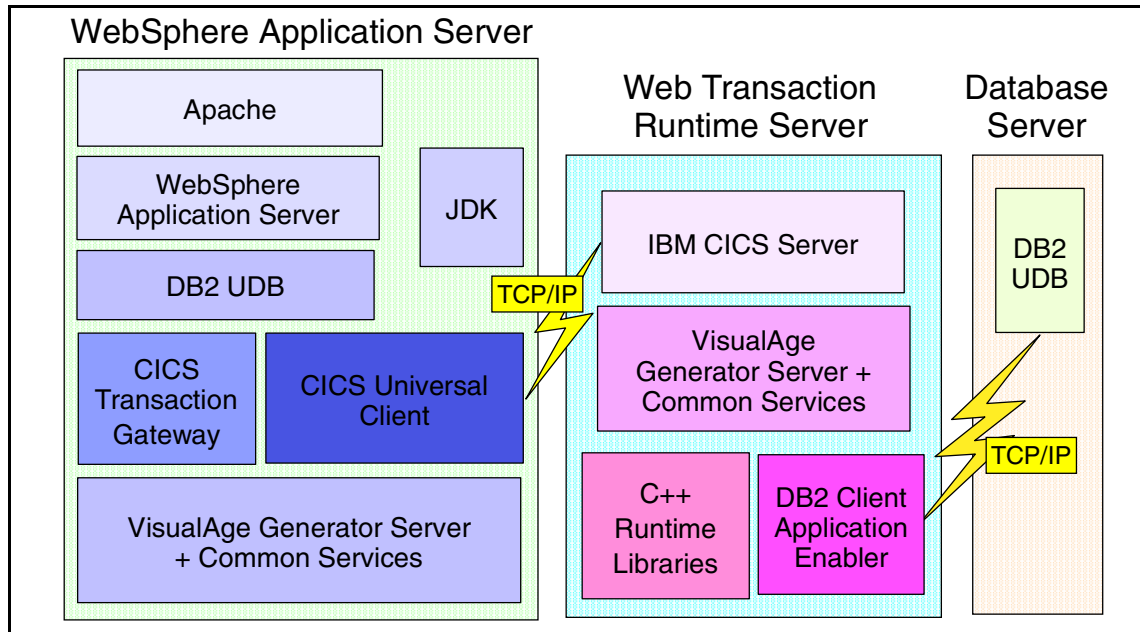


Figure 156. NT CICS and UDB implementation

The prerequisite software to run VisualAge Generator Web Transactions is as follows:

WebSphere Application Server

- Windows NT V4 with fixpack 4.
- TCP/IP.
- A servlet engine which supports JSDK 2.1 and JSP 1.0.

We chose to use IBM WebSphere Application Server, so for Web Transaction support we must use V3.0. It is intended with a fixpack, to allow Web Transactions to be used with servlet engines which only support JSDK 2.0 and JSP 0.91, such as WebSphere Application Server V2, but this is not available at GA.

We chose to use WebSphere Application Server Advanced V3.

- IBM WebSphere Application Server V3 runs as a plug-in to a Web server. The Web servers supported for NT include:
 - Apache V1.3.6 or later
 - IBM HTTP server V1.3.6 or later
 - Lotus Domino Go V4.6.2.5 or later
 - Domino V5.0 or later

- Netscape Enterprise V3.51 or later
- Microsoft IIS V4.0 or later

We build configurations using both the Apache and IBM HTTP. web servers.

- IBM WebSphere Application Server Advanced requires:
 1. One of the following database products to support the implementation of the *entity* flavor of Enterprise JavaBeans and the storage of configuration data which is managed via EJB:
 - IBM DB2 Universal Database Version 5.2 with fixpack 10 or later.
Note that this is either a full database server on the Web server machine or a Client Application Enabler provided connection to a remote DB2.
 - Oracle V8.05 with Driver Manager JDBC-Thin/100% Java for JDK1.1.x.

We chose to use UDB.

2. Java Development Kit V1.1.7B_003.

- VisualAge Generator Server and Common Services
They provide access to the Power Server API, which acts as a layer over the top of other software and communication protocols to allow this environment to invoke a VisualAge Generator server module. They also provide the Gateway Servlet and other associated runtime modules.
- According to the VisualAge Generator install documentation VisualAge Generator Server and Common Services require one of the following C++ runtime libraries:
 - Microsoft Visual C++ V5
 - Microsoft Visual C++ V6
 - VisualAge for C++ V3.5
 - VisualAge for C++ V3.6

We chose to use VisualAge for C++ V3.5.3.

Note: You may not actually need the C++ runtime libraries on a platform that will only implement Gateway Servlet processing in a WebSphere Application Server environment.

- Finally, we chose to use CICS for NT as the environment for our Web Transactions. The Power Server API requires CICS Transaction Gateway to be able to communicate with the CICS server. We used V3.0.2.

The Transaction Gateway includes two major components of use with VisualAge Generator:

- Transaction Gateway Classes
- CICS Universal Client

Web Transaction runtime server

- Windows NT V4 with fixpack 4.
- TCP/IP
- We chose to use CICS for NT (TXSeries) as the environment for our Web Transactions. We used both V4.2 plus latest patches and V4.3 (as shipped with the Enterprise Edition of WebSphere Application Server).
- VisualAge Generator Server and Common Services.
- We chose to use VisualAge for C++ V3.5.3 to satisfy the C++ compiler and runtime libraries requirement for building Web Transaction programs that will run in a Windows NT-based CICS environment with VisualAge Generator Server.
- For our database access we connected to a separate database server machine, so we installed DB2 Client Application Enabler V5.2 with fixpack 8 on the VisualAge Generator server machine.

Database server

- Windows NT V4 with fixpack 4.
- DB2 Universal Database V5.2 with fixpack 8.

12.2.2 Implementation tasks

To implement a Web Transaction runtime environment in CICS on a Windows NT system, you need to:

- Implement a VisualAge Generator runtime environment (including the required CICS software) with a CSOGW.properties configuration that uses the CICSECI to invoke Web Transaction programs (see Chapter 13, “VisualAge Generator Web Transaction runtime setup” on page 257).
- Install and configure a Gateway Servlet, with support for the CICS Transaction Gateway, in a WebSphere Application Server environment (see Chapter 14, “WebSphere Application Server setup” on page 301).
- Generate a Web Transaction program (see Chapter 15, “Web Transaction generation” on page 335).

- Implement the generated code in the runtime environment with the appropriate CICS/ESA security configuration (see Chapter 16, “Running Web Transactions” on page 341).

12.3 CICS/ESA Web Transactions

Our runtime implementation is shown in Figure 157.

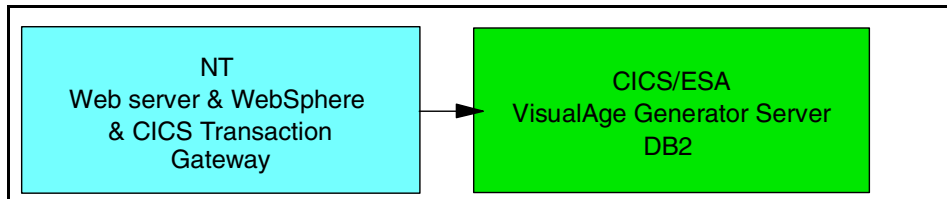


Figure 157. CICS/ESA and DB2 scenario

12.3.1 Software requirements

The software components used for the CICS/ESA runtime configuration are shown in Figure 158.

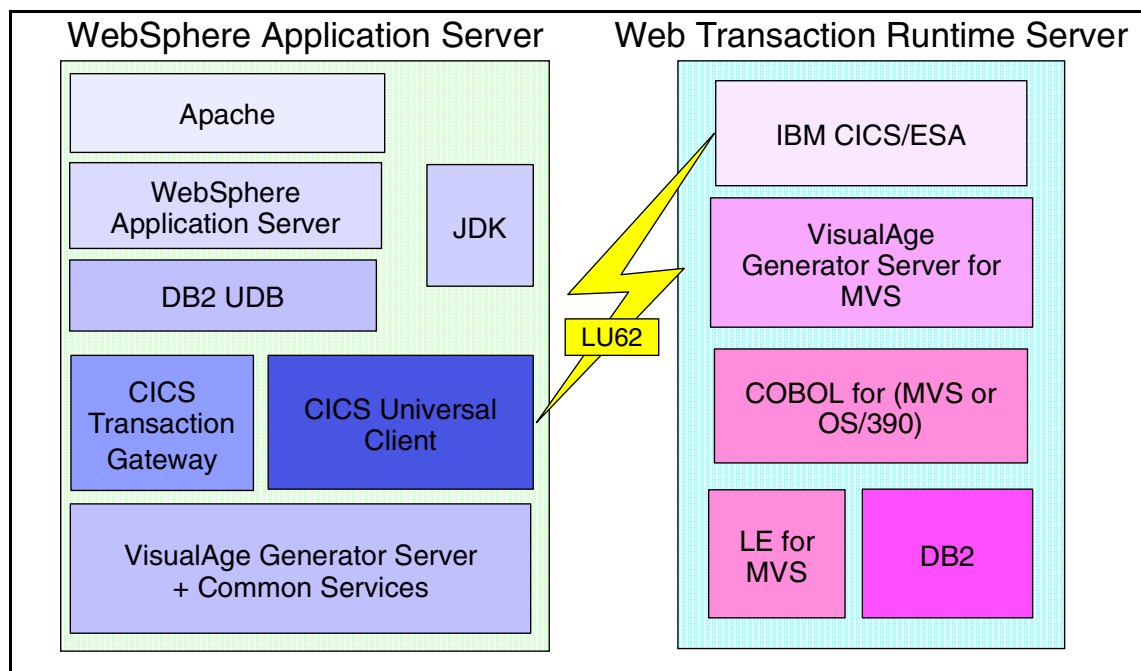


Figure 158. CICS/ESA and DB2 implementation

The prerequisite software to run VisualAge Generator Web Transactions is as follows:

WebSphere Application Server

- The initial software required for the WebSphere Application Server required for CICS/ESA is the same as that listed in 12.2, “CICS for NT Web Transactions” on page 250.
- In addition, IBM Personal Communications V4.0 or later for APPC support to allow CICS client to communicate with CICS/ESA is required. Only the Web server requires the APPC and CICS client software, not the client browser machines.

Note: You can also use TCP62 (APPC over TCP/IP) to establish an APPC connection to the target CICS/ESA runtime platform.

Web Transaction runtime server

- At least MVS/ESA V4.2. There are some restrictions, so please refer to the installation manual.
- SMP/E V1.8.1 or later.
- IBM High Level Assembler V1.2 or later.
- IBM COBOL for MVS V1.2 or later or IBM COBOL for OS/390 V2.1 or later. This is required for preparation of the Web Transactions.
- IBM Language Environment for MVS V1.5 or later.
- TCP/IP V3.1 or later or IBM 3270/PC File Transfer Program/MVS Release 1, so that VisualAge Generator Developer machines may transfer COBOL source code and JCL to the host at generation time to initiate and the preparation step.
- CICS/ESA V3.3 or later.
- IBM Database 2 V3.1 or later.

12.3.2 Implementation tasks

To implement a Web Transaction runtime environment in a CICS/ESA system, you need to:

- Implement a VisualAge Generator runtime environment with a CSOGW.properties configuration that uses the CICSECI (and the required CICS connection software) to invoke Web Transaction programs (see Chapter 13, “VisualAge Generator Web Transaction runtime setup” on page 257).

- Install and configure a Gateway Servlet, with support for the CICS Transaction Gateway, in a WebSphere Application Server environment (see Chapter 14, “WebSphere Application Server setup” on page 301).
- Generate a Web Transaction program (see Chapter 15, “Web Transaction generation” on page 335).
- Implement the generated code in the runtime environment with the appropriate CICS security configuration (see Chapter 16, “Running Web Transactions” on page 341).

Chapter 13. VisualAge Generator Web Transaction runtime setup

The goal of this chapter is to set up and configure a runtime environment where we can deploy VisualAge Generator Web Transaction programs.

13.1 Base software

Support software installed on all workstations is reviewed in this section.

13.1.1 DB2 Client Application Enabler

We used a shared DB2 system installed on a workstation accessible on the network. To implement access to this shared DB2 system, we did the following:

- Install the software and apply the fixpack(s).
- Define a connection to the UDB on your designated Database Server. This is easily done via the **Client Configuration Assistant**.
 - Start the tool and ask to **ADD....** Figure 159 shows the window which opens.

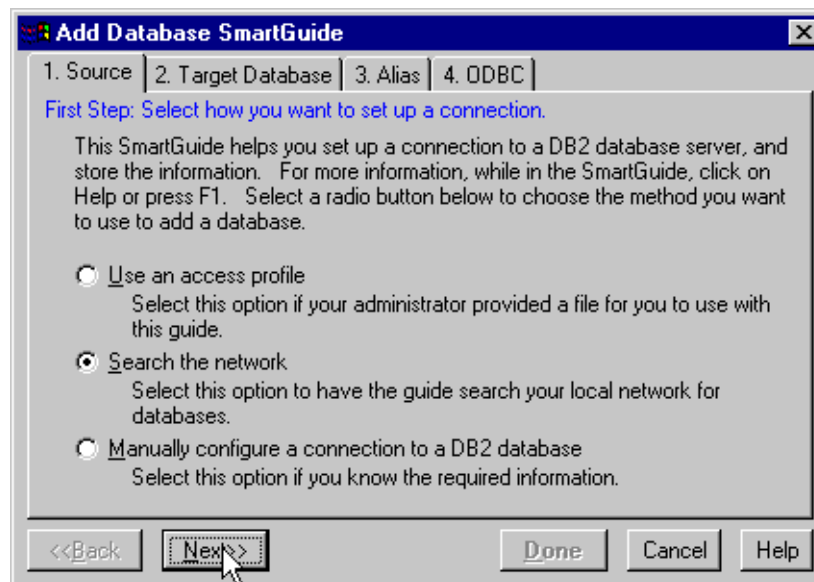


Figure 159. Attaching DB2 CAE to a database on a remote database server — part 1

- The simplest way is to ask to search the network for available database server machines. This produces the window shown in Figure 160 below.

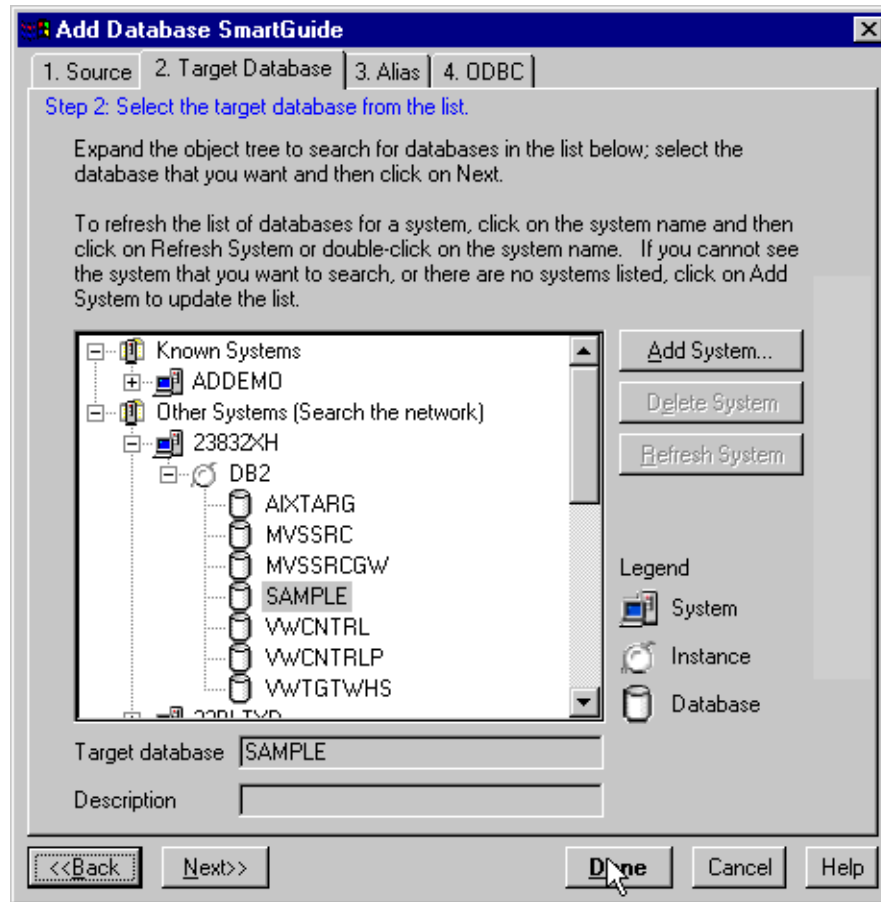


Figure 160. Attaching DB2 CAE to a database on a remote database server — part 2

- Select the database you desire and click **Done**. This will establish a connection to this database using TCP/IP. Once the database connection is defined, you can edit the connection properties if you want to change the protocol or port.

13.1.2 VisualAge for C++

We installed VisualAge for C++ V3.5 with support for development (compilation) and runtime. The nmake command must be functional to prepare a VisualAge Generator Web Transaction program for Windows NT or CICS on a Windows NT system.

After installation, we reviewed the Windows NT system environment variables and corrected several (such as INCLUDE and LIB) where the installation definition for the user variable did not include the system variable reference (for example, %LIB%).

13.1.3 VisualAge Generator Server

We installed VisualAge Generator Common Services and VisualAge Generator Server software. In the several different installations we used either the default target directories or target directories of d:/vc and d:/vs.

13.1.4 Setting up FTP support for program preparation

When you generate, you can ask that preparation processing use FTP to transfer the generated components (Java code, JSPs, and program source) to another machine for preparation.

To set up FTP support, you need to:

- Add x:\IBM\JAVA\IDE\PROGRAM to the PATH environment variable
- Install FTP support using Microsoft Peer Web Services
- Configure the FTP web service:
 - Start the Internet Service Manager using the **Start->Microsoft Peer Web Services->internet service manager** program start option.
 - Select the FTP service as shown in Figure 161.

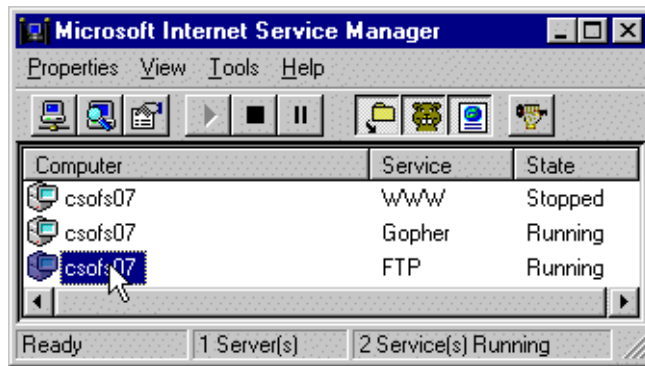


Figure 161. Configuring FTP — part 1

- Specify a username and password which will be used to connect to this machine, as shown in Figure 162.

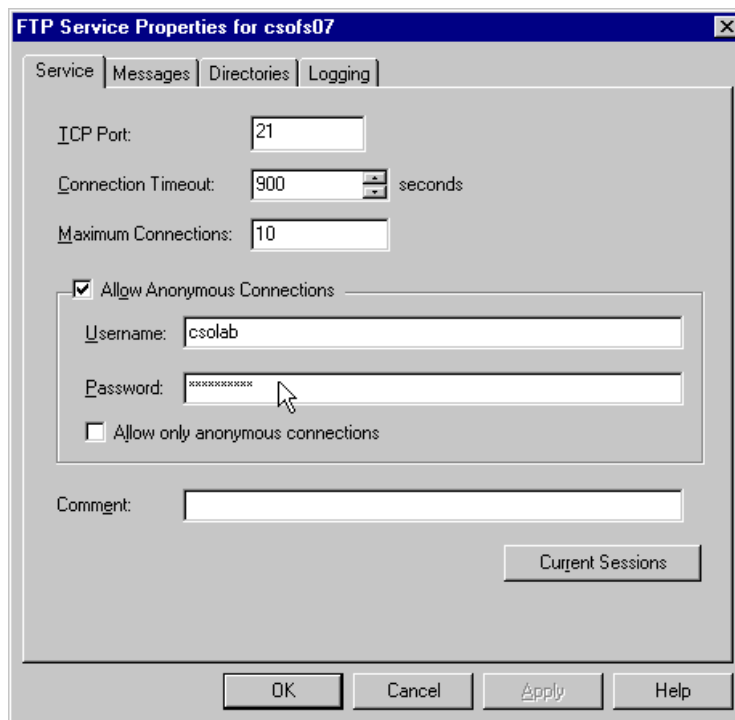


Figure 162. Configuring FTP — part 2

- Click to the directories tab and ask to add a directory, as shown in Figure 163.

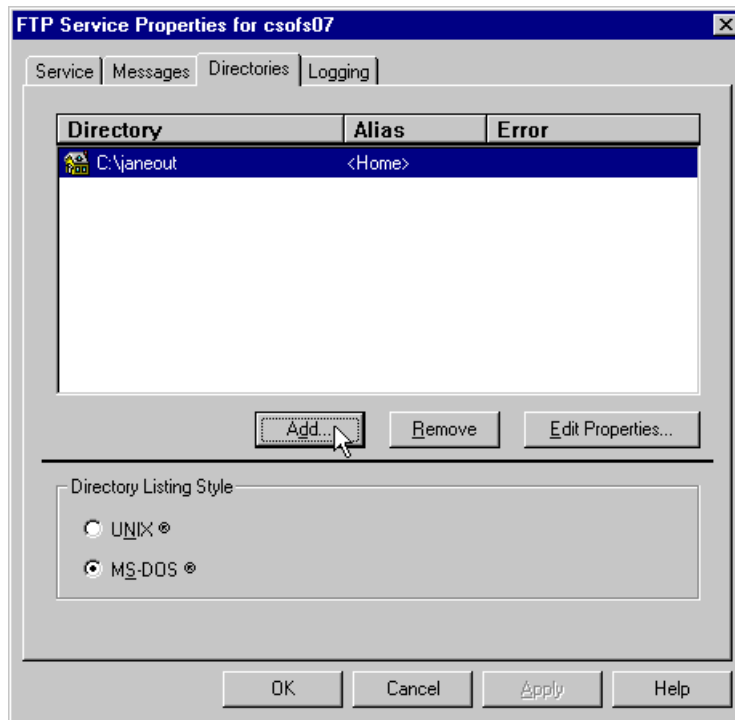


Figure 163. Configuring FTP — part 3

- You must name a home directory to share if none exists. In our case we wish to allow write access, as shown in Figure 164.

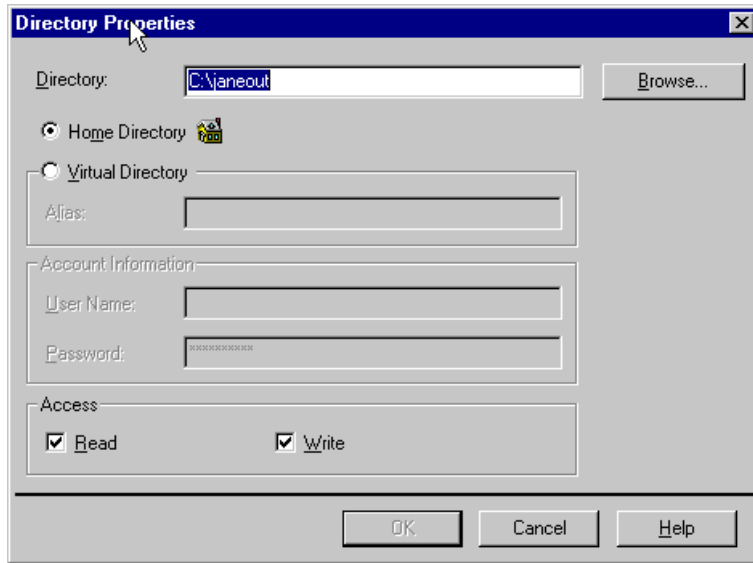


Figure 164. Configuring FTP — part 4

- Finally, we need to specify the directory as shared using Windows NT. In Windows Explorer, select the directory, click with mouse button 2 for the context menu, and choose **Sharing...** This will give you Figure 165.

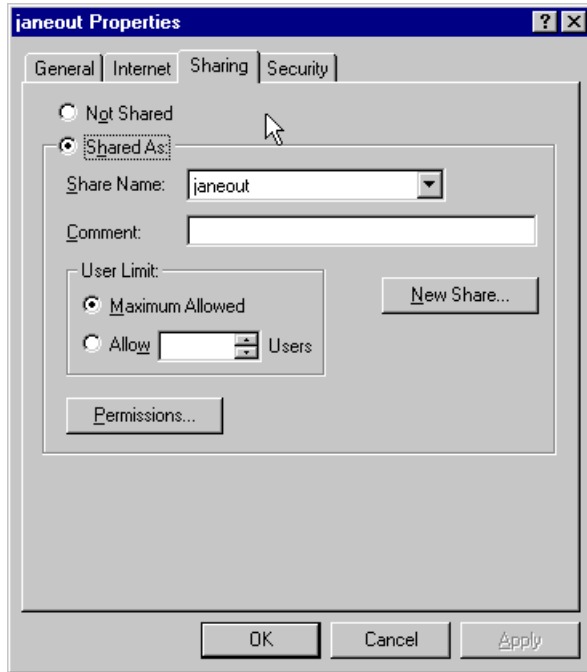


Figure 165. Configuring FTP — part 5

- Set up the sharing and add permission for the userid previously defined in the FTP service.

Preparation control generation options can be defined to identify the target machine name, the FTP service userid and password, and a target directory. The directory location is relative to the FTP home directory, so if you want the code placed in the home directory, enter a \ as the directory value (do not leave it blank).

If you are having problems, there is a trace you can write out to FTP.OUT. You turn it on by setting the FCETROPT environment variable to 3.

13.2 Web Transaction gateway interface configuration (csogw.properties)

The Gateway Servlet interface to the Web Transaction runtime platform is defined in the CSOGW.properties file (found in the VisualAge Generator Server installation directory). Control, application, and server linkage entries define processing, trace, platform, and Web Transaction control values.

13.2.1 Control entries

The refresh interval entry controls Gateway Servlet processing and is defined as `hptGateway.propertiesRefreshInterval=n`, where n equals number of minutes.

This refresh setting determines how often the CSOGW.properties file is re-read from disk into memory. If n is set to 0, the file is never re-read unless you stop and then start the IBM WebSphere Application Server.

Changes to settings in the CSOGW.properties file will not necessarily be picked up instantaneously when the file is re-read; connection settings which are in use for a conversation with an end user will be stored in the active session object until that conversation ends.

13.2.2 Application entries

CSOGW.properties application entries are shown in Table 8.

Table 8. CSOGW property file: application definition

application.WEBTRAN.xxx	The value for webtran is the name of the actual Web Transaction as defined to the VisualAge Generator server environment (the CICS definition or DLL name, for example). The incoming webtran value to match against comes from the selected item in an HTML SELECT list or HIDDEN field called hptAppld. Make sure webtran is in upper case or you may get entry point failures.
application.WEBTRAN=sname	You can use generic short name (sname) values terminated with a wild card (*) character. For example, application.C*=cicsa will apply to all Web Transaction programs that start with a C, unless a more specific match is found (best match used regardless of order of entries in the file). You can have several different application entries pointing at a set of serverLinkage entries for a single arbitrary name.
application.WEBTRAN.traceFlag=1	This setting turns on tracing of the Web transaction to help with debugging.

Note: Some example CSOGW.properties files shipped with VisualAge Generator include the entries `hptRuntimeProperties` and `hptErrorPackage`. These are not used by the Gateway Servlet. These entries were part of an early design, but they were not removed from the example.

13.2.3 serverLinkage entries

CSOGW.properties serverLinkage entries are shown in Table 9.

Table 9. CSOGW property file: serverLinkage definition

serverLinkage.sname	This value is used as a prefix for all entries that define the linkage table attributes that will be used for the named resource.
serverLinkage.sname.commtype=a	The commtype value identifies the communication services that will be used to invoke the Web Transaction. (Compare with remotecommtypes linkage table parameter) Valid commtype values are discussed in Table 10.
serverLinkage.sname.location=c	The location value configures the communication service. (Compare with location linkage table parameter) Valid location values are discussed in Table 10.
serverLinkage.sname.serverid=d	serverid is equivalent to the SERVERID linkage table option used by the Power Server API for typical VisualAge Generator client/server systems. (Compare with serverid linkage table parameter) Valid serverid values are discussed in Table 10.
serverLinkage.sname.groupid=f	The groupid value represents a security grouping in the IMSTCPIPOTMA connection (ITOC). Valid only for IMS target systems.
serverLinkage.sname.destid=g	The destid value represents the IMS system the request is sent to as defined in the IMSTCPIPOTMA connection (ITOC). Valid only for IMS target systems.
serverLinkage.sname.contable=b	The contable value defines how data should be converted (code page format) when passed between the web server and the target location for the VisualAge Generator Web Transaction. Sun Java conversion routines are actually used to do the conversion. Conversion processing is sensitive to the defined structure for the UI Record. Contable value format: CSOzxxxx , where z=binary format and xxxx=code page of target machine. Valid values for z include: I (Intel) or E (EBCDIC). (Compare with ConTable linkage table parameter)
serverLinkage.sname.javaProperty=e	The JavaProperty value identifies the Java package where the data bean and interface bean for the Web Transaction can be found. This value is case sensitive (invalid values generate null pointer exceptions at runtime).

13.2.4 Protocol specific entries

The serverLinkage location and serverid values, by commtype, are shown in Table 10.

Table 10. Valid locations and serverids for a given commtype

COMMTYPE	LOCATION	SERVERID
CICSECI	CICS system identifier. This corresponds to the server name as specified in the CICSCLI.INI file of the CICS client.	CICS transaction name. CPMI is the default CICS server mirror transaction.
TCP/IP	TCP/IP host name where VisualAge Generator server programs are.	Port number to be used by the csotcpui catcher program.
TCP/IMS	TCP/IP host name where VisualAge Generator server programs are.	Port number to be used by the IMS catcher program.

Unlike in a linkage table definition (used to define the client/server interface), the LUWCONTROL and PARMFORM settings available in the PowerServer API cannot be specified in a serverLinkage entry. Logical unit of work (LUW) control is effectively fixed as SERVER unit of work when a Web Transaction directly accesses a database. A linkage table would be used to define the processing rules for server calls made by a Web Transaction.

13.2.5 Overriding serverLinkage entries

The control values can be altered for a named Web Transaction by using an application entry to override a serverLinkage entry.

For example, while most settings define the basic communication technique, for CICS, the serverid value also defines the runtime transaction ID. You may need to adjust certain settings for specific target Web Transactions.

Figure 166 shows how the runtime transaction value can be altered for a named Web Transaction.

```
hptGateway.propertiesRefreshInterval=2

application.C*=CICSNT

application.CHRIS=CICSNT
application.CHRIS.serverid=SLMI

serverLinkage.CICSNT.commtype=cicseci
serverLinkage.CICSNT.contable=csol1252
serverLinkage.CICSNT.location=CICSTCP
serverLinkage.CICSNT.serverid=VGMI
serverLinkage.CICSNT.javaProperty=vgwt.beans
```

Figure 166. Overriding CSOGW.properties file entries

In Figure 166 all Web Transactions with names that start with a C will use the CICSNT serverLinkage definition set. The Web Transaction named CHRIS will also use the CICSNT serverLinkage settings, but with the serverid override defined as part of the application entry, so the SLMI transaction will be used in the target CICS system for the Web Transaction CHRIS.

If you use application entries to override serverLinkage entries, the Web Transaction name entered as part of the application entry override (application.WEBTRAN) must match the name in the application entry where we got a match against the incoming Web Transaction name.

For example, this set of entries will work (CHRIS will use SLMI):

```
application.C*=CICSNT
application.CHRIS=CICSNT
application.CHRIS.serverid=SLMI
```

However, these two do not (null pointer exception):

```
application.C*=CICSNT
application.CHRIS.serverid=SLMI
```

Matching incoming Web Transaction names with the appropriate application entry is done by searching for the most specific entry first, then regressing back to the most generic entry. That is, the order does not matter in the file; the most specific entry match is used.

This is different from the way a linkage table is processed, which is the first valid match found in a top-to-bottom search.

13.3 Windows NT Web Transactions

This section details the installation and configuration of the various software products required to support a native Windows NT runtime platform.

13.3.1 VisualAge Generator control settings

- Set the following environment variables, as appropriate for your required configuration:

FCWDPATH=directory for VisualAge Generator resource association files and VisualAge Generator tables

FCWTROPT=level of server tracing (0,1,2,4,8,16,31)

FCWTROUT=location of trace file

EZERSQLDATE=3 letter code for the date and time format for DB2, for example, EZERSQLDATE=EUR.

FCWRSC=resource association file name, default is FCW.RSC

EZERGRGL_XXX=Gregorian date edit mask using 4 digit years, where XXX is the NLS code.

EZERGRGS_XXX=Gregorian date edit mask using 2 digit years, where XXX is the NLS code.

EZERJULL_XXX=Julian edit mask using 4 digit years, where XXX is the NLS code.

EZERJULS_XXX=Julian date edit mask using 2 digit years, where XXX is the NLS code.

Database configuration environment variables include:

EZERSQLDB=database for VisualAge Generator to attach to

or

DB2DBDFT=database for VisualAge Generator to attach to

Note: EZERSQLDB takes precedence over DB2DBDFT if both are set.

FCWDBUSER=userid to attach to DB2 with

FCWDBPASSWORD=password to go with userid

Note: The userid and password specified must be defined to the platform where DB2 server is installed as a Windows NT user. The userid and password are necessary if you have server authentication set up on the DB2 instance.)

It is recommended that you use a main transaction (text map) program to validate the VisualAge Generator runtime environment before attempting to invoke a Web Transaction program.

13.3.2 Configure TCP/IP listener support

To set up support for the VisualAge Generator Windows NT TCP/IP catcher program, an appropriate entry in the TCP/IP services file is required.

Edit the x:\winnt\system32\drivers\etc\services file. You need to specify a services name and port for the catcher program; add a line like:

```
VAGenWebUI 4200/tcp
```

The service number should correspond to that used in the csogw.properties file when configured for TCP/IP protocol support for Web Transactions. See 16.1, “Deploy generated code” on page 341 for additional details.

We used a command file (see Figure 167) to start the csotcpui VisualAge Generator Windows NT TCP/IP catcher program.

```
set fcwtropt=31
set fcwdbname_db2samp=sample
set fcwdbuser=vgdba
set fcwdbpassword=VGDBA
set ezersqldb=ITSOBANK

set csolinktbl=e:\vglink\vgjavatcpip.lkg
set csotrout=csoUI.out
set csotropt=3

start /min csotcpui
```

Figure 167. TCP/IP catcher program start command

Note: The TCP/IP catcher program name for Web Transactions is csotcpui; this is a different program than the one used to support server calls (csotcps).

The generated Web Transaction programs must be in a directory included in the Windows NT PATH environment variable. See 16.1, “Deploy generated code” on page 341 for additional details.

The linkage table used by a Web Transaction program controls how server programs are called.

13.3.3 Communications configuration

Edit the CSOGW.properties file and create the application and serverLinkage entries that are appropriate for calling a Windows NT-based Web Transaction program.

Figure 168 contains an example of CSOGW entries for TCP/IP-based communication.

```
hptGateway.propertiesRefreshInterval=2

application.WEBTRAN.traceFlag=1
application.CONVMOD.traceFlag=1
application.CXNVMOD=tcwinnt
application.*=tcwinnt

serverLinkage.tcwinnt.commtype=tcip
serverLinkage.tcwinnt.contable=csol1252
serverLinkage.tcwinnt.location=ireland
serverLinkage.tcwinnt.serverid=4200
serverLinkage.tcwinnt.javaProperty=vgtw.beans
```

Figure 168. CSOGW.properties file entries: Windows NT system

Additional detail on csogw.properties file entries is available in 13.2, “Web Transaction gateway interface configuration (csogw.properties)” on page 263.

Note: There were reports of runtime problems when using localhost as the TCP/IP hostname. We were not able to replicate these problems, but you may wish to use the host name identified by the hostname command. The key is being able to ping the identified TCP/IP host.

13.4 CICS for NT Web Transactions

This section details the installation and configuration of Web Transactions in a TXSeries (CICS) environment on Windows NT.

13.4.1 Base software for CICS system

CICS software is available in many forms; we used TXSeries 4.2 and the WebSphere Enterprise Edition V3 packaging of CICS software (4.3). To set up our CICS system on Windows NT we followed the following steps:

- Install the software and apply any patches.
- As our system had no dependencies on Distributed Computing Environment (DCE) services, we configured CICS as a Remote Procedure Call (RPC)-only environment. To set up RPC-only, we had to:

- Run the following command:

```
C:\> cicscp -v -l logFile create dce -R
```

- Set up the following environment variables:

```
CICS_HOSTS=the IP address or hostname of the machine where our
CICS server was installed
```

```
ENCINA_BINDING_FILE=x:\VAR\CICS_SERVERS\SERVER_BINDINGS
```

- Reboot .

13.4.2 Region definition

We decided to use a local Structured File Server (SFS) rather than DB2, so the next step was to define the CICS region. This is done by starting the **CICS Administration Utility**. Choose the **Subsystem** menu option, then **New**, then **CICS region**. The file system we used was local.

Note: There were some reports of problems with the TXSeries administration interface on some systems (unable to enter data in the fields). Changing the Windows NT display properties and selecting a font size of small fonts seemed to correct the problem.

Creating the CICS region actually creates both a region definition and an SFS server definition. You only need to start the CICS region and it will bring up the SFS server also. You start the CICS region by selecting it and using the right mouse button to invoke a context menu and then choosing the **Start...** menu option.

- We found we had to edit x:\VAR\CICS_SERVERS\SERVER_BINDINGS to remove the uuid before we could successfully start the CICS region.

Below are the "before" and "after" versions of the SERVER_BINDINGS entry:

```
././cics/sfs/IRELAND 866f0650-a77d-11d3-bbda-002035aee2f4@ncadg_ip_udp: [10050]
././cics/sfs/IRELAND ncadg_ip_udp: [10050]
```

See the online document *Getting Started with TXSeries for Windows NT*, for more information on installing and initial configuration for CICS for Windows NT.

13.4.3 CICS DB2 attachment

There are two way to attach CICS to DB2 when using VisualAge Generator:

- **Using VisualAge Generator environment variables** — When environment variables are used, the database settings are easy to determine and alter.

- **Using an XA product definition** — Other COBOL and C++ programs can make use of an XA product definition when they need to connect to DB2 and, by defining an XA product definition, you allow CICS to control syncpoint processing instead of each individual VisualAge Generator program.

Environment variable implementation

To set up DB2 access with environment variables:

- Edit the `x:\var\CICS_regions\regionname\environment` file, where *regionname* is the name you chose to give to your CICS region when you defined it. Set the following environment variables inside it:
 - `FCWDBUSER=userid to attach to DB2 with`
 - `FCWDBPASSWORD=password to go with userid`
(The userid and password specified must be defined to the DB2 server machine as an NT user)
 - `FCWTRDB_<transactionid>=database this transaction should attach to`
For example, `FCWTRDB_CPMI=SAMPLE`

If you do not wish to control database attachment as finely as the transaction level, you can set:

- `EZERSQLDB=database for CICS to attach to`
or
- `DB2DBDFT=database for CICS to attach to`

EZERSQLDB takes precedence over DB2DBDFT if both are set.

XA product definition implementation

Note: During the residency we experienced problems trying to use the `cicsxadb2` switch load file due to exceptions being thrown during SYNCPOINT processing. Time restrictions prevented us from resolving the problem. Test your level of code to determine if the problem still exists.

To set up DB2 access with an XA product definition:

- Invoke a DB2 command window and run:
`db2 update dbm cfg using tp_mon_name libEncServer.dll`
- Grant the following authorities to the userid CICSUSER:
bindadd on the database you are connecting to.
select on the SYSIBM.SYSINDEXES table in the database you are connecting to.

- Create the environment variable DB2INSTANCE and set it to the value of the DB2 instance you are attaching to; the default value UDB uses is DB2.
- Select the CICS region on the **CICS Administration Utility**, invoke the context menu, and choose **Resources-> Product**, as shown in Figure 169.

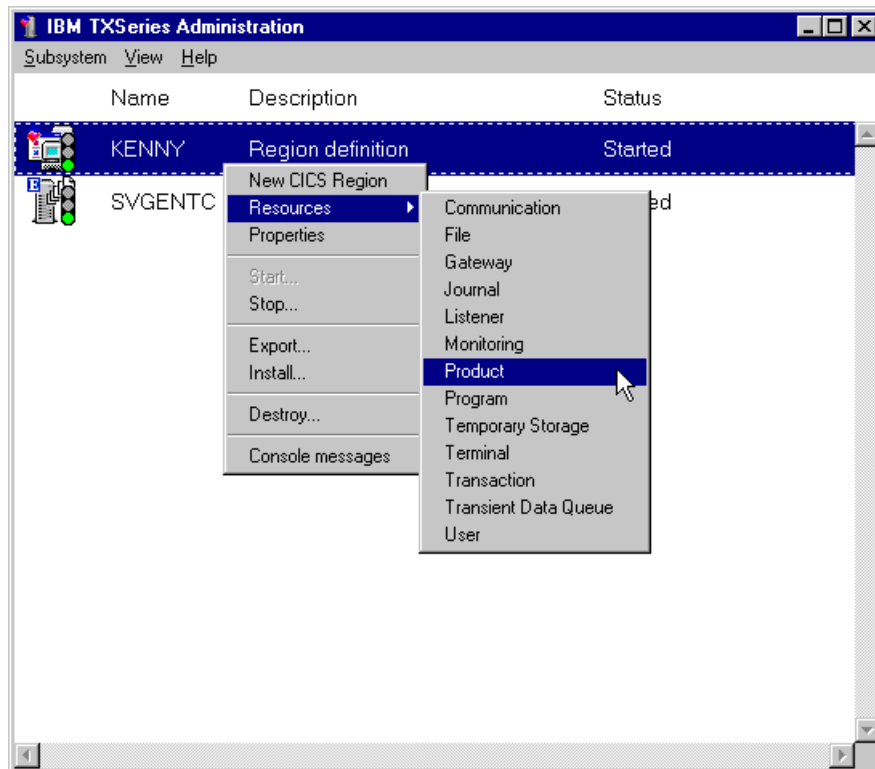


Figure 169. Attaching CICS to DB2 — part1

- When **Product** is clicked, the **Products** window opens. From that window choose the **Products** menu option, then **New....** This opens a definition window, as shown in Figure 170 below.

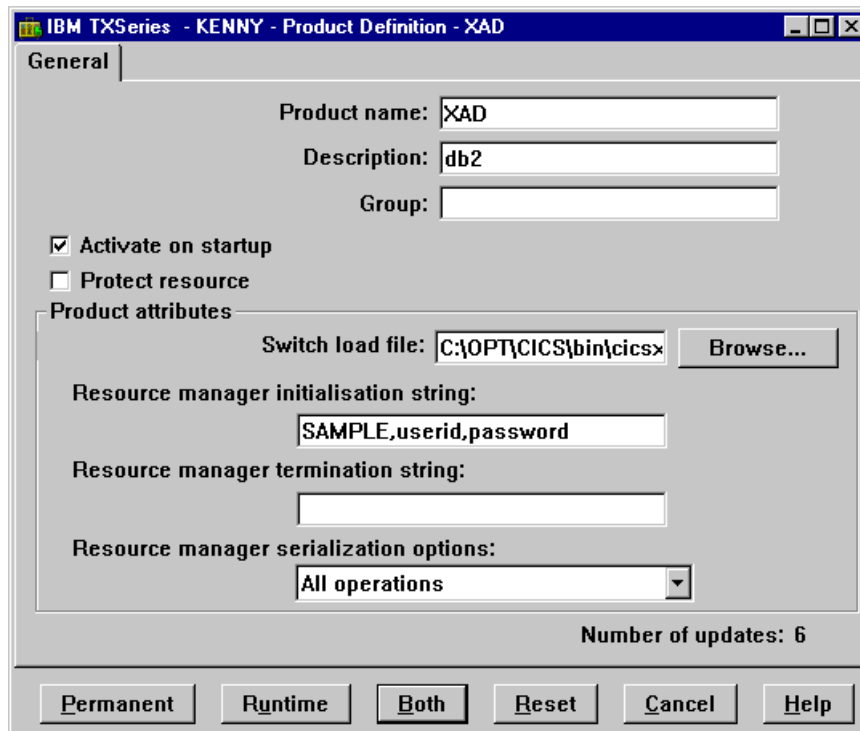


Figure 170. Attaching CICS to DB2 — part2

- Here you can fill in details for:
 - Product name — this is an arbitrary name of your choosing
 - Database you wish to attach to
 - Userid to connect to the database with
 - Password to go with userid

The database, userid, and password are strung together, separated by commas, as the **Resource manager initialization string**.

(If your DB2 instance is set up with client authentication, you need not specify a userid or password, or put in the commas. If you are using server authentication, the userid and password you specify must be defined to the DB2 server machine as an NT user.

Switch load file — x:\OPT\CICS\BIN\cicsxadb2.dll (for two-phase commit) or cics1pcdb2.dll (for single-phase commit).

- Edit the x:\var\CICS_regions*regionname*\environment file, where *regionname* is the name you chose to give to your CICS region when you defined it. Set the following environment variable inside it:

- FCWDBNOOP=yes
- FCWDBUSER=*userid to attach to DB2 with*
- FCWDBPASSWORD=*password to go with userid*

(The userid and password specified must be defined to the DB2 server machine as an NT user. The userid and password are necessary if you have server authentication set up on the DB2 instance.)

- FCWTRDB_<*transactionid*>=*database this transaction should attach to*
For example, FCWTRDB_CPMI=SAMPLE

If you do not wish to control database attachment as finely as transaction level, you can set:

- EZERSQLDB=*database for CICS to attach to*
or
- DB2DBDFT=*database for CICS to attach to*

EZERSQLDB takes precedence over DB2DBDFT if both are set.

The database specified in the XA product definition does not have to be the same as that used in FCWTRDB_<*transactionid*> or EZERSQLDB/DB2DBDFT, or even a database that VisualAge Generator uses. The crucial thing is that FCWTRDB_<*transactionid*> or if not set, EZERSQLDB/DB2DBDFT is the database your VisualAge Generator transactions will use.

- Cold start the CICS region.

See the appropriate CICS documentation for full details on attaching CICS to database servers using XA definitions.

13.4.4 Add CICS system listeners

First cold start the CICS region and then add two listeners:

- Named pipes listener so you can invoke a local terminal on the machine where the CICS server software is installed.
- TCP/IP listener, so CICS clients can talk to the server through TCP/IP.

Adding the named pipes listener:

- Select the CICS region on the **CICS Administration Utility**, invoke the context menu, and choose **Resources-> Listener**.

Choose the **Listeners** menu option and select **New...** Figure 171 shows the window which then opens.

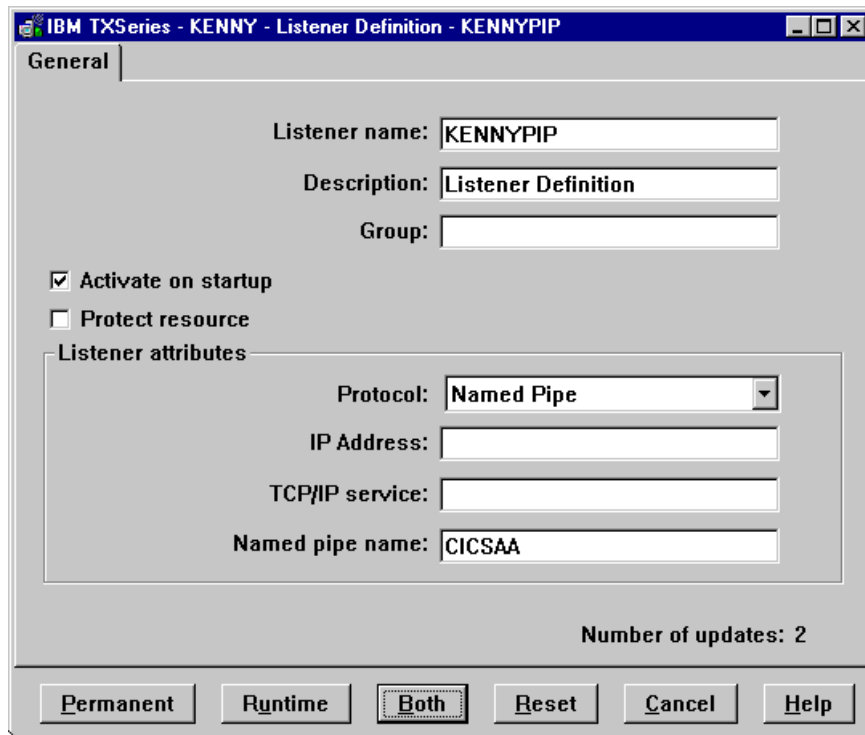


Figure 171. Adding a listener to CICS

The **Listener name** is arbitrary. Choose the **Named Pipe** protocol and specify a **Named pipe name** which corresponds to a NamedPipeName definition in the CICSCLI.INI file.

Adding the TCP/IP listener:

- In a similar way to setting up the named pipes listener, ask to create a new listener so you get the window shown in Figure 171.

Just specify a listener name and select the TCP/IP protocol. If you leave the TCP/IP service value empty, the default will be used for the TCP/IP port (1435).

The listener definition is referenced in 14.3.2, “Customization for TX Series (CICS NT) access” on page 320.

13.4.5 Define CICS user

We chose not to use DCE for our security, but to let CICS do it internally, so we need to set up a USER definition using the **CICS Administration Utility**:

- Select the region, and then using the context menu (mouse button 2), choose **Resources-> User**.
- On the window which opens, choose the **Users** menu option, then **New...**
- Specify a user name, then click the **Security/DCE** tab.
- Uncheck the **None** check box in the security DCE information, and fill in a **CICS password**.

13.4.6 Add VisualAge Generator runtime and debug transactions

The Gateway Servlet uses a CICS mirror transaction when invoking a VisualAge Generator Web Transaction. The definition for the default mirror transaction (CPMI) may need to be altered, or a new transaction defined, to support VisualAge Generator programs in CICS.

To define an alternate runtime transaction:

- Create a new mirror transaction definition (VGMI) based on the CICS-provided (and protected) CPMI mirror transaction.
- Use the same definition for VGMI as found for CPMI.

This alternate mirror transaction definition is referenced in 13.4.8, “Communications configuration” on page 279.

A telnet session will allow you to use the CEDF utility to support debugging when invoking Web Transactions (see 16.2.6, “Debugging Web Transactions at runtime with CEDF” on page 351 for details on how to use the telnet session).

To create a telnet session:

- Run the command `C:>cicscp -v create telnet_server cicsteld`

The output from the command assigns a port; note this down.

- Use **IBM Personal Communications->Start or Configure session**. Choose TCP/IP, **Configure...** and then ask to **Configure Link**. Use **localhost** as the **Host Name**. Click **Advanced....**and specify the **Port Number** as the port the previous command assigned.

13.4.7 VisualAge Generator control settings

After the base software has been installed and the system restarted the following tasks must be performed:

- Define an environment variable named CICSREGION set to the name of the CICS region you set up in 13.4.2, "Region definition" on page 271.
- Run x:\VGSERVW\FCWINSTALL.BAT to set up the CICS definitions for VisualAge Generator Server. This file uses the CICSREGION environment variable from the previous step so you can either reboot the system or open a fresh command window to get the current CICSREGION setting.

Note: You may have to edit the FCWINSTALL.BAT file to adjust the program location values for the VisualAge Generator Server programs that are added to CICS. The program location referenced a \DLL directory that does not exist. The programs are in the VisualAge Generator Server root directory now. They \DLL and \EXE directories were removed to reduce the size of the PATH environment variable.

- Edit the x:\var\CICS_regions*regionname*\environment file, where *regionname* is the name you chose when defining the CICS region. Add settings for any environment variables that you want to change from the system settings. This might include:
 - FCWDPATH=*directory for VisualAge Generator resource association files and VisualAge Generator tables*
 - FCWTROPT=*level of server tracing*
 - FCWTROUT=*location of trace file*
 - EZERSQLDB=*name of target database*
 - EZERSQLDATE=*3 letter code for the date and time format for DB2, for example, EZERSQLDATE=EUR.*
 - FCWRSC=*resource association file name, default is FCW.RSC*
 - EZERGRGL_XXX=*Gregorian date edit mask using 4 digit years, where xxx is the NLS code.*
 - EZERGRGS_XXX=*Gregorian date edit mask using 2 digit years, where xxx is the NLS code.*
 - EZERJULL_XXX=*Julian edit mask using 4 digit years, where xxx is the NLS code.*
 - EZERJULS_XXX=*Julian date edit mask using 2 digit years, where xxx is the NLS code.*

In our CICS NT system we only added an entry for EZERSQLDB. The others were either not required or our system settings were acceptable.

13.4.8 Communications configuration

Edit CSOGW.properties file and create appropriate application and serverLinkage entries. Figure 172 contains an example of csogw.properties entries for CICS-based communication.

```
hptGateway.propertiesRefreshInterval=2

application.WEBTRAN.traceFlag=1
application.CONVMOD.traceFlag=1
application.*=CICSNT

serverLinkage.CICSNT.commtype=cicseci
serverLinkage.CICSNT.contable=csol1252
serverLinkage.CICSNT.location=CICSTCP
serverLinkage.CICSNT.serverid=VGMI
serverLinkage.CICSNT.javaProperty=vgtw.beans
```

Figure 172. CSOGW.properties file entries: CICS NT system

The serverid value matches the CICS mirror transaction defined in 13.4.6, “Add VisualAge Generator runtime and debug transactions” on page 277.

The location value references the CICS system identifier used in the CICS Transaction Gateway configuration (see 14.3.2, “Customization for TX Series (CICS NT) access” on page 320).

Additional detail on csogw.properties file entries is available in 13.2, “Web Transaction gateway interface configuration (csogw.properties)” on page 263.

13.5 CICS/ESA Web Transactions

This section details the installation and configuration of the various software products as required to support the use of Web Transactions in a CICS/ESA environment.

13.5.1 Install the PCOMM software

Configure PCOMM as follows:

- Click on **Start->Programs->Personal Communications->SNA Node Configuration**. You should get a window like that shown in Figure 173.

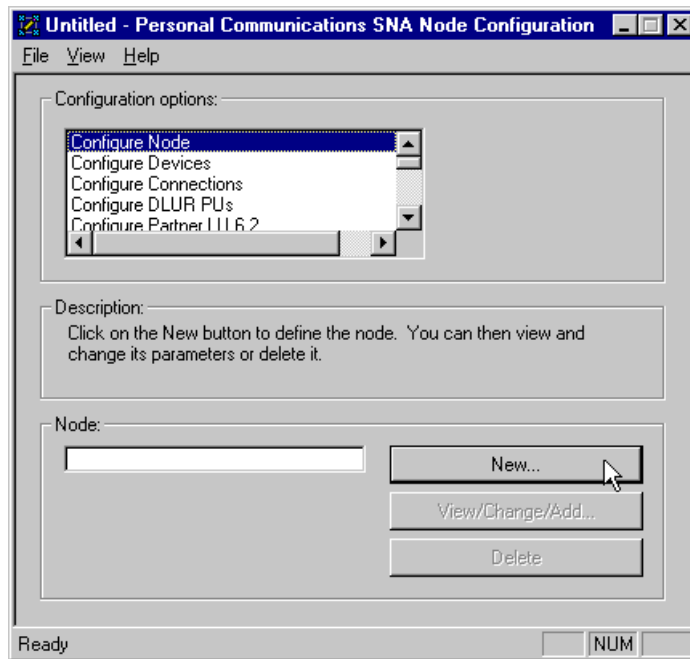


Figure 173. Add a node

- Select **Configure Node** and click on **New...**, this should open a window like Figure 174.

The screenshot shows a dialog box titled "Define the Node" with a close button (X) in the top right corner. The dialog has three tabs: "Basic", "Advanced", and "DLU Requester". The "Basic" tab is selected. Inside the dialog, there are two main sections. The first section is labeled "Control Point (CP)" and contains a "Fully qualified CP name:" label followed by two text boxes: the first contains "SNANETWK" and the second contains "JANESXID", with a period between them. Below this is a "CP alias:" label followed by a text box containing "JANESXID". The second section is labeled "Local Node ID" and contains two labels: "Block ID:" and "Physical Unit ID:". Below "Block ID:" is a text box containing "05D". Below "Physical Unit ID:" is a text box containing "00000". At the bottom of the dialog are four buttons: "OK", "Cancel", "Apply", and "Help".

Figure 174. Basic node details

- Specify your CP name and qualify it with the name of the SNA network you are connecting to. The CP manages the node and its resources and network communication; CPs talk to other CPs.
- The alias provides an alternative to having to use the fully qualified CP name, and if local programs use this, it allows the fully qualified name to be changed without any impact.
- Block represents the product type.
- PU identifies the physical unit. The PU manages the links associated with the node.
- The Block ID should stay the same although the physical Unit ID may change depending on how it is defined on the host system.

- Check what registrations you require on the advanced tab (Figure 175).

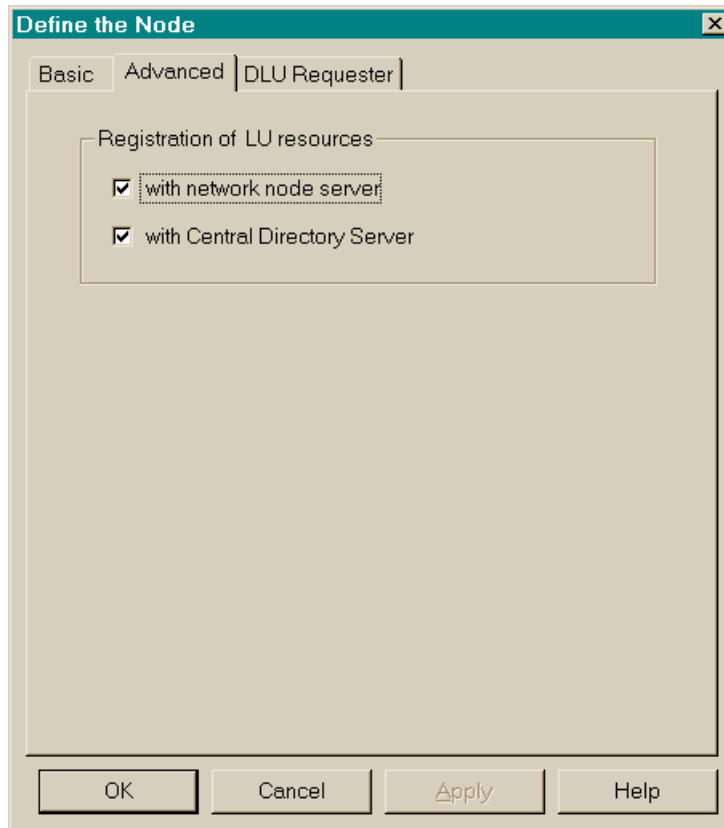


Figure 175. Advanced node details

- The last tab allows you to specify dependent LUs (Figure 176). We do not have any here.

The screenshot shows a dialog box titled "Define the Node" with a close button (X) in the top right corner. It has three tabs: "Basic", "Advanced", and "DLU Requester", with the "DLU Requester" tab selected. The dialog contains the following fields and controls:

- "DLUS name:" followed by two text input boxes separated by a period (.)
- "Backup DLUS name:" followed by two text input boxes separated by a period (.)
- "DLUS connect retry timeout:" followed by a text input box containing the number "5" and the word "seconds".
- "DLUS connect retry limit:" followed by a text input box containing the number "3".

At the bottom of the dialog, there are four buttons: "OK", "Cancel", "Apply", and "Help".

Figure 176. DLU requester details

- Click OK and now choose to configure a new device as shown in Figure 177. The device configuration allows you to specify basic details of the physical communication adapter and port you are using, for example, Token ring or COM port. We chose to use LAN, as we are using a Token ring card.

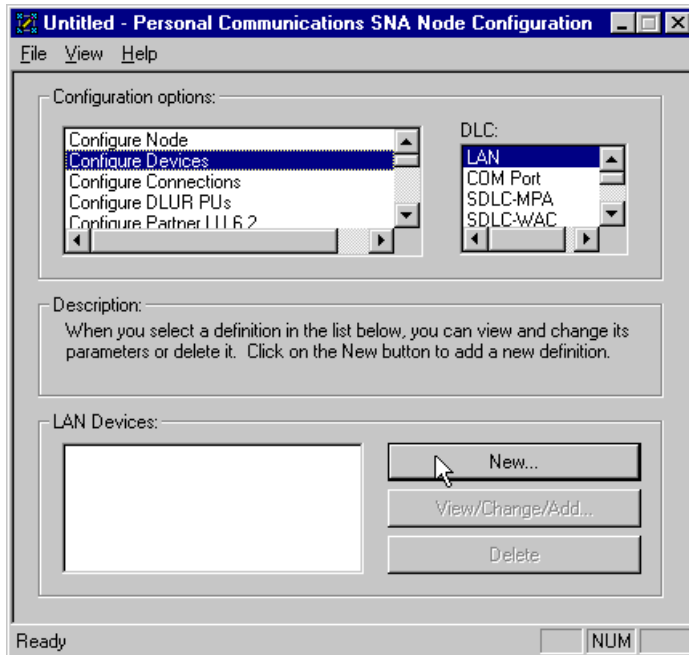


Figure 177. Configure devices

- Figure 178 allows you to input basic details. The name is generated for you, based on the adapter number and local service access point. You are prompted with the possible choices to help you complete this panel.

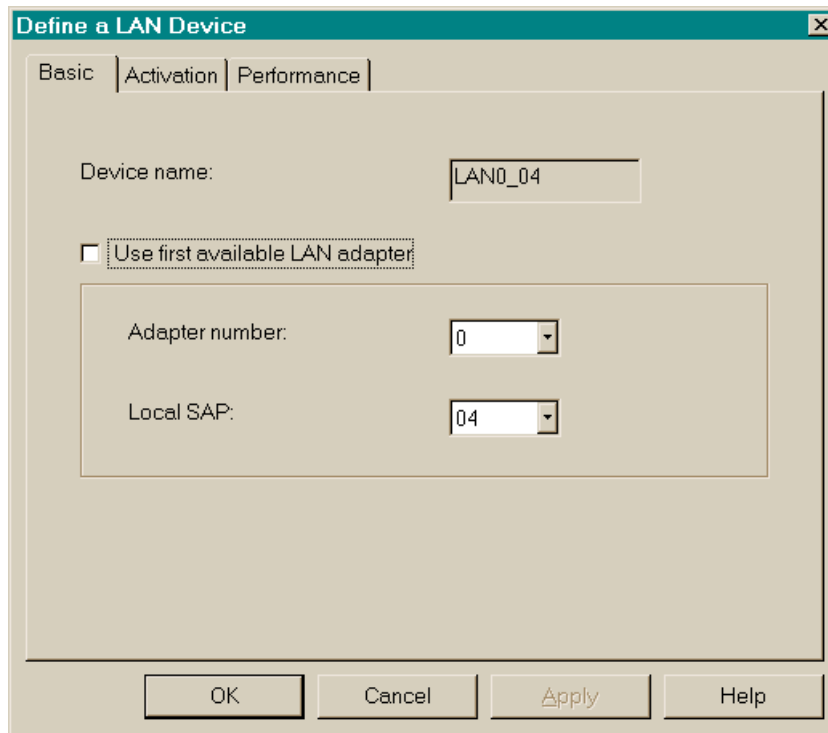


Figure 178. Defining a LAN device — basic

- Figure 179 shows the Activation tab. The maximum PIU size represents the size of the SNA data buffer. The maximum size varies depending on what the device actually is.

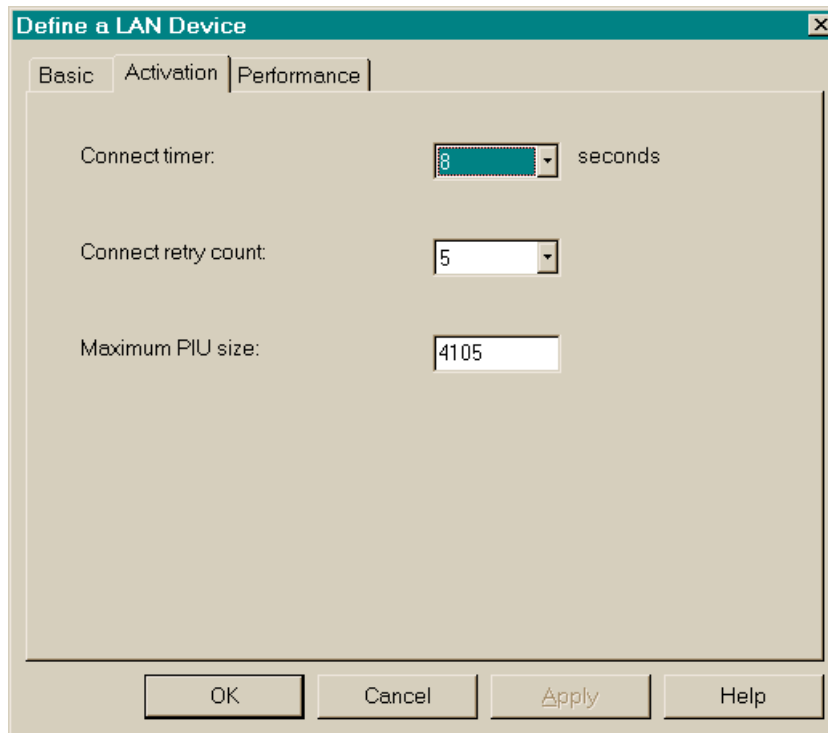


Figure 179. Defining a LAN device — activation

- Figure 180 allows for some tuning. We used the defaults.

The screenshot shows a dialog box titled "Define a LAN Device" with a close button (X) in the top right corner. The dialog has three tabs: "Basic", "Activation", and "Performance", with "Performance" currently selected. The "Performance" tab contains several configuration options, each with a text input field and a unit label:

Parameter	Value	Unit
Inactivity timer - Ti:	30	seconds
Response timer - T1:	3000	ms
Retry count:	10	
Acknowledgement timer - T2:	100	ms
Anticipated outstanding transmits:	16	
Receive window count:	8	
Receive buffer count:	16	

At the bottom of the dialog, there are four buttons: "OK", "Cancel", "Apply", and "Help".

Figure 180. Defining a LAN device — performance

- We now need to specify our mainframe connection. This is shown in Figure 181. We are using LAN again, as we want Token ring.

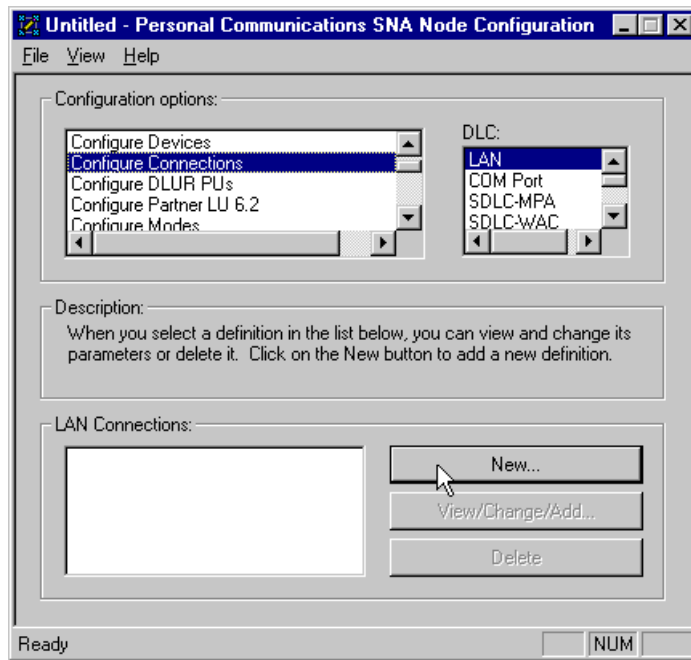


Figure 181. Configure new connection

- We specify to use the adapter we have set up and use the gateway destination address supplied when the Node (Control Point) was set up. These basic details are shown in Figure 182.

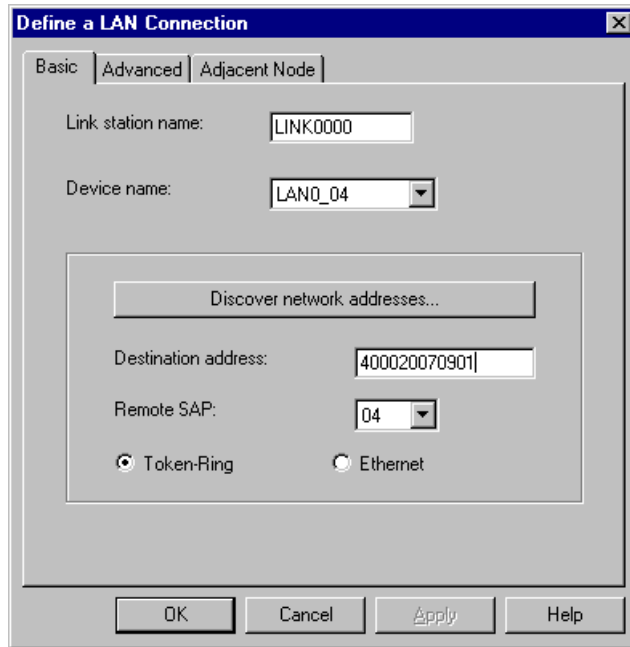


Figure 182. Defining a LAN connection — basic

- Figure 183 shows the advanced panel. A physical unit name is generated for you. Again we have PIU size, block ID and PU ID.

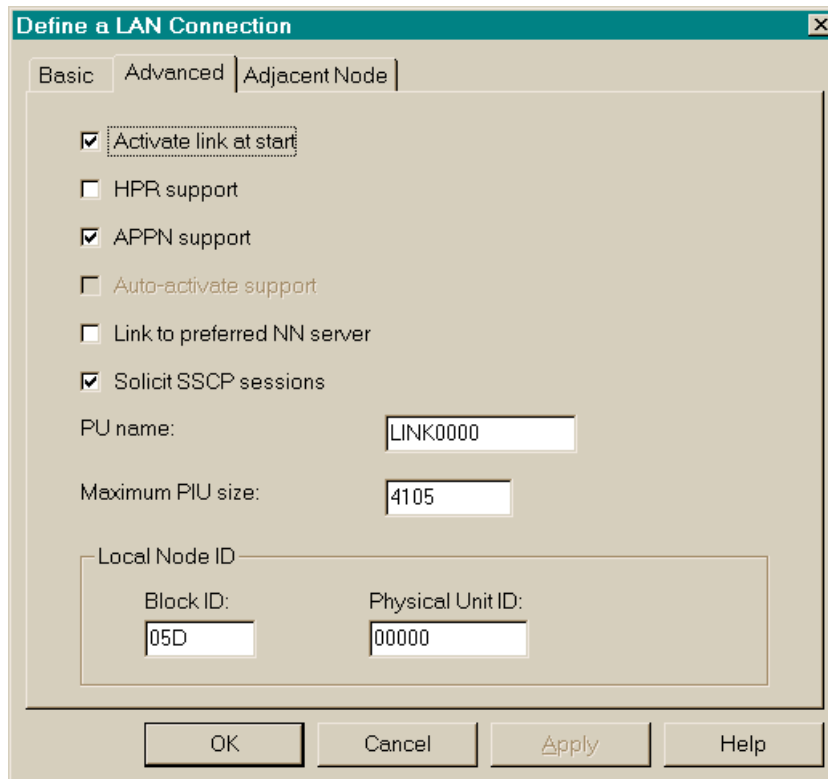


Figure 183. Defining a LAN connection — advanced

- Figure 184 shows optional security details you can fill in about the adjacent node. We left this to default.

The screenshot shows a dialog box titled "Define a LAN Connection" with a close button (X) in the top right corner. The dialog has three tabs: "Basic", "Advanced", and "Adjacent Node", with "Adjacent Node" currently selected. The "Adjacent CP name:" field consists of two text boxes separated by a period. The "Adjacent CP type:" is a dropdown menu set to "APPN Node". The "TG number:" is a dropdown menu set to "0". Below these is a section titled "Adjacent node ID" containing two sub-fields: "Block ID:" with a text box containing "000" and "Physical Unit ID:" with a text box containing "00000". At the bottom of the dialog are four buttons: "OK", "Cancel", "Apply", and "Help".

Figure 184. Defining a LAN connection — Adjacent Node

- Now, in Figure 185, we define the actual CP our local CP is going to communicate with. This is the CICS system.

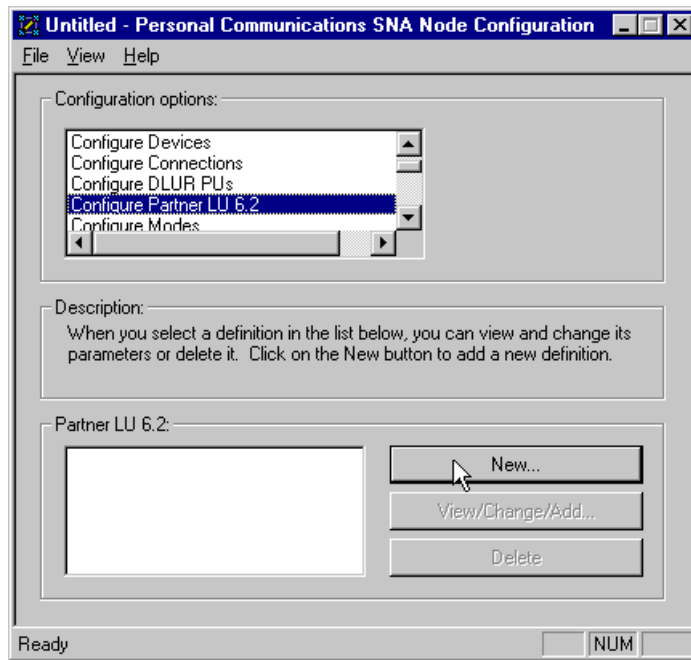


Figure 185. Configure partner LU 6.2

- Figure 186 shows where we can specify the fully qualified CP of the CICS system and its logical LU. The logical LU shown here is the applid of the CICS system. We again have the option of using an alias.

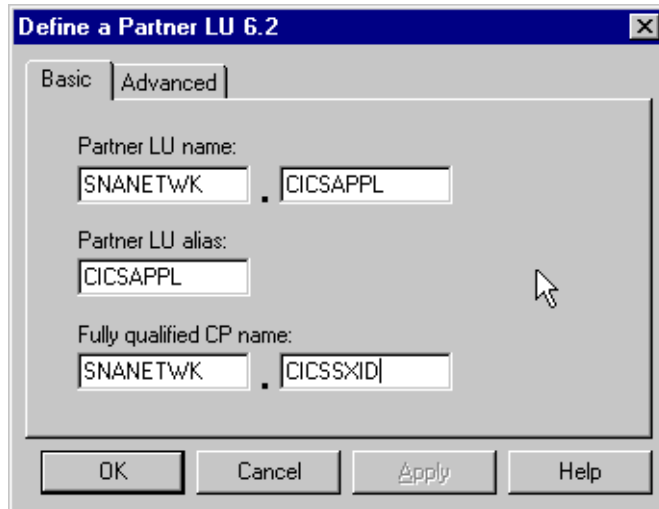


Figure 186. Defining a Partner LU — basic

- Figure 187 shows the advanced tab. Here we can specify the maximum size of data we can send in conversations. 32767 is the limit.

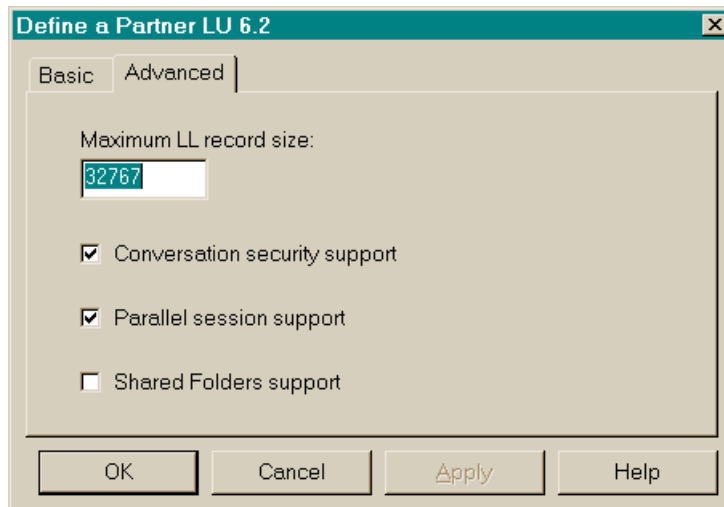


Figure 187. Defining a partner LU — advanced

- Finally in Figure 188 we specify a mode. This mode will have been defined to VTAM and indicates characteristics of the communications you are making; for example that it is a CICS address space you are talking to.

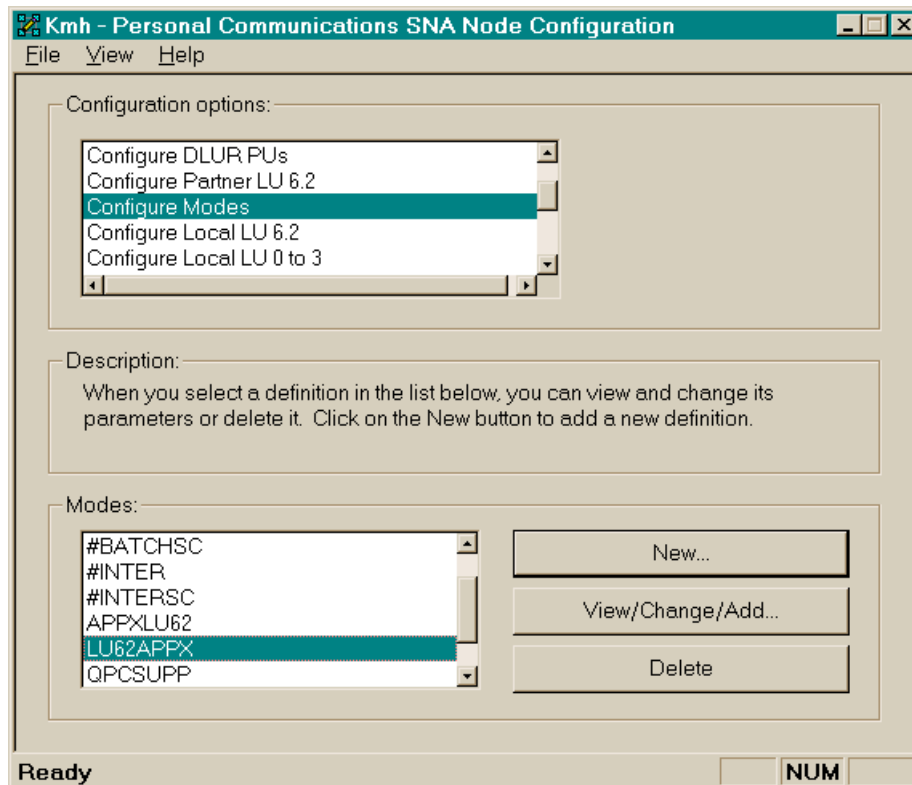
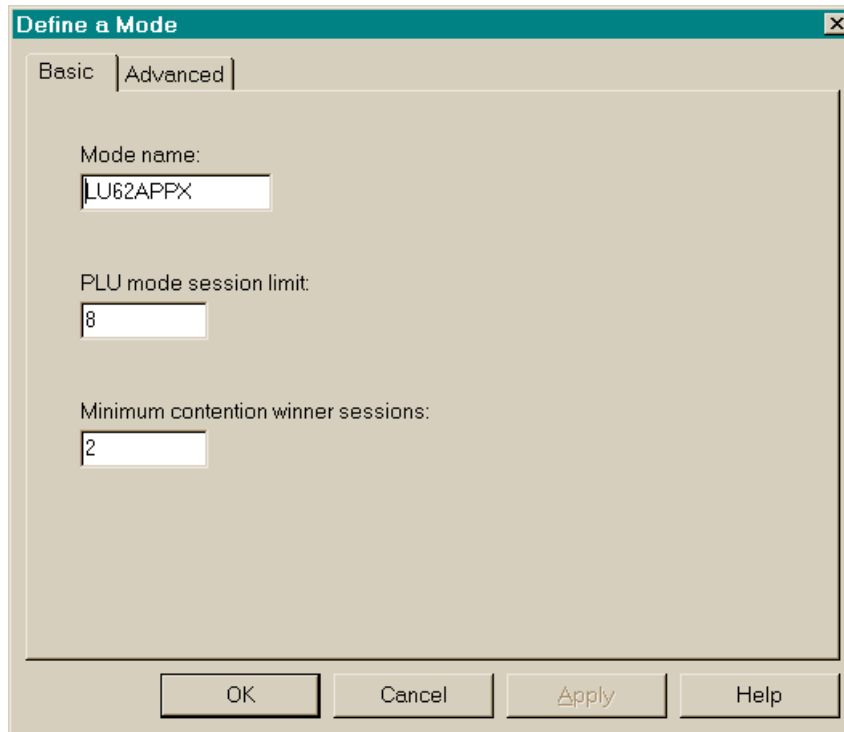


Figure 188. Configure mode

- Figure 189 shows where you can specify the mode name.



The image shows a dialog box titled "Define a Mode" with a close button (X) in the top right corner. It has two tabs: "Basic" (selected) and "Advanced". The "Basic" tab contains three input fields:

- "Mode name:" with a text box containing "LU62APPX".
- "PLU mode session limit:" with a text box containing "8".
- "Minimum contention winner sessions:" with a text box containing "2".

At the bottom of the dialog box, there are four buttons: "OK", "Cancel", "Apply", and "Help".

Figure 189. Defining a mode — basic

- Figure 190 shows the advanced tab. Here you need to specify the class of service; this indicates a set of transport network characteristics that need to be fulfilled for you to be able to get a session.

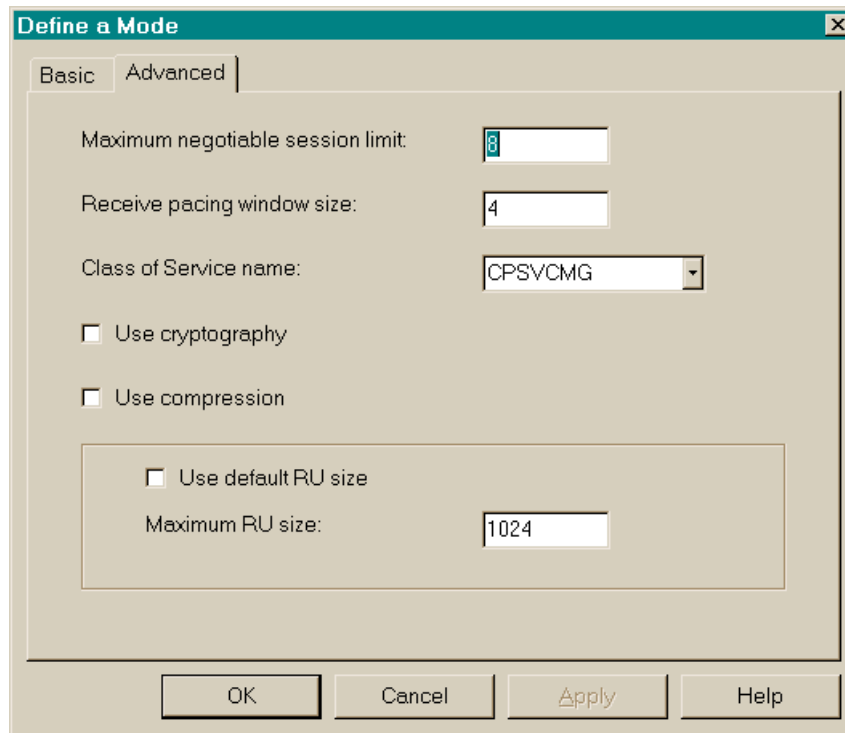


Figure 190. Defining a mode — advanced

- Save your configuration.

13.5.2 CICS connection definition

CICS Sessions/Connections must be defined for XID.

- Use CEDA to define a session for with the same name as your local CP; this session definition gives you your netname. You need to specify a 4-character connection id. This id corresponds to the connection you need to define through CEDA. You will also need to specify the mode name you are using for the APPC protocol. Beware of case sensitivity here.
- When you define the connection, this cross references your local CP name as the netname. You will be using the APPC protocol through access method VTAM.
- Install the definitions and acquire the connection.

You are now ready to test your connection.

Select the menu option **IBM Personal Communications->Administrative and PD aids->SNA node operations** to start the process (see Figure 191)

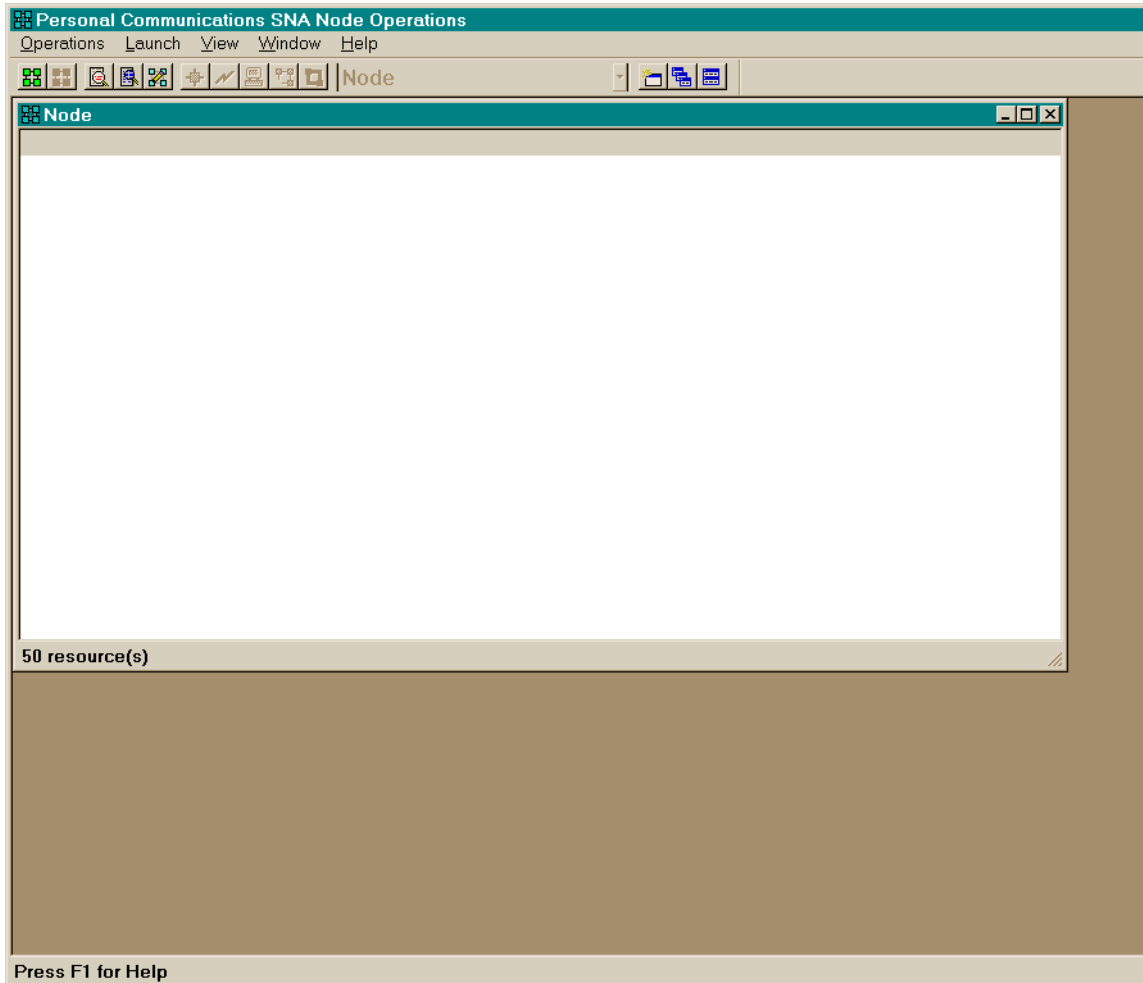


Figure 191. Starting the link

Select the menu option **Operations->Start Node** and choose the file name in which you previously saved your configuration. Figure 192 shows the node definition.

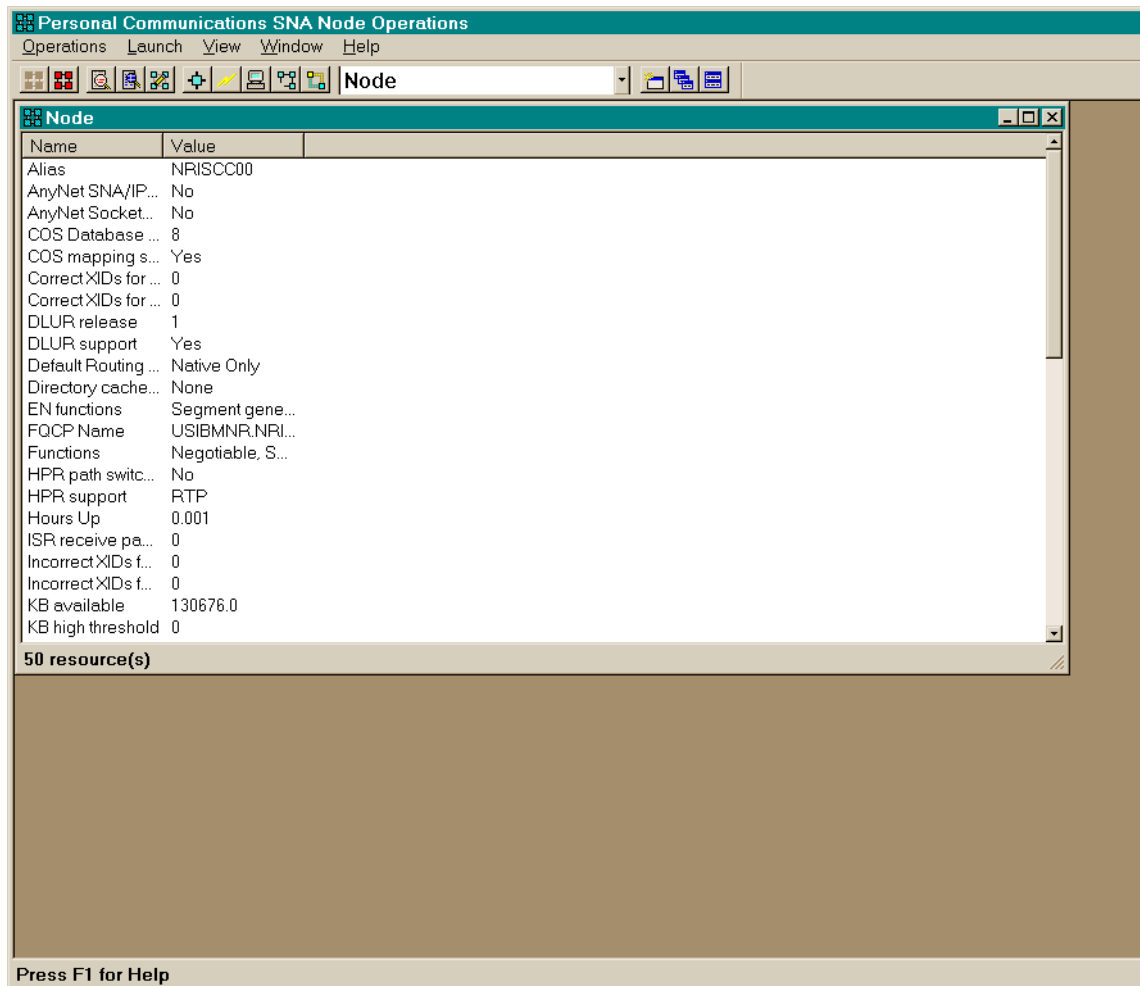


Figure 192. Start the node

Finally you need to select **Operations->CNOS Initialize**. You need to fill in the your local CP name (in our examples JANESXID), the fully qualified LU we used for CICS earlier (not its CP name), (so SNANETWK.CICSAPPL in our case) and the mode name you specified (LU62APPX), as in Figure 193. Your APPC session should now be established.

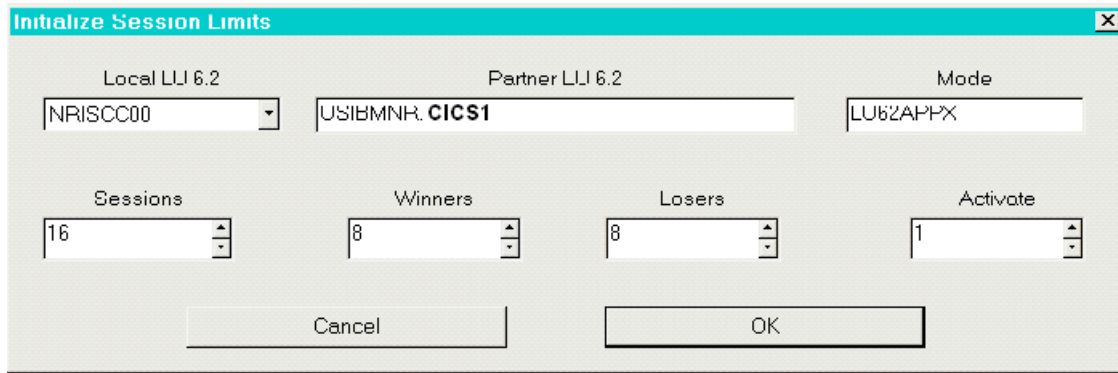


Figure 193. Initialize the session

13.5.3 CICS security

A userid and password valid for the target CICS system must be defined and available for use when configuring system access (see 16.3, “Security” on page 352 for a Windows NT-based discussion).

13.5.4 Set up VisualAge Generator Host Services

Follow the installation manuals to install and configure the mainframe software listed in 12.3, “CICS/ESA Web Transactions” on page 254.

The CPML transaction must be altered or an alternate transaction definition created to support the VisualAge Generator requirements for the CICS ECI transaction (TWASIZE 1024). This was discussed in 13.4.6, “Add VisualAge Generator runtime and debug transactions” on page 277.

13.5.5 Communications configuration

Figure 168 contains an example of CSOGW.properties entries for communication to a CICS/ESA server.

```
hptGateway.propertiesRefreshInterval=2

application.*=cicsesa

serverLinkage.cicsesa.commtype=cicseci
serverLinkage.cicsesa.contable=csoE037
serverLinkage.cicsesa.location=NRACICS1
serverLinkage.cicsesa.serverid=WEBS
serverLinkage.cicsesa.javaProperty=vgmt.beans
```

Figure 194. CSOGW.properties file entries: Windows NT system

Additional detail on CSOGW.properties file entries is available in 13.2, “Web Transaction gateway interface configuration (csogw.properties)” on page 263.

See 14.3.3, “Customization for CICS/ESA access” on page 321 for additional CICS Client connectivity considerations.

Chapter 14. WebSphere Application Server setup

This section details the installation and configuration of the various software products required for the implementation of an IBM WebSphere Application Server environment on Windows NT.

14.1 Installed software base

The following software was installed prior to beginning the IBM WebSphere Application Server installation and configuration task.

Web server

Install Web server software that is supported by IBM WebSphere Application Server.

We used both the Apache and IBM HTTP Web servers during the residency.

Java Development Kit

- Install the software.
- Ensure that you are using the correct version of the JDK (or IBM Development Kit, as the case might be). To verify that you have the right JDK installed on Windows NT, execute this command in a command prompt window:

```
java -fullversion
```

The output should be:

```
java full version "JDK 1.1.7 IBM build n117p-19990618 (JIT enabled:
ibmjitc)"
```

- Make sure the PATH and JAVA_HOME environment variables are set properly and that they do not point to a JDK other than the required JDK.

Add the Java /bin directory to the PATH environment variable.

Create an environment variable called JAVA_HOME and set it to the JDK directory name. Note that problems have been encountered when the JAVA_HOME value contains a space in the directory name for the JDK.

DB2 UDB

- Install database software and apply any required fixpacks.

Note: This is only required on the IBM WebSphere Application Server platform if you are using the Advanced or Enterprise Edition of the IBM WebSphere Application Server (or have other Java servlets that directly access DB2).

VisualAge for C++

- The software is defined as a prerequisite for VisualAge Generator Server and Common Services (we did not test if C++ was actually required to run the Gateway Servlet).

VisualAge Generator Server and Common Services

- Install the software.

14.2 IBM WebSphere Application Server for Windows NT

Make sure you have completed the steps discussed in 13.3, “Windows NT Web Transactions” on page 268 and rebooted before attempting this step.

14.2.1 Install IBM WebSphere Application Server software

We chose to install IBM WebSphere Application Server Advanced Edition for Windows NT, although the steps below should also be functionally complete for both the Standard and Enterprise Editions of IBM WebSphere Application Server.

- If you choose a custom install, ask for a default application server and a default application.
- Use a demo key ring file if you are just getting started with WebSphere. Make sure the userid and password you specify with this is a valid one for Windows NT. If you need to change the defined password or userid later, you need to change it in two places:
 1. In the file `x:\websphere\applicationserver\properties\sas.server.props`
There are two userid entries in this file.
 2. In the NT services entry for IBM WebSphere Application Server
- Specify the location of your JDK and choose to plug in to the installed Web server.
- There is a panel which asks for the database product configuration you wish to use for Enterprise Java Bean support.

Note 1: The full DB2 database environment may only be required when using the Advanced or Enterprise Edition of the IBM WebSphere Application Server.

We used DB2. We entered the database administrator userid and password used when DB2 was installed. You can identify the DB2 administrator using the Windows NT User Manager.

Note 2: If you are reinstalling, make sure you have dropped any EJB tables from the database name you specify here before proceeding.

Note 3: If you upgraded an installed WebSphere Application Server V3.0 system to 3.02, you may have to add the /lib/jsp10.jar and /ibmwebas.jar files to the classpath defined in the /bin/admin.config file.

- Follow the post installation instructions in the readme file, these should mention to:
 - Create the "was" database (or whichever database name you specified during installation)
 - In the DB2 control center, select the database, right-click for the context menu and select configure. On the performance tab, scroll to the application heap size and change it to 256.
- Reboot — You are now ready to bring up IBM WebSphere Application Server.

14.2.2 Startup WebSphere Application Server

To start the IBM WebSphere Application Server:

1. Start the IBM WS AdminServer using the Windows NT Services dialog.
2. Start the Administrator's Console using the program start menu.

The console is shown in Figure 195.



Figure 195. WebSphere Administration Console

14.2.3 Configure a new Application Server

Once the Administration Console has started, configure a new Application Server:

- Click to the **Tasks** tab, expand **Configuration**, select **Configure an application server**, and click the green button to start.
- Deselect the Enterprise Beans and select the Web Applications, then click **Next**, as shown in Figure 196 below.

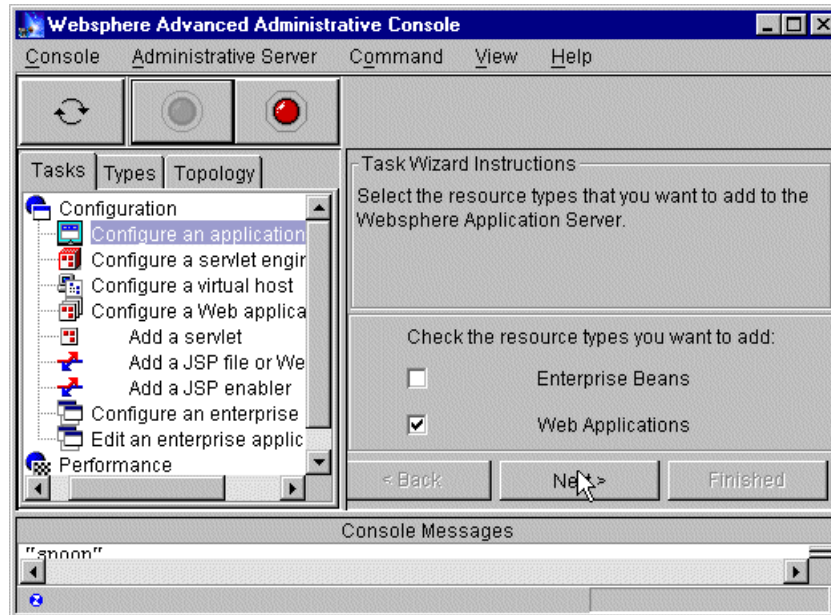


Figure 196. Application Server configuration: starting the task

Now we need to name our IBM WebSphere Application Server and potentially specify the VisualAge Generator Web Transaction support files.

- Enter a name for the application server. (See Figure 197.)

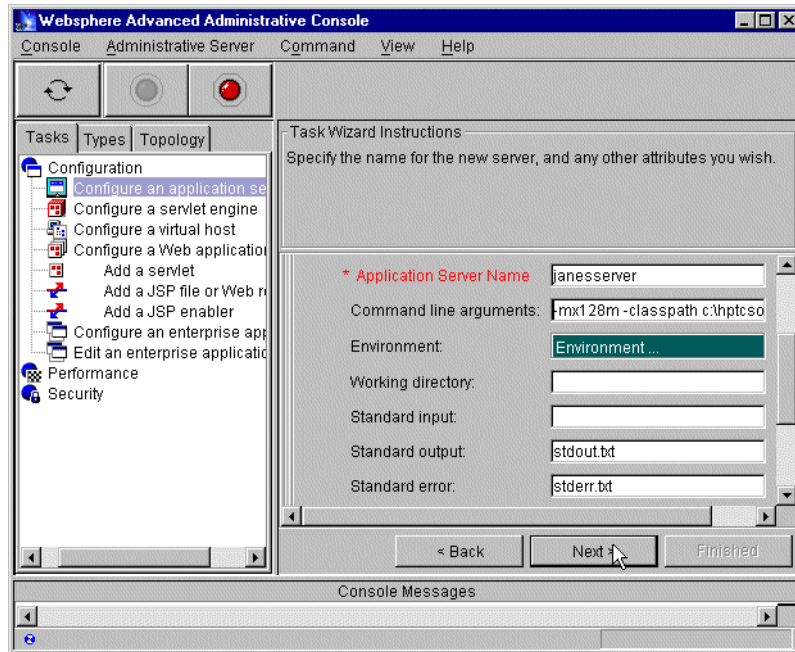


Figure 197. Application Server configuration: defining parameters — part 1

Note: You may need to add .JAR file references to the command line invocation shown in Figure 197.

- **View 1:** It is VERY important that you add the .JAR file references to the command line options shown in Figure 197 and that you do NOT use the app classpath shown later in Figure 203.
- **View 2:** Systems that have been set up using the app classpath shown in Figure 203 work fine when they call Web Transaction programs generated for Window NT.

There seem to be restrictions associated to calling Web Transaction programs in CICS that are only satisfied when the .JAR files are referenced as part of the command line arguments for the application server.

- **View 3:** When you use the -classpath technique, you must either include the directory where the generated JavaBean class files can be found, or copy these files to a directory already known to WebSphere Application Server (/classes). The alternate place for class path definitions is not *heard* when the -classpath option is used.

As required (such as when the GatewayServlet will use the Java Native Interface (JNI) to invoke the Web Transaction, such as when CICS is the destination for the Web Transaction program), we added the following command line arguments:

```
-classpath x:\vgservw\hptGateway.jar;x:\hptcsow\hpt.jar
```

Beware of spaces in the directory list; if you have any, the app server will not start. Try enclosing the string in double quotes.

When the name and optional command line definition are complete, click **Next**.

Note: We have heard that, to improve performance, the a value such as `-ms64m` should be specified whenever you provide arguments for a Java Virtual Machine (JVM). One suggested configuration approach has an `mx` value of a quarter physical memory and an `ms` value of half of `mx` on NT, AIX and Solaris" (except for Solaris on Sun JDK).

For example, you could modify the command line arguments shown above to read as follows (note that the complete command line is shown):

```
-mx128m -ms64m -classpath x:\vgservw\hptGateway.jar;x:\hptcsow\hpt.jar
```

The `-mx128m` entry was already on the command line before we started to add the `-classpath` addition.

- Choose **not** to autostart the server after creation, as there are some things we want to do first, and click **Next**, as shown in Figure 198.

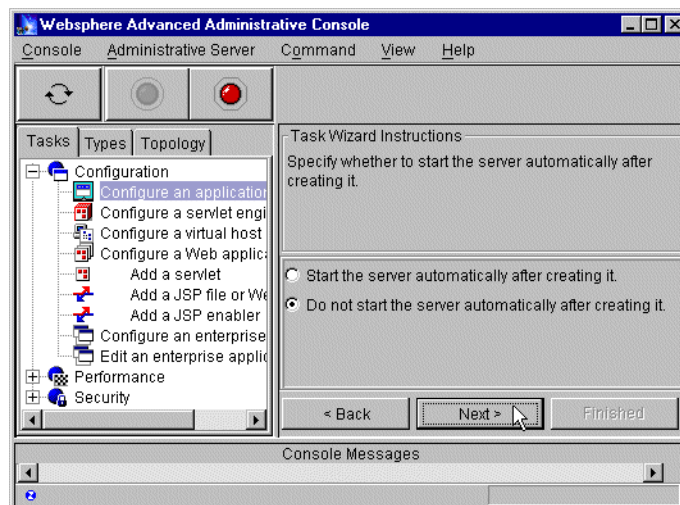


Figure 198. Application Server configuration: defining parameters — part 2

- Choose a node; there should only be one option, as shown in Figure 199; and click **Next**.

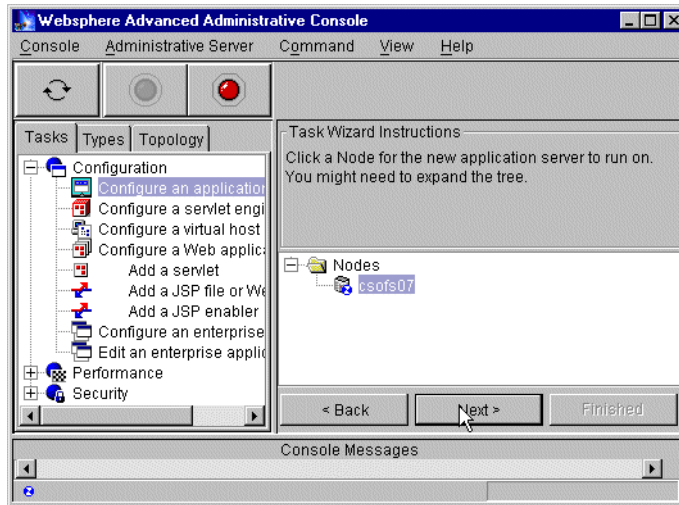


Figure 199. Application Server configuration: choosing a node

- Choose a virtual host; there should only be one choice, as in Figure 200; and click **Next**.

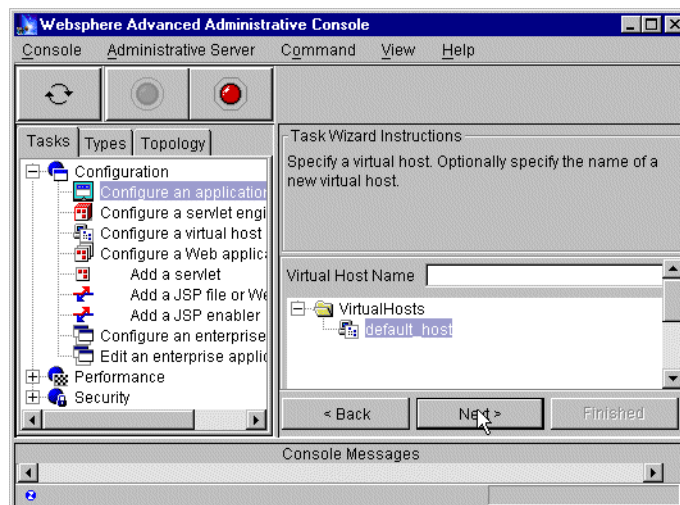


Figure 200. Application Server configuration: virtual host

- Leave the servlet engine to default, as in Figure 201, and click **Next**.

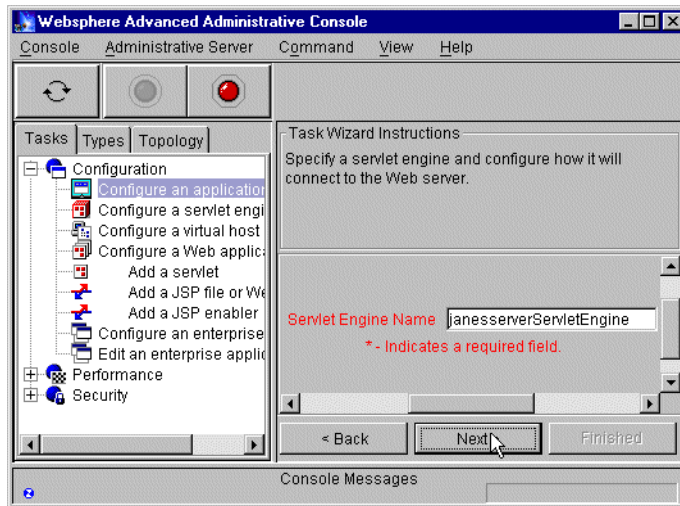


Figure 201. Application Server configuration: servlet engine

- Leave the app name and Web path to default, as in Figure 202, and select the **Advanced** tab.

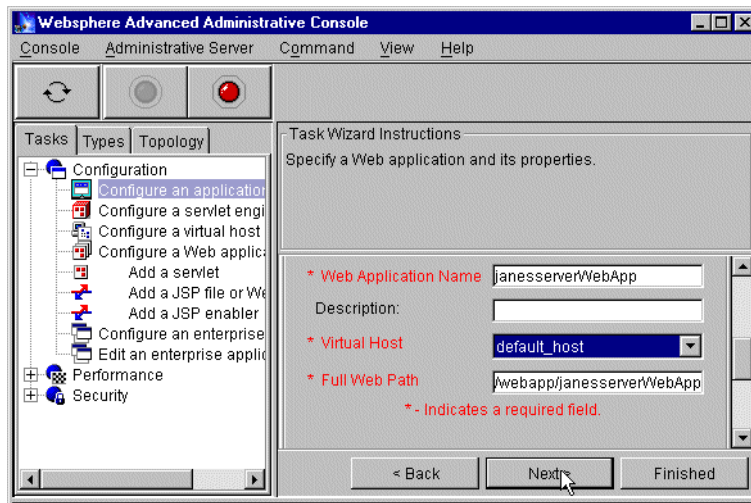


Figure 202. Application Server configuration: defining Web application

- If you wish, you can accept the application document root and classpath default settings, as in Figure 203, and click on the **Next** push button.

The document root controls where you put your Java Server Pages. The document root could be a different value, it all depends on where you want these files to be stored. You might want to store these files outside the WebSphere Application Server directory structure.

The classpath represents other directories for servlets and beans. But, when the -classpath option is defined, as suggested in Figure 197 on page 305, the classpath settings for the Web application are not used.

JavaBeans generated by VisualAge Generator can either be placed in the WebSphere Application Server \classes directory (see 16.1, “Deploy generated code” on page 341) or in a directory added to the -classpath option definition.

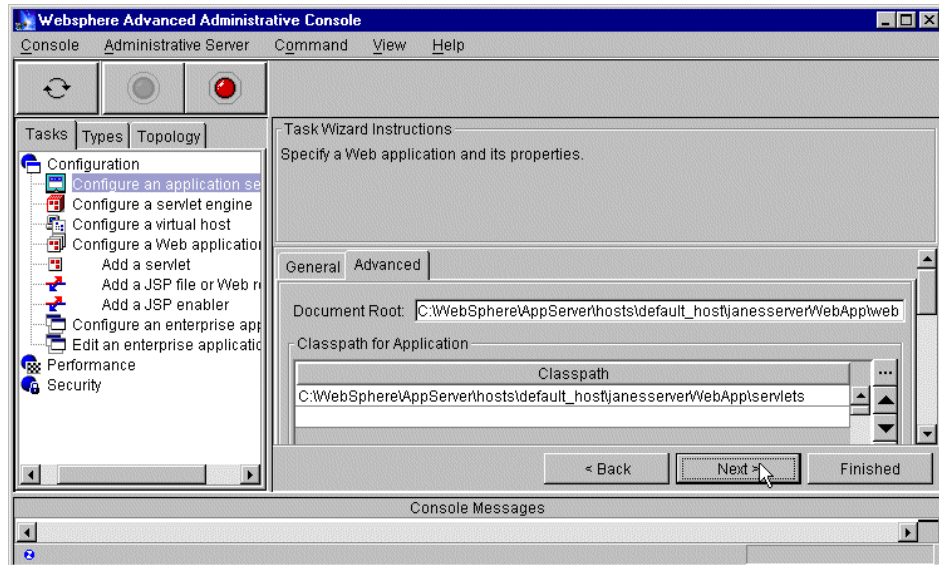


Figure 203. Application Server configuration: Web application properties

- Leave the **enable file servlet** checked and the **serve servlets by classname** unchecked. Make sure **JSP 1.0** is enabled, as in Figure 204.
The file servlet allows GIFs placed with the JSPs to be loaded, which is what we will be doing with the JSPs supplied with VisualAge Generator.
- Click on **Finished** to trigger the processing required to add the new application server definition to the environment.

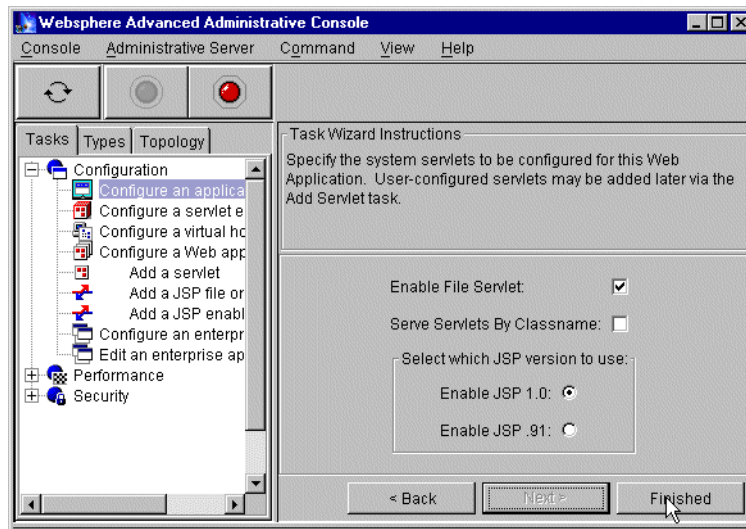


Figure 204. Application Server configuration: system servlets

- Go to the **Topology** tab. Select the application server you have just built and expand the tree, select the WebApp you have made, and click to the **Advanced** tab. Type `/ErrorReporter` into the **Error Page** and click **Apply**, as shown in Figure 205.

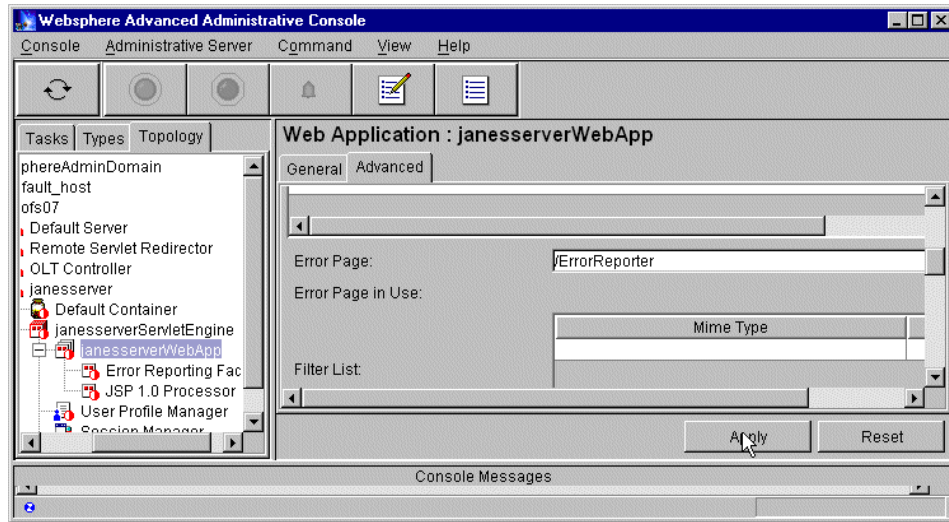


Figure 205. Application Server configuration: defining error page

If you do not complete this step, your application server will not start.

14.2.4 Define VisualAge Generator Gateway Servlet

Use the Topology page to define the Gateway Servlet to IBM WebSphere Application Server.

Select the Web App you just defined, and using mouse button 2, select **Create... > Servlet** (see Figure 206).

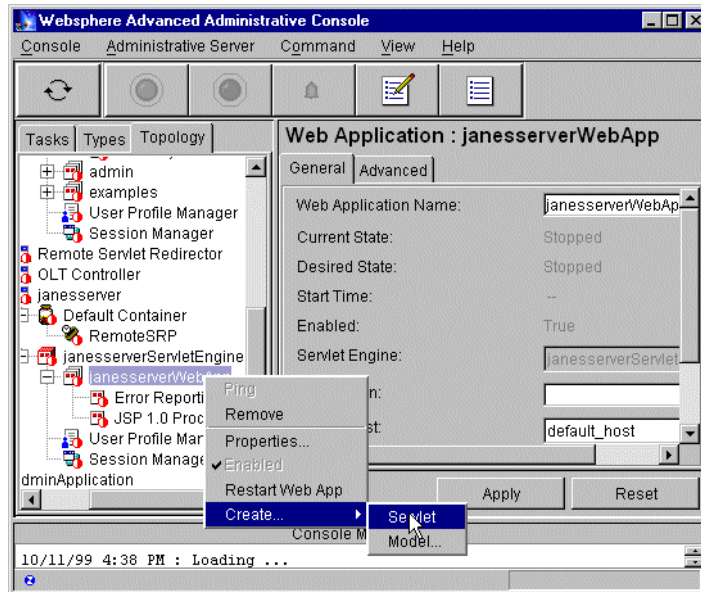


Figure 206. Application Server configuration: defining Gateway Servlet

When you click to create, Figure 207 shows the window which is launched.

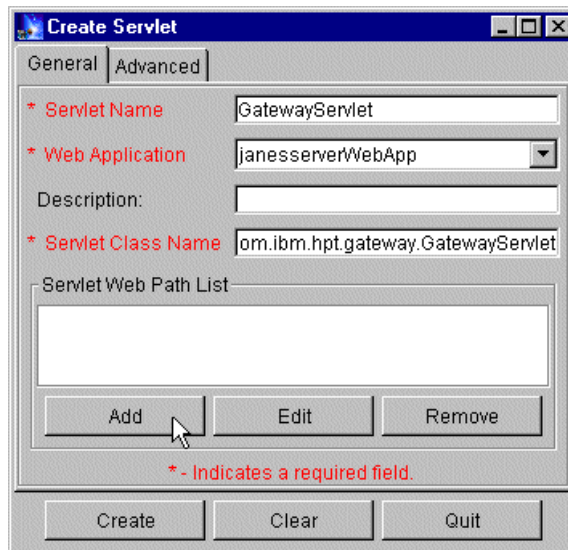


Figure 207. Gateway Servlet properties — part 1

- Name the servlet, for example, *GatewayServlet*.
- Specify the Java class of the servlet:
`com.ibm.hpt.gateway.GatewayServlet`
- Click the Add push button under the Servlet Web Path List.
- Put the cursor in the Servlet Path field and fill out the end of the URI (Universal Resource Indicator) with a / followed by a meaningful name, such as *GatewayServlet* (see Figure 208), and then click on OK.

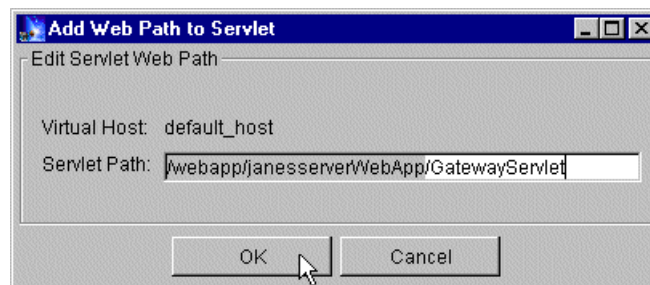


Figure 208. Gateway Servlet properties — part 2

This should give a URI of the form *default_host/webapp/yourserverWebApp/GatewayServlet*. This URI is what would be typed on a browser location field to invoke the servlet, where "default_host" would be replaced by the protocol, actual host name and optionally port, for example, *http://localhost*. The complete URL (protocol+host name+optional port+URI) would be:

`http://hostname/webapp/yourserverWebApp/GatewayServlet`

- Now we can specify initialization parameters for the servlet; choose the **Advanced** tab. This is shown in Figure 209 below.

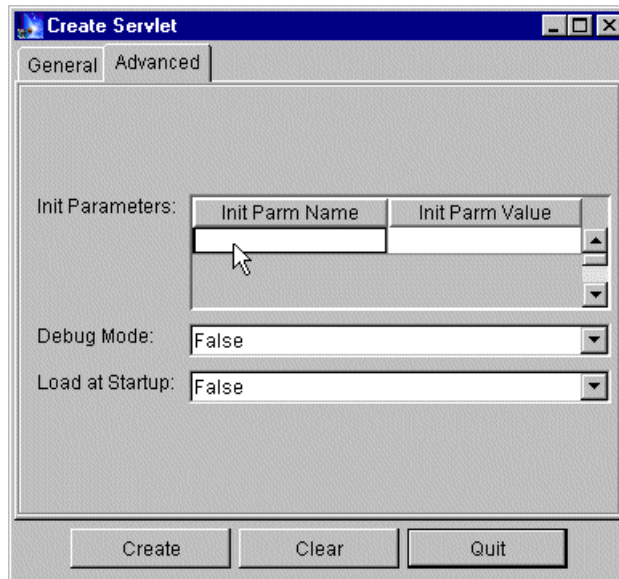


Figure 209. Gateway Servlet properties — part 3

Valid initialization parameters and their values are listed in Table 11. Beware of case sensitivity in parameter names and values. Mistakes in these settings will trigger Gatewayservlet null pointer exceptions.

Table 11. Initialization parameters for the Gateway servlet

InitParameter Name / Value	Notes
hptGatewayProperties	This parameter allows you to include all the parameters described in this table in a plain text file and just specify one initialization parameter in the GatewayServlet itself. The format inside the text file is param=value.
filename	
hptLogonPage	Without this parameter no logon will be requested. There is no point setting this parameter in a native NT, AIX, Solaris or HP/UX environment, it does not do any validation with the userid or password that the user enters.
/Vagen1LogonPage.jsp	
hptEntryPage	The menu. Either this or hptEntryApp must be specified.
/Vagen1EntryPage.jsp	
hptEntryApp	The name must correspond exactly to a definition in the VisualAge Generator server environment, for example a CICS definition or DLL name. There must also be an entry for it in the CSOGW.PROPERTIES file. This Web Transaction will be invoked at startup, after logon (if logon is enabled), and when any Web Transaction ends by EZECLDS.
VAGen Web transaction name	
hptErrorPage	This is displayed when an error occurs.
/Vagen1ErrorPage.jsp	
hptErrorLog	Log file for Gateway Servlet.
/Vagen1Gateway.log	
hptLinkageProperties	The control file for communications between the Gateway Servlet and the Web Transaction. The file name must use forward slashes(/) such as x:/VGServw/CSOGW.properties.
filename	
hptIDManagerHost	The host name where the session ID Manager process runs (the session ID manager is discussed in 2.3.3, "Session ID Manager (SIDM)" on page 46). Localhost is used by default if you omit this entry.
<host name>	
hptDateMask	This mask gives a consistent internal store for date fields associated with UI records. It is very important that this matches with your host settings.
dd?mm?yyyy	

Note: An example Gateway Servlet properties file shipped with VisualAge Generator (GW.properties) includes the entries hptRuntimeProperties and hptErrorPackage. These are not used by the Gateway Servlet. These entries were part of an early design, but they were not removed from the example.

We chose to use the `hptGatewayProperties` initialization parameter to define a file that would contain all other parameters (see Figure 210).

Init Parameters:	
Init Parm Name	Init Parm Value
hptGatewayProperties	e:\wgenout\WGASGateway.properties

Init Parameters in use:	
Init Parm Name	Init Parm Value
hptGatewayProperties	e:\wgenout\WGASGateway.properties

Figure 210. Gateway Servlet properties — using an external parameter file

- Click on the Create push button to save the Gateway Servlet definition when you are done defining the parameters.

The contents of the referenced file are shown in Figure 211.

```
xhptLogonPage=/Vagen1LogonPage.jsp
hptEntryPage=/Vagen1EntryPage.jsp
xhptEntryApp=CONVMOD
hptErrorPage=/Vagen1ErrorPage.jsp
hptErrorLog=e:/VgenOut/Vagen_BT_Gateway.log
hptLinkageProperties=e:/vs/CSOGW.PROPERTIES
hptIDManagerHost=cork
hptDateMask=dd/mm/yyyy
```

Figure 211. Gateway Servlet properties — external parameter file contents

The only way to reset the Gateway Servlet properties is to stop and start the WebSphere Application Server.

Some entries are prefixed with an "x" so that they are ignored by the Gateway Servlet (the altered property name does not mean anything to the Gateway Servlet). The `hptLogonPage` property is excluded in the scenario shown in Figure 211 because we are using a Web Transaction runtime platform of Windows NT, and there is no security processing during Web Transaction initiation (so why ask for a userid and password?). See 16.3, "Security" on page 352 for more details on logon page processing.

14.2.5 Customize JSPs (as required)

If required, edit the VisualAge Generator JSPs to correct invalid syntax:

- Change any "imports =" directives to "import =" (drop the s).
- Remove any create="false" clauses inside useBean tags.

The create clause was only found in the Vagen1EntryPage.jsp file.

Notes:

We had to change both the JSP files provided by the product and those later generated by VisualAge Generator for a given Web Transaction.

These changes were required with the VisualAge Generator V4 code we had installed during the residency. They may not be required if you have applied fixes to your development system.

The CSOERRORUIR.jsp file provided by VisualAge Generator is used to display runtime errors. The name of this error JSP file cannot be modified.

The Vagen1EntryPage.jsp file is customized later (see 16.1.3, "Invoking Web Transaction from default entry point JSP" on page 342) to add definitions for Web Transactions that will be directly accessible from the default entry page.

14.2.6 Deploy JSPs and GIFs

Deploy the VisualAge Generator JSPs and GIFs. Copy the *.JSP and *.GIF files from the VisualAge Generator Server installation directory (x:\VGSERVW) to the path you specified as the Application Server Document Root earlier (see Figure 203 on page 309).

You may need to define the directory when you do this. We used x:\WEBSHERE\APPSERVER\HOSTS\DEFAULT_HOST\janesserverWebApp\WEB*.*.

If your Gatewayservlet URL included any extra slashes, for example: "default_host/webapp/yourserverWebApp/xxx/GatewayServlet" rather than "default_host/webapp/yourserverWebApp/GatewayServlet", you will need to make a subdirectory of the x:\WEBSHERE\APPSERVER\HOSTS\DEFAULT_HOST\janesserverWebApp\WEB directory called xxx and put the GIFs in there instead. This is so the File Loader servlet can find them.

As an alternative, you could add an alias entry to the httpd.conf file:

```
# Alias to locate the VAGen GIFS
Alias /vgengifs/ "e:/vs/"
```

And then you would need to adjust the VisualAge Generator provided JSPs to reference this logical directory:

```
<BODY background="/vgengifs/vawcg-wp.gif">
<img SRC="/vgengifs/visage.gif" >
<FONT SIZE=6>VisualAge Generator System Entry</FONT>
<img SRC="/vgengifs/visage.gif" >
```

14.2.7 Configure the vgj.properties file

This file is for VisualAge Generator Java generated code, so it applies to the interface bean and data bean, especially since the former will contain Web-side edit routine code.

The file contains settings for:

- Default NLS code
- Decimal separator character
- Gregorian and Julian date masks (EZEDTELC and EZEDAYLC format)
- Tracing of VisualAge Generator code

The vgj.properties file (it may be shown in upper case on some installs) is found in the directory where VisualAge Generator Common Services was installed.

14.2.8 Set up VisualAge Generator session ID manager

The following line in a bat file will start the session ID manager:

```
java com.ibm.hpt.gateway.SessionIDManager
```

See 2.3.3, "Session ID Manager (SIDM)" on page 46 for details on SIDM processing.

14.2.9 Start application server

You can now start your application server. Still on the **Topology** tab, select your application server, and click the green button to start it, as shown in Figure 212 below. This process may take some time.

Note: You may have to start your Web server first, if it does not autostart. See 16.2.1, "System Startup" on page 344 for additional details.



Figure 212. Starting WebSphere Application Server

Invoke the GatewayServlet to check whether it is operational. Enter a URL such as:

`http://localhost/webapp/yourserverWebApp/GatewayServlet`

Here, your serverWebApp is the value you entered for the Application Server name (see Figure 197 on page 305).

14.3 Adding CICS support

If the generated Web Transaction programs will be implemented in CICS, you will need to install and configure the CICS Transaction Gateway software which allows the Gateway Servlet to communicate with the target CICS environment.

14.3.1 CICS Transaction Gateway

Install the CICS Transaction Gateway software. Configuration is easier if you install into an alternate directory (such as `x:\ctg`), rather than the default directory (`x:\Program Files\IBM\CICS Transaction Gateway`). The spaces in the default directory name cause problems when trying to add the JAR files to the WebSphere Application Server classpath.

Add the CICS Transaction Gateway classes support classes to IBM WebSphere Application Server, by adding the following to the -classpath entry for the Application Server:

```
;x:\CTG\CLASSES\ctgclient.jar;x:\CTG\CLASSES\ctgserver.jar
```

The initial -classpath entry definition is shown in Figure 197 on page 305.

Beware of spaces in the directory name; if the defined directories have a blank then the Application Server will not start. If your directory name has spaces, either uninstall/reinstall or copy the JAR files to another directory and add the alternate directory to the classpath.

It is very important you add the JAR files to WebSphere as shown above, because it causes WebSphere to load the classes in a way that permits the use of the Java Native Interface (JNI) in the code. JNI is used to access the CICS Transaction Gateway.

Note: All that was required to add CICS support when using WebSphere Application Server V3.0 is documented above. When we used (or converted to) WebSphere Application Server V3.02, and tried to use a target runtime environment of CICS, we had problems. The logs identified a missing DLL (CTGJNI.DLL). This DLL is in the x:\ctg\bin directory, which is in the PATH defined for the Windows NT system, but the DLL is not found when using WebSphere Application Server V3.02. To resolve this we modified the WebSphere Application Server admin.config file (x:\ws\as\bin) and added x:\ctg\bin to the Nanny path as shown below:

```
com.ibm.ejs.sm.util.process.Nanny.path=e:\ws\as\bin;e:\ws\as\bin;  
d:\JDK11-1.7\bin;e:\ctg\bin
```

14.3.2 Customization for TX Series (CICS NT) access

Customize the initialization file to identify the target CICS system. The initialization file may be named CICSCLI.INI or CTG.INI, depending on your code base. We focused on the following settings in the initialization file:

- NetName — the TCP/IP address of the target CICS server machine
- CICS server name — referenced in the csogw.properties file (see 13.3.3, “Communications configuration” on page 270)
- UpperCaseSecurity=N — Security control setting for password validation

The settings for our CICS NT system are shown in Figure 213.


```

SECTION SERVER = CICSTCP
DESCRIPTION=TCP/IP Server
UPPERCASESECURITY=N
USENPI=N
PROTOCOL=TCPIP
NETNAME=ireland.almaden.ibm.com
PORT=0
CONNECTTIMEOUT=0
TCPKEEPALIVE=N
ENDSECTION

```

Figure 213. CICS client settings for Windows NT-based CICS system

The port setting must match the listener definition (see 13.4.4, “Add CICS system listeners” on page 275). Here a port value of 0 will trigger the use of the default port value (1435).

If desired, you can edit the properties of the *Start CICS client* Windows NT START menu option so the command string reads "CICSCLI /S=x /N /W", where x is the name you gave the CICS server in the initialization file.

14.3.3 Customization for CICS/ESA access

Customize the initialization file to identify the target CICS system. The customization for CICS/ESA is similar to the CICS NT customization.

The settings for our CICS/ESA system are shown in Figure 214.

```

Server = CICSSNA
; The named server must match the name used in the serverLinkage.serverid entry.

Protocol = SNA
; The CICSCLI settings below referenced the PCOMM host session settings
; These settings are explained in the section on configuring PCOMM.

Netname = SNANETWK.CICSAPPL
;fully qualified LU name for CICS

LocalLUName = JANESXID
;local CP name

ModeName = LU62APPX
;mode name defined in VTAM

```

Figure 214. CICS client settings for MVS-based CICS/ESA system

The configuration in Figure 214 tells the CICS client to use the APPC protocol to communicate with the target CICS/ESA region. In order to configure APPC we had to install and configure Personal Communications (see 13.5.1, "Install the PCOMM software" on page 280).

As part of this operation, we had to request a Control Point (CP) name for our Web server machine on which the CICS client was installed. We also requested the destination address (MAC) address of the remote controller for VTAM, along with the partner LU name of the CICS region and the owning control point in VTAM.

14.3.4 CICSTERM behavior and signon capable terminals

One of the easiest ways to validate connectivity to the target CICS system is to use a CICS terminal session to test access.

We used a mix of CICS environments, one of the last being TX Series V4.3. In doing so, we had some problems connecting to the Windows NT based system until we read the README.TXT available with V3.1 of CICS Transaction Gateway for Windows NT. A portion of this file is repeated below.

CICSTERM now attempts to install a signon capable terminal by default. Use the option '-a' to request the default CICSTERM behavior as in releases prior to v3.1.0.

CICS Servers require APAR fixes to support the terminal signon capability function available in this release:

CICS/ESA 4.1	PQ30167
CICS TS for OS/390 V1.2 and V1.3	PQ30168
CICS/VSE 2.3	PQ30169
CICS TS for VSE/ESA V1.1	PQ30170
TX Series	IY03691

CICS TS for OS/390 v1.3 Servers

If the server does not have the required APAR applied and the '-a' option is not specified on CICSTERM, the installed terminal will give unpredictable results.

TX Series Servers

If the server does not have the required APAR applied and the '-a' option is not specified on CICSTERM, the client will display the message:

```
CCL7053E Errors found while communicating with server
```

and the message:

```
CCL3105 Inbound CICS datastream error (CTIN, 4, 0)
```

will be written to CICSCLI.LOG.

On the server, the message:

```
ERZ042004E/0112: An invalid request was received from client
```

will be written to CSMT.out and console.msg will include:

```
ERZ014016E/0036: Transaction CTIN Abend A42B
```

Security on OS/390 Servers

Security checking done in the server for transactions started at a signon capable terminal installed by a client application does not depend on what is specified by the ATTACHSEC option for the connection representing the client.

Instead, security checking depends on whether the user signs on while using the terminal.

If the user does not sign on, the client installed terminal is associated with the default user defined for the server in the SIT. When a transaction is run, the security checks are carried out against this default user. A check is also done against the userid associated with the connection to see whether the client itself has authority to access the resource.

When a user does sign on, the terminal is associated with the userid just authenticated. For transactions attempting to access resources, security checking is done against the userid associated with the connection and the signed-on user's userid.

It is recommended that the Usedfltuser parameter on the server connection definition be set to Yes if using signon capable terminals, and to No if using signon incapable terminals.

14.4 VisualAge for Java WebSphere test environment

There are two basic ways of testing a VisualAge Generator Web Transaction:

Source Testing—The 4GL program source can be directly tested using the test facility to emulate VisualAge Generator implementation of runtime processing.

Web server and UI Record processing is emulated by the test facility and direct interaction with a Web browser (see 2.1.5, “Testing” on page 27).

Runtime Testing—Runtime testing can be enhanced through the use of the VisualAge for Java WebSphere test environment.

Customization of the generated JSPs is expected. Testing these changes typically requires a full runtime environment. (Note: test facility support for invocation of a Web Transaction from the runtime Gateway Servlet is a requirement).

To better support testing of the customized JSP, you can configure the Gateway Servlet in the VisualAge for Java WebSphere test environment.

Version notes:

- You can get this configuration to work with the GA version of VisualAge for Java. Initial packaging of VisualAge Generator V4 included a supported beta version of VisualAge for Java, while the GA version is now shipped with VisualAge Generator (and available to those who received the beta version package).
- Gateway Servlet session data management issues will be resolved in VisualAge for Java V3.02.

14.4.1 Setup

1. Install VisualAge Generator Server and VisualAge Generator Common Services on the VisualAge for Java developer machine. VisualAge Generator Common Services will have already been installed if the VisualAge Generator Developer on Java has been installed.

Note: You can run the Gateway Servlet on a workstation where the Developer on Java feature has not been added to the workspace. The Gateway Servlet will invoke the generated version of the Web Transaction.

2. Ensure that the required VisualAge for Java components have been installed.

If one of the required components has not been installed update the installation (using the VisualAge for Java installation CD setup command) to add the component:

- a. The **EJB/JSP Development Environment** and **Servlet Builder** components must be installed to provide access to the **IBM WebSphere Test Environment** feature.
- b. The **Transactions Access Builder** component must be installed only if CICS will be used as the runtime platform for Web Transactions. This will provide access to the **IBM Common Connector Framework** and **CICS Connector** features.

If you do not plan to invoke Web Transactions that run in a CICS system, this component can be skipped. The Gateway Servlet references to CICS connector classes will not be resolved, but you can still use TCP/IP to invoke Web Transactions called by the Gateway Servlet in the WebSphere Test Environment.

3. Ensure that the required VisualAge for Java features have been added to the workspace.

If one of the required features has not been added, use the VisualAge for Java Quick Start (F2 or **File->Quick Start**) to select the features that must be added to the workspace.

- a. The **IBM WebSphere Test Environment 3.0** and **IBM JSP Execution Monitor 1.1** features must be added to the workspace. This will also trigger the installation of other required features.
- b. The **IBM Common Connector Framework** and **CICS Connector** features must be added only if CICS will be used as the runtime platform for Web Transactions.

Note: The CICS software you use depends on the install order:

- During VisualAge for Java installation, CICS software can be installed in the \IBM Connectors directory. If you had CICS Client or CICS Transaction Gateway software installed before installing VisualAge for Java, you may have blocked this installation.
- To use the pre-existing CICS software, which may be from a different version (different DLL names) you must import and load the matching CTGclient.jar and CTGserver.jar files into the VisualAge for Java workspace.

4. If using CICS as the runtime platform for Web Transactions, the CICS Transaction Gateway support that is installed must be configured for the target CICS system (see 13.4, “CICS for NT Web Transactions” on page 270).
5. Import the Gateway Servlet runtime classes from the hptGateway.jar file (found in the VisualAge Generator Server installation directory) into the VisualAge for Java workspace.

We used a project named **z WebSphere Test Support** to hold the Gateway Servlet and other classes that will also be imported.

If the **IBM Common Connector Framework** and **CICS Connector** features have not been loaded (because CICS will not be used as the runtime platform for Web Transactions), there will be a problem listed for the Gateway Servlet in the WebSphere Test Environment, as shown in Figure 215.

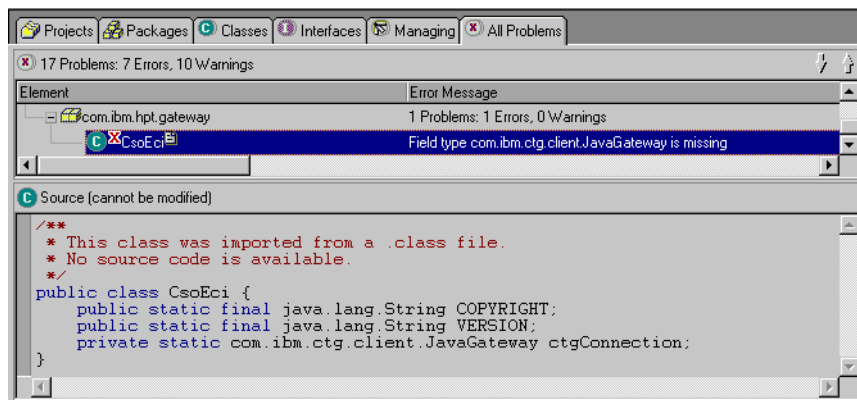


Figure 215. Gateway Servlet problems when CICS support is not loaded

If the CICS support required by the Gateway Servlet has been loaded, there will still be problems indicated in the workbench in the WebSphere Test Environment, as shown in Figure 216.

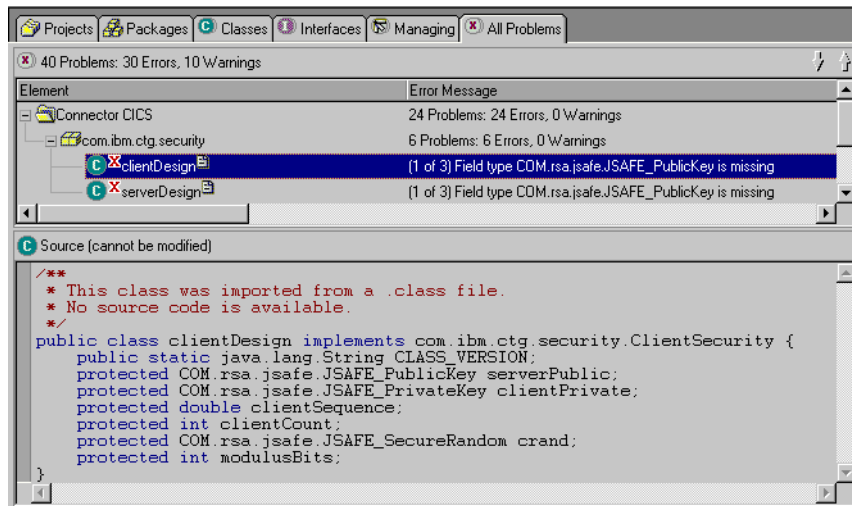


Figure 216. Gateway Servlet problems when CICS support is loaded

These references to classes do not impact the testing of the JSPs generated by VisualAge Generator in the WebSphere Test Environment. You only need to load the CICS support if your target runtime platform for the Web Transaction program is CICS.

6. Import the hpt.jar, if required.

You do not have to import the hpt.jar file if the VisualAge Generator Developer on Java feature has been loaded. The same code is loaded from the Developer on Java install base as part of the feature.

14.4.2 Configure WebSphere Test Environment

1. Update the SERunner.properties file (found in the c:\ibmvjava\ide\project_resources\ibm websphere test environment directory) to adjust the port and document root settings.

httpPort—If you want to use both the WebSphere Test Environment and the VisualAge Generator test facility, you must change the port used by the SERunner class to avoid conflicts.

docRoot—By changing the document root you can, if you wish, point to the same directory used by the runtime implementation of the Gateway Servlet (see Figure 203 on page 309).

The SERunner.properties file we used included these settings:

```
httpPort=8181
docRoot=v:\\wsas
serverRootd:\\j3\\ide\\project_resources\\IBM WebSphere Test Environment
```

A similar port change may be required in the default.servlet_engine file (found in the c:\ibmvjava\ide\project_resources\ibm websphere test environment directory). The servlet engine transport definition includes a port argument.

Note: As an alternative, you can change the port used by the VisualAge Generator test facility. See the Test Web page in the VisualAge Generator options dialog.

2. Update the com.ibm.servlet.SERunner class properties (part of IBM WebSphere Test Environment 3.0 project) by adding all the projects in the VisualAge for Java workspace to the project class path.

14.4.3 Configure GatewayServlet

1. Update the default_app.webapp file to add the Gateway Servlet definition and its initialization parameters. The file can be found in this directory:

```
c:\ibmvjava\ide\project_resources\ibm websphere test environment
\hosts\default_host\default_app\servlets
```

The Gateway Servlet definition in the WebSphere Test Environment is shown in Figure 217.


```

<servlet>
  <name>GatewayServlet</name>
  <description>VAGen Web Transaction Gateway Servlet</description>
  <code>com.ibm.hpt.gateway.GatewayServlet</code>
  <servlet-path>/GatewayServlet</servlet-path>
  <init-parameter>
    <name>hptLogonPage</name>
    <value>/Vagen1LogonPage.jsp</value>
  </init-parameter>
  <init-parameter>
    <name>hptEntryPage</name>
    <value>/Vagen1EntryPage.jsp</value>
  </init-parameter>
  <init-parameter>
    <name>hptErrorPage</name>
    <value>/Vagen1ErrorPage.jsp</value>
  </init-parameter>
  <init-parameter>
    <name>hptErrorLog</name>
    <value>e:/vgtrace/Vaj3ErrorLog.log</value>
  </init-parameter>
  <init-parameter>
    <name>hptLinkageProperties</name>
    <value>e:/vs/csogw.properties</value>
  </init-parameter>
  <init-parameter>
    <name>hptIDManagerHost</name>
    <value>ireland</value>
  </init-parameter>
  <init-parameter>
    <name>hptDateMask</name>
    <value>dd/mm/yyyy</value>
  </init-parameter>
</servlet>

```

Figure 217. Gateway Servlet definition and initialization

Notes:

It appears that if there are two servlet definitions in the XML file with the same <servlet-path>, then the last one parsed wins.

The FileServlet is the one responsible for serving files, and since my gateway definition was after it and had the same <servlet-path>, it was overriding the file serving, and thus a request for an HTML file was invoking the gateway servlet.

So there are two options (option "a" is recommended):

- a. Define an alternate servlet path for the GatewayServlet (something other than /). In Figure 217 we used:

```
<servlet-path>/GatewayServlet</servlet-path>
```

- b. Put the HptGateway servlet definition before the FileServlet definition in the XML and then it works fine, because the FileServlet gets the servlet path of / and serves files properly, since the Gateway servlet is a servlet, and the invocation of /servlet (defined as the servlet path for the Servlet Invoker servlet) causes the gateway to be picked up as a servlet.
2. Update the default_app.webapp file and change the JSP support servlet entry (com.ibm.ivj.jsp.debugger.pagecompile.IBMPageCompileServlet) to switch JSP support to V1.0 (V.91 support is configured in VisualAge for Java). The file can be found in this directory:

```
c:\ibmvjava\ide\project_resources\ibm websphere test environment  
\hosts\default_host\default_app\servlets
```

The replacement definition for the JSP servlet that provides support for JSP V1.0 in the WebSphere Test Environment is shown in Figure 218.

```
<servlet>  
  <name>jsp</name>  
  <description>JSP support servlet</description>  
  <code>com.ibm.ivj.jsp.runtime.JspDebugServlet</code>  
  <init-parameter>  
    <name>workingDir</name>  
    <value>${server_root}/temp/default_app</value>  
  </init-parameter>  
  <init-parameter>  
    <name>jspemEnabled</name>  
    <value>>true</value>  
  </init-parameter>  
  <init-parameter>  
    <name>scratchdir</name>  
    <value>${server_root}/temp/JSP1_0/default_app</value>  
  </init-parameter>  
  <init-parameter>  
    <name>keepgenerated</name>  
    <value>>true</value>  
  </init-parameter>  
  <autostart>true</autostart>  
  <servlet-path>*.jsp</servlet-path>  
</servlet>
```

Figure 218. Defining JSP 1.0 support

3. If required, copy the VisualAge Generator supplied JSP and GIF files to the document root directory. This might be:

```
c:\ibmvjava\ide\project_resources\ibm websphere test
environment\hosts\default_host\default_app\web
```

which is the default, or a directory something like:

```
v:\wsas
```

if you defined a customized WebSphere Test Environment as shown in 14.4.2, “Configure WebSphere Test Environment” on page 327.

Note: You may need to correct the supplied JSP files. See 14.2.5, “Customize JSPs (as required)” on page 317.

4. If required (not done previously), edit the deployed VisualAge Generator entrypoint JSP file (Vagen1EntryPage.jsp) and add a line to support invocation of your Web Transaction.
5. Define the appropriate application and serverLinkage entries in the csogw.properties file to support invocation of the Web Transaction using the target runtime platform (see 13.3.3, “Communications configuration” on page 270).

14.4.4 Add generated components for Web Transaction

After the Web Transaction has been generated and implemented in the target runtime platform (see Chapter 16, “Running Web Transactions” on page 341) we need to add the JSPs and generated JavaBeans to the configured WebSphere Test Environment.

See Chapter 15, “Web Transaction generation” on page 335 for details on generation.

1. Import the class files for the generated beans to a project in the workbench (we used **z WebSphere Test Support**, the same project we used when importing the hptGateway.jar and hpt.jar files).
2. If not already done, update the com.ibm.servlet.SERunner class properties (part of IBM WebSphere Test Environment 3.0 project) by adding all the projects in the VisualAge for Java workspace to the project class path.

3. Copy the generated JSP files, and the .tab files for any VisualAge Generator tables used in the UI Record, to the directory that acts as the document root for the WebSphere Test Environment. This might be:

```
c:\ibmvjava\ide\project_resources\ibm websphere test
environment\hosts\default_host\default_app\web
```

which is the default, or a directory something like:

```
v:\wsas
```

if you defined a customized WebSphere Test Environment.

Note: You may need to modify the generated JSP files. See 14.2.5, “Customize JSPs (as required)” on page 317.

4. If VisualAge Generator tables are used by the generated UI Records, you need to make sure the .tab files can be found by the generated components at runtime.

This is done by adjusting the `com.ibm.servlet.SERunner` class properties (in the IBM WebSphere Test Environment 3.0 project) to add the directory where the .tab files are stored to the Extra directories path setting.

14.4.5 Test generated Web Transaction in VisualAge for Java

1. Start the WebSphere Test Environment by running the `SERunner` class or choosing the **Selected -> Run -> Run main...** Workbench menu option.
2. Start the VisualAge Generator sessionID manager. This can be done by running the `com.bim.hpt.gateway.SessionIDManager` class inside VisualAge for Java or by using this command in a command window to invoke the class:

```
start /min java com.ibm.hpt.gateway.SessionIDManager
```

The `hptGateway.jar` file must be included in the current class path setting of the command window for the command to function.

3. Prepare the Web Transaction runtime platform. This might mean starting the TCP/IP Web Transaction program catcher (`csotcpui.exe`) or the CICS Transaction Gateway and target CICS server.
4. If you want to make use of the JSP execution monitor to trace execution of JSPs, you need to run on the JSP Execution Monitor.

Use the **Workspace -> Tools -> JSP Execution Monitor** menu option. Toggle on (select) the **Enable monitoring JSP execution** toggle.

If you do not enable JSP execution monitoring, the end result is a system that will work just like the runtime configuration. The whole goal is to get access to how the JSPs are being processed, so we recommend that you enable JSP monitoring.

Details on JSPs used for Web Transactions are provided in Chapter 4, “Java Server Pages and the UI Record interface bean API” on page 75.

5. Start a browser and invoke the Gateway Servlet using this URL:

```
http://localhost:8181/GatewayServlet
```

Here, 8181 is the port defined for the WebSphere Test Environment (see 14.4.2, “Configure WebSphere Test Environment” on page 327).

If enabled, the JSP Execution Monitor will be displayed when JSP files are processed by the Gateway Servlet. The JSP Execution Monitor view when processing the Gateway Servlet logon page is shown in Figure 219.

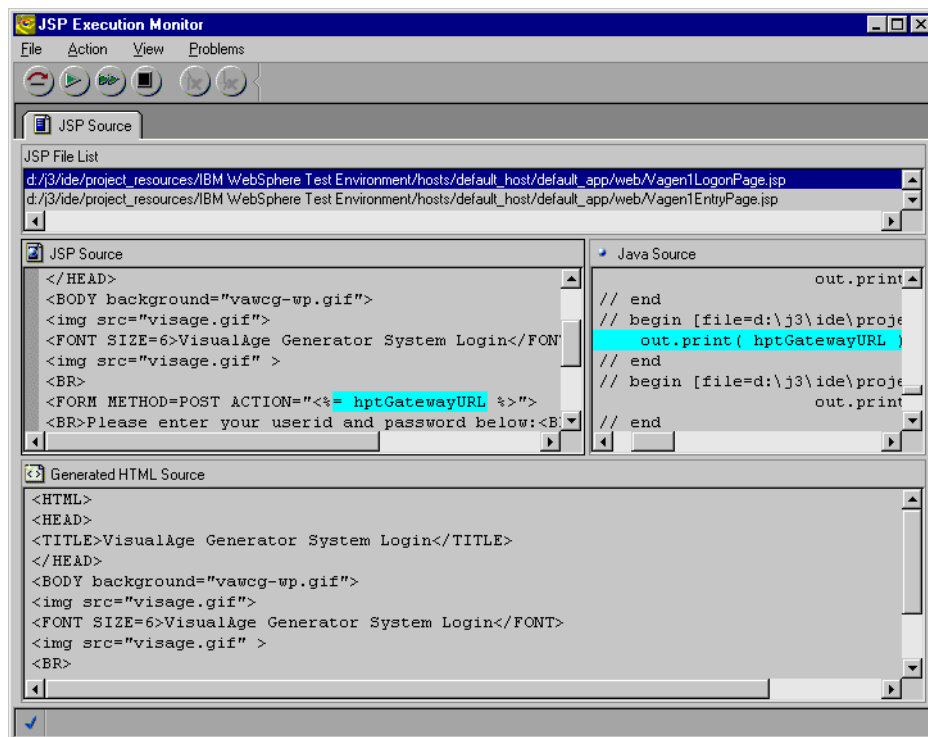


Figure 219. JSP execution monitor

Note: Problems sometimes occur that prevent the WebSphere Test Environment from successfully running the generated system. Often these problems can be resolved by stopping the WebSphere Test Environment, deleting the *JSP Page Compile Generated Code* project, and starting again.

Chapter 15. Web Transaction generation

To support generation, preparation, and subsequent implementation, make sure that the following preparation and runtime platform requirements are satisfied:

Web Transaction preparation

- VisualAge Generator Common Services and Server are installed.
- For Windows NT-based runtime systems, C++ must be installed.
 - If you are using IBM C++, check that the listed directories are included in these environment variables:
 - **INCLUDE** —
x:\IBMCPPW\INCLUDE;x:\IBMCPPW\SDK\WINH;x:\IBMCPPW\SDK\WINH\WINNT and
x:\VGSERVW\INCLUDE;x:\HPTCSOW\INCLUDE
 - **LIB** — x:\IBMCPPW\LIB and x:\VGSERVW\LIB;x:\HPTCSOW\LIB
 - If you are using DB2, the machine where you prepare should have the DB2 Software Developer's Kit installed.
- For Windows NT-based generation and host (MVS) preparation platforms, you must install Object REXX V1.0.3.0 when setting up VisualAge Generator Developer.

This is so the generated COBOL source and JCL may be transferred to the host. You have the option of using LU2 or TCP/IP for the source transfer. PCOMM must be installed to support the use of LU2 for a transfer.

Java bean preparation

- VisualAge Generator Server and common services must be installed.
- Java development kit (JDK) must be installed.
- A value must be supplied for the HPTCLASSDIR environment variable, which identifies where the .CLASS files created during compilation of the generated JavaBeans will be copied as part of preparation processing.

This will tie up with an entry in IBM WebSphere Application Server's classpath (or the classpath of whichever servlet engine you have chosen to use).

The installation process for IBM WebSphere Application Server will add x:\websphere\appserver\classes to the Java classpath (see the admin.config file in the WebSphere /bin directory).

- A value must be supplied for the HPTJSPDIR environment variable, which identifies where the generated JSP files will be copied as part of preparation processing.

The directory identified should be related to the document root you specified for the WebSphere WebApp (see Figure 203 on page 309).

Notes:

- Depending on your generation and preparation configuration, you may have to manually invoke the **webtranJ.bat** file, where webtran is the name of your Web Transaction program, to compile the JavaBeans generated for the UI Record. The xxxJ.bat files reference the HPTCLASSDIR and HPTJSPDIR environment variables.

Note: We found that the xxxJ.bat files did not automatically start for single machine generation/preparation until we had installed fixpak 1.

- Generation processing can be implemented so that a single shared machine performs generation for all programmers in a development team (LAN-based generation or LANGEN). LANGEN setup is described in the VisualAge Generator Generation guide.
- Preparation processing using FTP is discussed in 13.1.4, “Setting up FTP support for program preparation” on page 259.

To implement the runtime system, you must generate and prepare the Web Transaction and UI Record definitions. The Gateway Servlet configuration is then validated for the chosen target runtime environment.

15.1 Windows NT Web Transactions — base system deployment

The implementation of a Windows NT-based system is discussed in this section.

Note: The linkage table referenced in the generate command applies to the calls from a Web Transaction program to server (CALLED BATCH) programs. The linkage table does not control the generation or runtime interface between the Gateway Servlet and the Web Transaction program.

15.1.1 Generation

Generate and prepare the Web Transaction programs, associated UI Records, and VisualAge Generator tables, and any called server programs.

We used the generation command shown in Figure 220.

```
HPTCMD GENERATE CONVMOD
/SYSTEM=WINNT
/LINKAGE=VGJAVATCPIP.LKG
/PACKAGENAME=vgtwt.beans
/JAVASYSTEM=WINNT
/PROJECT="zz ISA ITSO Tests", "2.8"
/GENOUT=e:\vgenout\%EZEENV%
/DBUSER=vgtwt /DBPASSWORD=VGDBA
```

Figure 220. Generation command: Windows NT system

Generation started preparation processing.

15.1.2 Configure Gateway Servlet access

Edit the CSOGW.properties file and create the appropriate application and serverLinkage entries (see 13.3.3, “Communications configuration” on page 270 for details).

15.2 CICS NT Web Transactions — base system deployment

The implementation of a CICS NT-based system is discussed in this section.

15.2.1 Generation

Generate and prepare the Web Transaction programs, associated UI Records, and VisualAge Generator tables, and any called server programs.

We used the generation command shown in Figure 221.

```
HPTCMD GENERATE CONVMOD
/SYSTEM=NTCICS
/LINKAGE=VGJAVACICS.LKG
/PACKAGENAME=vgtwt.beans
/JAVASYSTEM=WINNT
/PROJECT="zz ISA ITSO Tests", "2.8"
/GENOUT=e:\vgenout\%EZEENV%
/DBUSER=vgtwt /DBPASSWORD=VGDBA
```

Figure 221. Generation command: CICS NT system

Generation started preparation processing.

15.2.2 Define your generated Web Transaction(s) to CICS

Select the CICS region on the **CICS Administration Utility**, invoke the context menu, and choose **Resources-> Program**. On the window that opens, select the **Programs** menu and choose **New...** Figure 222 shows the window which then opens.

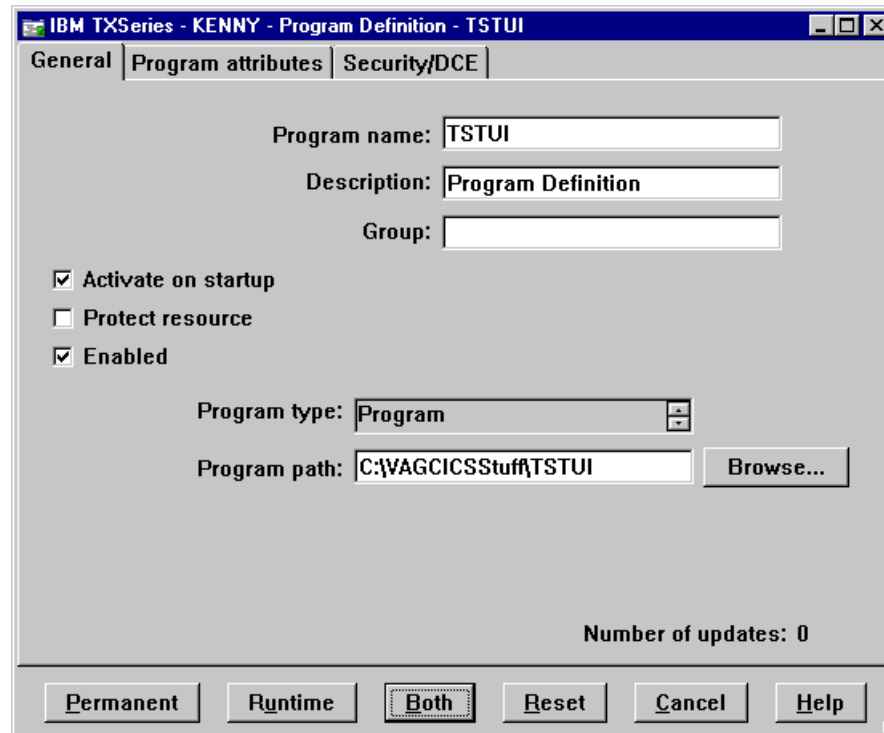


Figure 222. Adding a new program to CICS

Specify the following on the General and Security/DCE pages:

- Program name — We recommend that program name matches the actual Web Transaction name as defined in VisualAge Generator.
VisualAge Generator map group load modules and server side VisualAge Generator table DLLs do not need to be defined to CICS.
- Group — Use this to identify relationships between your CICS definitions.
- Program path — Use the **Browse...** button to find the program .IBMCPP file.
- Resource level security key — We used a setting of public. See 16.3, “Security” on page 352 for a more detailed security discussion.

15.2.3 Configure Gateway Servlet access

Edit the CSOGW.properties file and create the appropriate application and serverLinkage entries (see 13.4.8, “Communications configuration” on page 279 for details).

15.3 CICS/ESA Web Transactions — base system deployment

The implementation of a CICS/ESA-based system is discussed in this section.

Note: The linkage table referenced in the generate command applies to the calls from a Web Transaction program to server (CALLED BATCH) programs. The linkage table does not control the generation or runtime interface between the Gateway Servlet and the Web Transaction program.

15.3.1 Generation

Generate and prepare the Web Transaction programs, associated UI Records, and VisualAge Generator tables, and any called server programs.

We used a generation command similar to that shown in Figure 223.

```
HPTCMD GENERATE WEBTRAN
/SYSTEM=MVSCICS
/LINKAGE=VGJAVACICS.LKG
/PACKAGENAME=vgtw.beans
/JAVASYSTEM=NTCICS
/PROJECT="zz ISA ITSO Tests", "2.8"
/GENOUT=e:\vgenout\%EZEENV%
/DBUSER=vgtw /DBPASSWORD=VGDBA
```

Figure 223. Generation command: CICS/ESA system

We also requested that generation start preparation processing.

15.3.2 Define your generated Web Transaction(s) to CICS

Define the Web Transaction program to CICS. We recommend that the program name matches the actual Web Transaction name as defined in VisualAge Generator. The program will be COBOL.

You will need a transaction definition which executes the mirror program DFHMIRS. CPMI is the default supplied transaction. If you wish to use this for your Web Transactions, you will need to change the TWASIZE setting to 1024.

If your program uses DB2, you need to ensure a Resource Control Table entry (RCT) is set up for the transaction you are using to point at a DB2 plan.

If your Web Transaction references any VisualAge Generator tables in the Web Transaction logic that runs on the VisualAge Generator server side, that is, tables not used only in either:

- The UI record definition
- The EZEUIERR invocations

Then you will need to create program definitions for these tables.

15.3.3 Configure Gateway Servlet access

Edit the CSOGW.properties file and create the appropriate application and serverLinkage entries (see 13.5.5, “Communications configuration” on page 299 for details).

Chapter 16. Running Web Transactions

The process required to run a Web Transaction using any of the VisualAge Generator and WebSphere Application Server runtime platforms we have configured is discussed in this chapter.

16.1 Deploy generated code

Several tasks must be performed to deploy the different types of code generated by VisualAge Generator for a Web Transaction.

16.1.1 JSPs, JavaBeans, and tables used in a UI Record

On the Web server, either manually or as part of the preparation processing started during generation, do the following:

- Copy the JSP(s) produced by generation to the document root directory (see Figure 203 on page 309 for the document root definition).

Be sure correct the JSP files, if required (to change *imports* to *import*). See 14.2.5, “Customize JSPs (as required)” on page 317.

- Copy all Java class files (produced when the JavaBeans created during generation are compiled) to the directory `x:\websphere\appserver\classes`. This directory is always searched by WebSphere Application Server.

Be sure to preserve any directory structure produced as a result of the package name used during generation.

An alternative directory, if included in the `-classpath` parameter, can be used. See Figure 203 on page 309 for details.

Note: You may have to correct the syntax in the Java files generated for a UI Record that uses message inserts in edit functions. Fixpak 1 code creates the string `[] []` in the Java implementation of an edit function when the code should only include `[]`.

- VisualAge Generator tables referenced by a UI Record must be in a directory included in the CLASSPATH used by the Gateway Servlet configured in the WebSphere Application Server environment.

We chose to copy the TAB files (produced during VisualAge Generator generation) to the directory `x:\websphere\appserver\classes`. This directory is always searched by WebSphere Application Server.

Note: Depending on your generation and preparation configuration, you may have to first invoke the `webtranJ.bat` file, where `webtran` is the name of your Web Transaction program, to compile the JavaBeans generated for the UI

Record. These generated .bat files reference the environment variables HPTCLASSDIR and HPTJSPDIR. See Chapter 15, “Web Transaction generation” on page 335 for details. With Fixpak 1 this xxxxJ.bat file runs during preparation.

16.1.2 Web Transaction program materials

- **Windows NT or CICS NT** — Copy any VisualAge Generator tables referred to in the Web Transaction, other than those only referenced in either a UI record definition or EZEUIERR invocation, into the directory referred to by the FCWDPATH environment variable.

- Tables referenced in a UI Record must be in a directory included in the CLASSPATH used by WebSphere Application Server.

- **Windows NT** — Copy your Web Transaction executables (DLLs) into a directory specified in the PATH environment variable.

If you find that your DLL is locked, stop and then restart the VisualAge Generator TCP/IP catcher program to release the DLL.

- **CICS NT** — Identify the location of each generated .IBMCPP executable as part of the CICS program definition for each Web Transaction.

Use CICS to request a CEMT SET program NEWCOPY if the program was already loaded.

- **CICS/ESA** — Make sure you generated the Web Transactions and VisualAge Generator server side tables into a library in the CICS system RPL.

Use CICS to request a CEMT SET program NEWCOPY if the program(s) were already loaded.

Make sure you bound the relevant package/plan if your Web Transaction issues any DB2.

Timestamps are stored and compared to ensure the runtime version of the Web Transaction matches the corresponding interface bean and data bean code. The Gateway Servlet will send an error message if a timestamp mismatch is found. (This something like the DB2 program and plan match.)

16.1.3 Invoking Web Transaction from default entry point JSP

If you wish to be able to invoke your Web Transaction from the default entry point page, you need to edit the Vagen1EntryPage.jsp, and add an entry for the target Web Transaction.

The source for a customized entry point page is shown in Figure 224.

```
<%@ page errorPage="Vagen1ErrorPage.jsp" %>
<jsp:useBean id="hptGatewayURL" class="java.lang.String" scope="request" />
<HTML>
<HEAD>
  <TITLE>VisualAge Generator System Entry</TITLE>
</HEAD>
<BODY background="vawcg-wp.gif">
<img SRC="visage.gif" >
<FONT SIZE=6>VisualAge Generator System Entry</FONT>
<img SRC="visage.gif" >
<BR>
<BR>
<FORM METHOD=POST ACTION="<%= hptGatewayURL %>">
<BR>Choose a Program to execute below:<BR>
<SELECT NAME="hptAppld" SIZE=10>
  <OPTION VALUE=WEBTRAN>Base WebTran Program
  <OPTION VALUE=NONE0>-----
  <OPTION VALUE=CONVMOD><b>Converse Model Demo</b>
  <OPTION VALUE=FRSTFRM>First Form Model Demo
  <OPTION VALUE=FRSTREC>First Record Model Demo
  <OPTION VALUE=FRSTPLK>Program Link Model Demo
  <OPTION VALUE=NONE1>-----
  <OPTION VALUE=BLBANWE>VAGT Bank List
  <OPTION VALUE=CLCUSWE>VAGT Cust List
</SELECT>
<BR>
<!-- VG Gateway control fields - DO NOT MODIFY -->
<INPUT TYPE="submit" VALUE="Execute" NAME="hptExec">
<INPUT TYPE="submit" VALUE="Logout" NAME="hptLogout">

<BR><A HREF='<%= hptGatewayURL
%>?hptAppld=MINTEST&hptExec=exec&yyy=yyy' target='xyz'>This is a test link</A>
<BR><A HREF='<%= hptGatewayURL
%>?hptAppld=MINTEST&hptExec=exec&yyy%23=xyz+abc%20%c3%a7def'>This is
another test link</A>
<!-- VG Gateway control fields - DO NOT MODIFY -->
<jsp:useBean id="hptErrorData" class="java.lang.String" scope="request" />
<BR><FONT COLOR="#FF0000"><%= hptErrorData %></FONT>
</BODY>
</HTML>
```

Figure 224. Gateway Servlet entry page definition

The target Web Transaction entry is added inside the SELECT list named hptAppld, as shown below:

```
<OPTION VALUE=webtran>description of your webtran
```

The *webtran* value entered must match the name and case (use upper) for the Web Transaction program, just as in the CSOGW.properties file.

16.2 Runtime processing

Runtime environment startup and the implementation of Gateway Servlet and Web Transaction processing is discussed in this section.

16.2.1 System Startup

We can now start up the deployed system.

VisualAge Generator runtime platform

Tasks are runtime platform dependent:

- **Windows NT or CICS NT** — Start the database server.
- **Windows NT** — Start the VisualAge Generator catcher program, CSOTCPUI.
- **CICS NT** — Start the CICS region on the VisualAge Generator server machine from the **IBM CICS Server for windows NT->TXSeries Administration Utility** option.
- **CICS/ESA** — Make sure the mainframe database system (if you are using it) and the CICS system are started.

Make sure the CICS system has successfully attached to the database system.

WebSphere Application Server platform

Some tasks are runtime platform dependent:

- Start the local DB2 system, if required. If you are using IBM WebSphere Application Server Standard Edition, a local DB2 system is not required.
- Start the Web server, if required. Start Apache from the **Apache Web Server->Start Apache as console app** (or start it as a Windows NT service). Actual Web server invocation depends on the product chosen.
- Start the session ID manager. The following line in a bat file will start the session ID manager:

```
java com.ibm.hpt.gateway.SessionIDManager
```

It is best to start this before the WebSphere Application Server, as you can get intermittent port conflicts between the two.

The session ID manager was discussed in 14.2.8, “Set up VisualAge Generator session ID manager” on page 318.

- Start IBM WebSphere Application Server from the services panel.
- Start the WebSphere administrators console from the start menu. Once the console has launched, select the **Topology** tab, expand the tree you find there, select the application server you created and start it by clicking the green blob on the toolbar.
- **CICS** — Start CICS client from the **start CICS client** START menu option if you are using a CICS server as a target runtime platform.
- **CICS/ESA** — Start PCOMM if you are using CICS/ESA as a target runtime platform.

16.2.2 Gateway Servlet invocation

This will be based on the URI you gave it when you defined it in Figure 202 on page 308. If you used our suggestion, the URL would be in the following form:

```
http://hostname/webapp/yourserverWebApp/GatewayServlet
```

Here, *hostname* is the TCP/IP hostname or address of the machine which has IBM WebSphere Application Server on it.

The Gateway Servlet will ask you to logon, if an `hptLogonPage` is specified as part of its initialization parameters (see Figure 225).

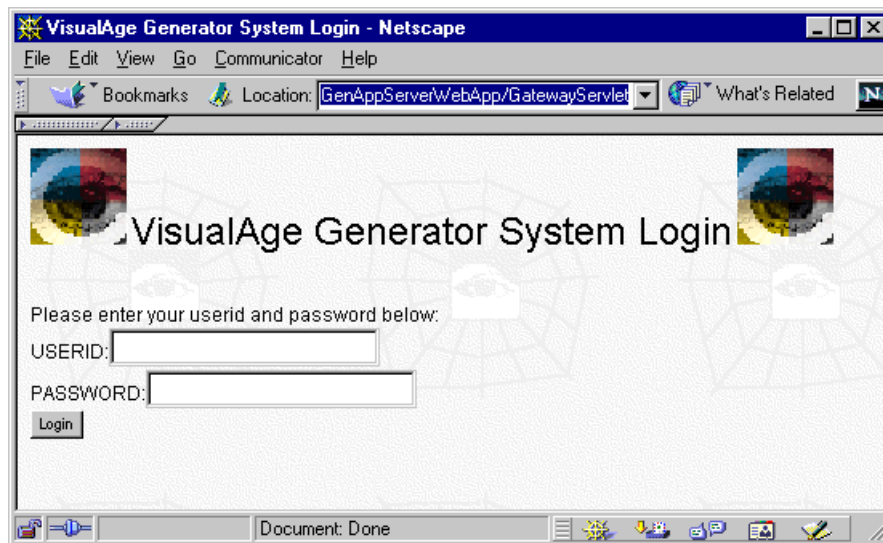


Figure 225. Gateway Servlet Logon page

After the logon, the Gateway Servlet will serve the entry point page or start the entry point application, again depending on the current initialization parameters. The entry point page, with a list of configured Web Transactions, is shown in Figure 226.

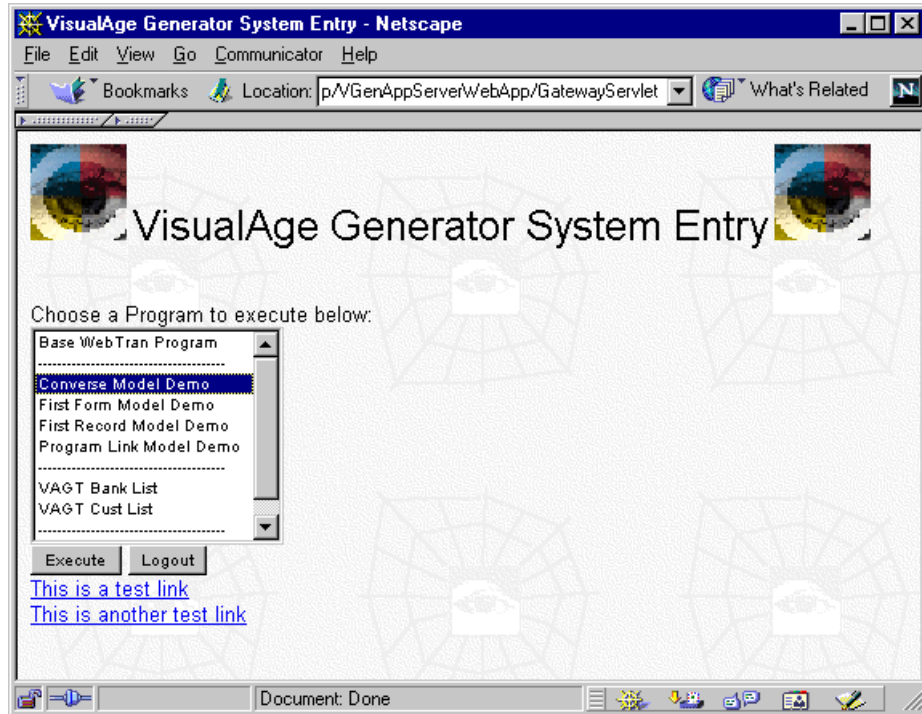


Figure 226. Gateway Servlet Entry page

See 14.2.4, "Define VisualAge Generator Gateway Servlet" on page 312 for more about Gateway Servlet configuration.

See Part 16.3, "Security" on page 352 for more about logon processing and security.

16.2.3 Gateway Servlet processing

Figure 227 shows the Gateway Servlet and other components configured and running in a Windows NT-based WebSphere Application Server runtime environment.

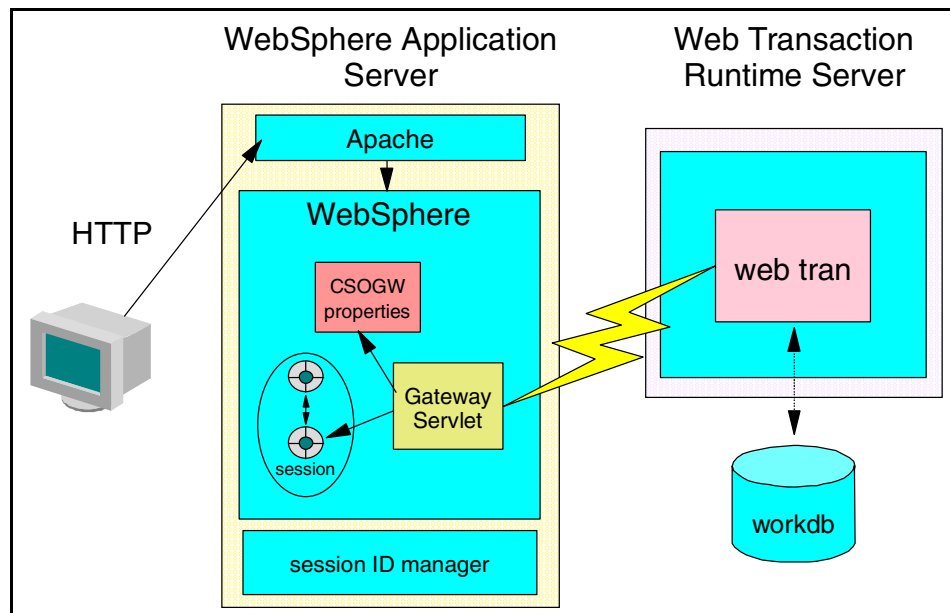


Figure 227. Gateway Servlet runtime system processing

When you select a Web Transaction entry from the entry point page list and click to run it, the following processing takes place:

- An HTTP request is sent to the Web server.
- Because of the format of the URL, the Web server passes the request to the WebSphere Application Server plug-in.
- WebSphere Application Server invokes the Gateway Servlet passing the HTML FORM input data to the servlet. The input data tells the Gateway Servlet which Web Transaction to invoke.
- The Gateway Servlet reads the CSOGW.property file to determine which technique will be used to connect to the target runtime platform to run the Web Transaction.
- The Gateway Servlet invokes other beans which in turn connect to the target Web Transaction program.

- The Web Transaction runs. When it XFERs with a UI Record or issues a CONVERSE, control returns back through the communication chain, passing back the UI Record data.

When required:

- A work database is used to store data for the Web Transaction program when control returns to the Gateway Servlet.
- Session objects are used to store the data beans populated by the Gateway Servlet for the user data associated to the UI Record.

This processing is performed for selected Web Transaction UI Record processing options:

- During a CONVERSE, the work database stores all program data.
- During an XFER when the XFER includes a working storage record (XFER webtran WS Record, UI Record), the work database only stores the working storage record included in the XFER.
- During a CONVERSE or when the XFER includes a working storage record, a session object is used to store the data bean.

See 5.1, “Concepts” on page 85 for details on Web Transaction implementation of saved state processing.

- As the final response to the initial request, the Gateway Servlet invokes a JSP which will use the UI Record data to compose the Web page that will be displayed in the browser.

Subsequent invocations of your Web Transaction go through a similar path, except:

- Before the CSOGW.property file is read:
 - If the Gateway Servlet is invoked from a SUBMIT button on an HTML FORM (the default FORM generated by VisualAge Generator, not one you defined yourself in your UI Record), it populates the data bean with data the user entered into the INPUT fields (if any) on the JSP that was previously served.
 - The Gateway Servlet invokes any edits specified for the data items in the UI Record (other than VisualAge Generator server side edit routines). The Gateway Servlet re-serves the original JSP if there is an error.

- When the Web Transaction is invoked by the catcher, the data stored in the work database is retrieved before the Web Transaction continues its processing, if:
 - The Web Transaction was invoked from a SUBMIT button in a JSP on the default form generated for a UI Record, that is to say, not a FORM item defined by you, the programmer, and
 - The JSP was previously sent as a result of an `XFER webtran WS Record, UI Record` transfer request, or a CONVERSE I/O processing option.

You may have noticed by now that when using FORMs you defined yourself in the UI Record, they behave differently from the default one. This is because your own FORM represents starting a new thread with a new invocation of a Web Transaction rather than continuing an ongoing conversation. With your own FORM you may pass your UI Record data into a new UI Record when the user clicks the SUBMIT button. No data will be retrieved from a work database on the runtime platform.

See Chapter 5, “Web Transaction design concepts and considerations” on page 85 for more information on how Web Transaction structure determines how runtime processing is implemented.

16.2.4 Windows NT Web Transaction processing

Figure 228 shows the components configured and running in a Windows NT-based runtime environment.

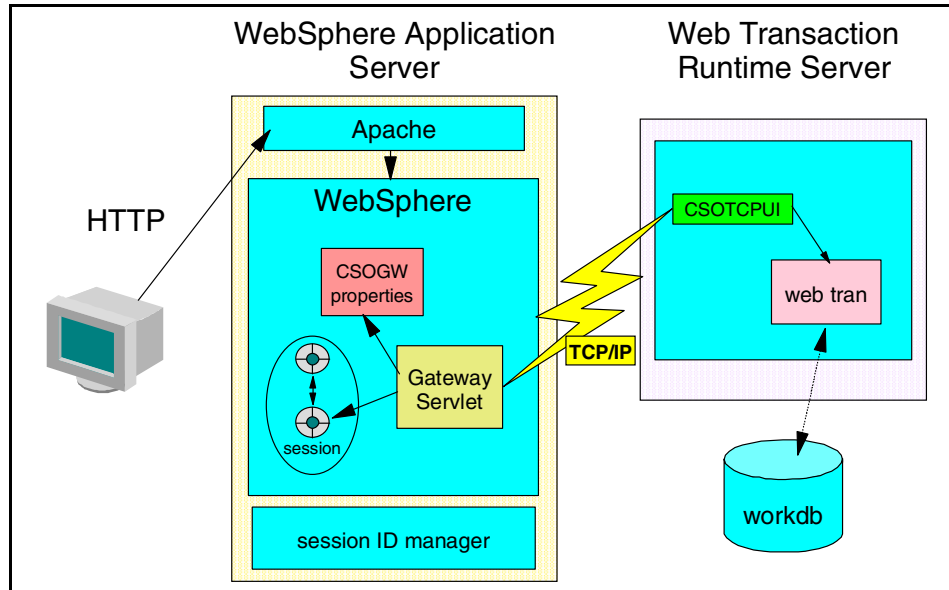


Figure 228. Windows NT runtime system processing

Web Transaction invocation on a Windows NT runtime platform occurs when the Gateway Servlet uses TCP/IP to connect to the VisualAge Generator server catcher program (CSOTCPUI). The name of the Web Transaction to be invoked and input data is passed.

Windows NT-based Web Transactions use a VisualAge Generator managed work database to store data over a CONVERSE I/O processing request or XFER webtran WS Record, UI Record transfer request.

16.2.5 CICS Web Transaction processing

Figure 228 shows the components configured and running in a CICS-based runtime environment.

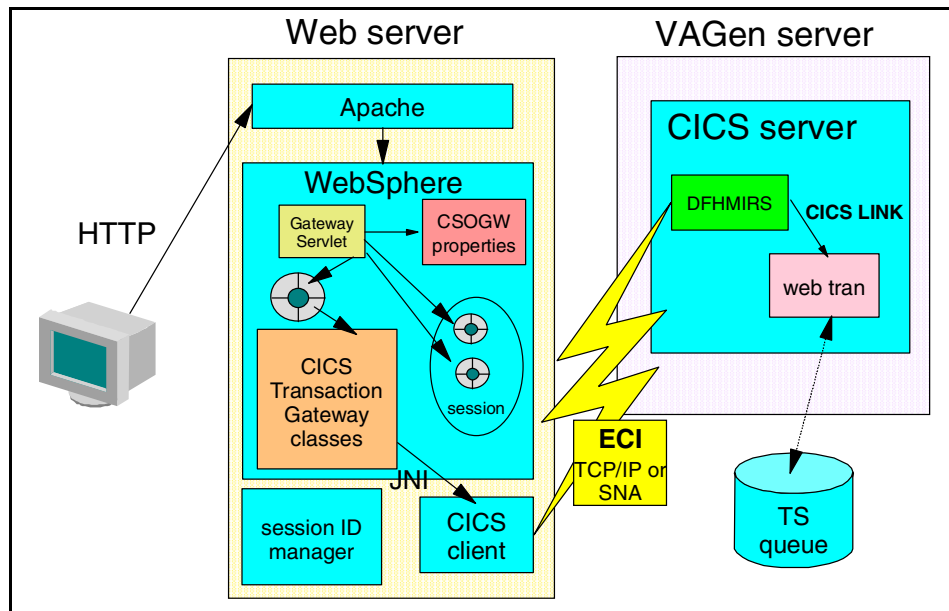


Figure 229. CICS runtime system processing

Web Transaction invocation on a CICS runtime platform occurs when the Gateway Servlet uses CICS Transaction Gateway classes to invoke the Java Native Interface to communicate with the CICS client. The CICS client uses the External Call Interface to invoke a transaction on the target CICS server.

Note: We used TCP/IP to connect to CICS NT and SNA with PCOMM to connect to CICS/ESA.

The transaction ID used will start the program DFHMIRS, passing the name of the Web Transaction to be invoked.

DFHMIRS LINKs to this program, passing client data (if any) on to the Web Transaction program.

CICS-based Web Transactions use CICS temporary storage to store data over a CONVERSE I/O processing request or XFER webtran WS Record, UI Record transfer request.

16.2.6 Debugging Web Transactions at runtime with CEDF

If you are getting processing errors after invoking a CICS-based Web Transaction, you may find it helpful to trace your Web Transaction in CICS NT using CEDF.

This can be done if a telnet session has been defined (see 13.4.6, “Add VisualAge Generator runtime and debug transactions” on page 277).

To use CEDF:

- Invoke the Web Transaction once, so a sysid is assigned.
- Run the CSTD transaction in your Telnet session and choose option 10 (ISC Summary Statistics). This screen shows you the sysid assigned.
- Exit from CSTD and type `CEDF sysid`, where `sysid` corresponds to that shown in the ISC Summary.
- Hit **Pause** to clear the screen.
- Now try to invoke your Web Transaction again from the browser. The CEDF initialization screen should pop up on your Telnet session and you can step through the code, and examine working storage as you normally would with CEDF.
- Make sure the security level you have applied to the CEDF transaction in CICS matches the security level on the actual transaction you are trying to run it against or you can get security violations when you try to use CEDF.

16.3 Security

We will look at how a user can explicitly logon and enter a userid and password and associated issues, then go on to look at how we can make the security transparent to the user.

Note: This discussion references techniques available in CICS for the Windows NT platform (we used TXSeries 4.3). Similar techniques are available for other CICS runtime environments. Consult the CICS client documentation for further configuration information.

16.3.1 Logon technique

When a user invokes the gateway servlet for the first time, as we do in 16.1.3, “Invoking Web Transaction from default entry point JSP” on page 342, they are presented with a logon page (unless you did not specify `hptLogonPage` as an initialization parameter of the Gateway servlet).

16.3.1.1 Logging on

The default logon page, `Vagen1LogonPage.jsp`, causes the Gateway Servlet to store the userid and password in an IBM WebSphere Application Server session object for the user for later use on any subsequent Web Transactions

the user attempts to invoke. The userid and password are not validated at all at this point. The user could even leave them blank.

The JSP source for the default logon page Vagen1LogonPage.jsp is shown in Figure 230.

```
<%@ page errorPage="Vagen1ErrorPage.jsp" %>
<jsp:useBean id="hptGatewayURL" class="java.lang.String" scope="request"/>
<HTML>
<HEAD>
<TITLE>VisualAge Generator System Login</TITLE>
</HEAD>
<BODY background="vawcg-wp.gif">

<FONT SIZE=6>VisualAge Generator System Login</FONT>

<BR>
<FORM METHOD=POST ACTION="<%= hptGatewayURL %>">
<BR>Please enter your userid and password below:<BR>
USERID:<INPUT TYPE="text" NAME="hptUserid"><BR>
PASSWORD:<INPUT TYPE="password" NAME="hptPassword"><BR>
<INPUT TYPE="submit" VALUE="Login" NAME="hptLogin">
<BR>
</FORM>
</BODY>
</HTML>
```

Figure 230. Default Gateway Servlet logon page

The user will not be asked to logon again unless:

- They close their browser.
- The session is timed out by the WebSphere Application Server due to user inactivity.
- The Gateway Servlet gets an internal error.
- They ask to log out of the system.

An example of how to logout is shown in the default entry page Vagen1EntryPage.jsp (see Figure 231). The Gateway Servlet is invoked from an HTML FORM, where the SUBMIT button pressed has a VALUE of *Logout*.

Once the user has responded to the logon page, the entry page or entry application (Web Transaction) is served. If you choose to have an entry application, then authentication of userid and password will now take place; otherwise the entry point JSP is served.

If you use an entry page, such as the supplied entry page shown in Figure 231, when an option is selected out of the list of Web Transactions and the submit button is clicked, authentication of the provided userid and password values will take place.

```

<%@ page errorPage="Vagen1ErrorPage.jsp" %>
<jsp:useBean id="hptGatewayURL" class="java.lang.String" scope="request" />
<HTML>
<HEAD>
<TITLE>VisualAge Generator System Entry</TITLE>
</HEAD>
<BODY background="vawcg-wp.gif">
<img SRC="visage.gif" >
<FONT SIZE=6>VisualAge Generator System Entry</FONT>
<img SRC="visage.gif" >
<BR>
<BR>
<FORM METHOD=POST ACTION="<%= hptGatewayURL %>">
<BR>Choose a Program to execute below:<BR>
<SELECT NAME="hptAppId" SIZE=10>
  <OPTION VALUE=MINTEST>MINTEST
  <OPTION VALUE=TSTUI>TSTUI
</SELECT>
<BR>
<!-- VG Gateway control fields - DO NOT MODIFY -->
<INPUT TYPE="submit" VALUE="Execute" NAME="hptExec">
<INPUT TYPE="submit" VALUE="Logout" NAME="hptLogout">
<BR><A HREF='<%= hptGatewayURL
%>?hptAppId=MINTEST&hptExec=exec&yyy=yyy' target='xyz'>This is a test link</A>
<BR><A HREF='<%= hptGatewayURL
%>?hptAppId=MINTEST&hptExec=exec&yyy%23=xyz+abc%20%c3%a7def'>This is
another test link</A>
<!-- VG Gateway control fields - DO NOT MODIFY -->
<jsp:useBean id="hptErrorData" class="java.lang.String" scope="request" />
<BR><FONT COLOR="#FF0000"><%= hptErrorData %></FONT>
</BODY>
</HTML>

```

Figure 231. Default entry page

So authentication of userid and password is delayed until the invocation of a Web Transaction, and is only performed if the target runtime system both uses a userid and password and is configured to accept the userid and password entered by an end user in the logon page.

This means that, for runtime environments such as Windows NT, where authentication is not performed, there is no reason to configure a hptLogonEntry page. The first page should be the default entry page with a list of available Web Transaction systems or an entry application.

16.3.1.2 Delayed authentication issues

The problem with delayed authentication is that if the userid and password are not valid, a target system such as CICS has no way to ask the Web browser user to send the logon data again. In our configuration the CICS client software is installed on the Web server. In many common client/server configurations, the client machine has the CICS client installed, which allows for a CICS logon prompt that can be sent for re-entry of the logon data.

If the user fails to enter a valid userid and password in the default logon page, and the CICS client was started to disable security prompts (/N option), when they try to invoke a Web Transaction from the entry point page, they will see an error message similar to that shown in Figure 232.

Note: If the CICS client was not started with the /N option, the logon prompt will be displayed on the Web server.

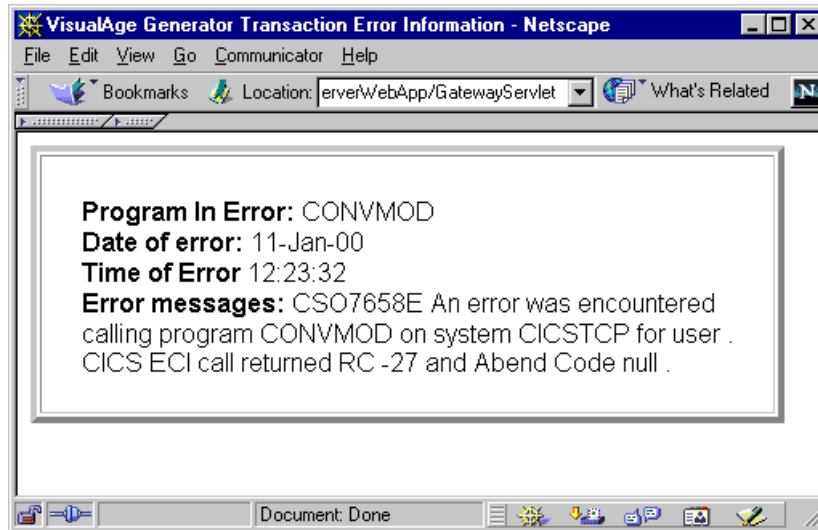


Figure 232. No userid or password

If the user does not specify a *valid* userid and password, CICS will just assume you are the "default user", but the user would not be told they had entered invalid logon data.

To ensure that only validated logons can run Web Transactions, resource level security must be applied to the CICS transactions and programs so that the default user does not have access.

If resource level security was implemented, the default user attempt to access a protected Web Transaction would result in an error message similar to that shown in Figure 233.

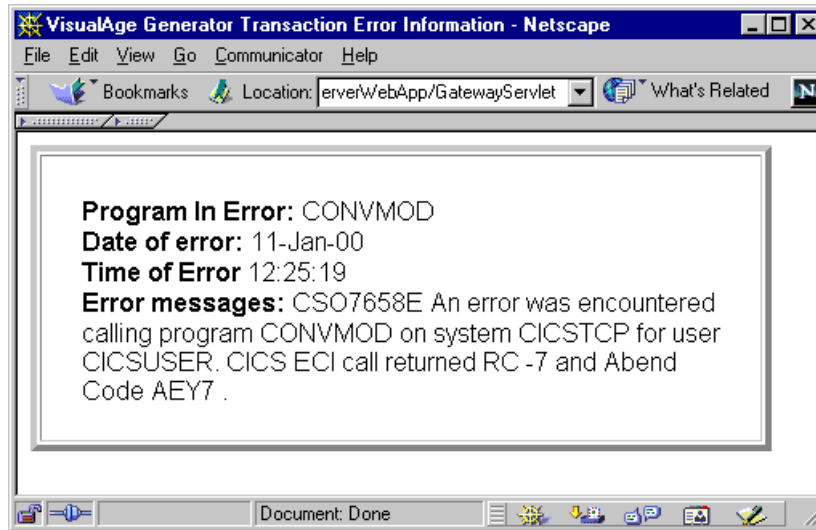


Figure 233. Invalid userid or password

The userid and password which CICS uses to validate against must be set up as a user definition in CICS. You could use DCE or CICS itself to deal with the security. Internet users will not have direct access to CICS or DCE, so this makes it nearly impossible for them to be in control of their userids or to change their passwords.

Note: We modified CSOERRORUIR.JSP to adjust the formatting of the error message so that the text wrapped in the browser. We removed the `<pre>` and `</pre>` tags from this line:

```
VGDataElement line = (VGDataElement)lines.nextElement(); %><pre><%=  
line.getTextValue() %></pre>
```

The line shown below resulted in the wrapped text shown in Figure 233:

```
VGDataElement line = (VGDataElement)lines.nextElement(); %><%=  
line.getTextValue() %>
```

16.3.1.3 Configuring for user login

In order to prevent logon dialogs being sent to the CICS client installed on the Web server machine when the user and password are left blank in the default logon page, you need to start up the CICS client with `/n` as shown:

```
C:\>CICSCLI /s=cics region name /n
```

Make sure you specify `UpperCaseSecurity=N` in the `CICSCLI.INI` file so that lower case user ids and passwords may be used.

To protect your Web Transactions, so that only those users defined to CICS can run them, you need to do the following:

- Edit the user definition you set up previously (see 13.4.5, “Define CICS user” on page 277). Choose the **Security/DCE** tab. In the **Resource level security key list**, type in a number from 2-24.
- Edit the transaction definition for CPMI. Choose the **Security** tab, and make sure the **Type of RSL checks** is *not* set to none. In our case, using CICS internal security, we want it set to internal. It is OK for the **Resource level security key** to be set to public.
- Edit the program definitions for your Web Transactions. Choose the **Security/DCE** tab. Set the **Resource level security key** to be a number which matches the one you specified for your user definition.

16.3.1.4 Implementing immediate authorization

The use of an entry application rather than an entry JSP page will give the appearance of instant userid and password validation once the logon page has been served. This is because the identified Web Transaction will have been run based on the provided userid and password values. Success means the business application starts; whereas failure (either an authentication or authorization problem) will result in an error message.

This will mean authentication and authorization failure error messages, like the ones shown in Figure 232 and Figure 233, will now appear on the error page defined in the Gateway Servlet’s initialization parameters. The source for the default error page, `CSOERRORUIR.jsp`, is shown in Figure 234.

Note: The specification of an entry JSP page or entry application is defined in the initialization parameters of the Gateway Servlet. See Table 11 on page 315.

```

<%@ page errorPage="Vagen1ErrorPage.jsp" %>
<%@ page imports = "com.ibm.vj.ui.bean.VGDataElement" %>
<jsp:useBean id="CSOERRORUIR" scope="request"
class="com.ibm.hpt.gateway.CSOERRORUIRBean" />
<!-- This is JAVA code that gets the individual data elements from the UI Bean that are to be used
by this page to access all dynamic data. -->
<%
    VGDataElement PGMERROR = CSOERRORUIR.getPGMERROR();
    VGDataElement DATEERROR = CSOERRORUIR.getDATEERROR();
    VGDataElement TIMEERROR = CSOERRORUIR.getTIMEERROR();
    VGDataElement MSGCOUNT = CSOERRORUIR.getMSGCOUNT();
    VGDataElement MSGERROR = CSOERRORUIR.getMSGERROR(); %>
<HTML><HEAD>
<TITLE><%= CSOERRORUIR.getTitle() %></TITLE>
</HEAD>
<BODY>
<TABLE BORDER="4" CELLPADDING="20">
<TR>
<TD>
<TABLE ALIGN="left">
<TR>
<TD>
<FORM METHOD="POST" ACTION="<%= CSOERRORUIR.getGatewayURL() %>"

<!-- No comment defined for item PGMERROR -->
<b><%= PGMERROR.getLabel() %> </b>
<%= PGMERROR.getTextValue() %>
<br>
<!-- No comment defined for item DATEERROR -->
<b><%= DATEERROR.getLabel() %> </b>
<%= DATEERROR.getTextValue() %>
<br>

<!-- No comment defined for item TIMEERROR -->
<b><%= TIMEERROR.getLabel() %> </b>
<%= TIMEERROR.getTextValue() %> <br>

<!-- No comment defined for item MSGERROR -->
<% if (MSGERROR.notEmpty()) { %>
<b><%= MSGERROR.getLabel() %> </b>
<% { %>
<%         java.util.Enumeration lines = MSGERROR.occurrences();
while (lines.hasMoreElements()) {
    VGDataElement line = (VGDataElement)lines.nextElement(); %>
<pre>
<%= line.getTextValue() %></pre>
<% } %><% } %><% } %><br><P>

<!-- VG Gateway control fields - DO NOT MODIFY -->

<INPUT TYPE=HIDDEN NAME="hptAppId" VALUE="<%= CSOERRORUIR.getAppId() %>">
<INPUT TYPE=HIDDEN NAME="hptSessionId" VALUE="<%= CSOERRORUIR.getSessionID()
%>">
<INPUT TYPE=HIDDEN NAME="hptPageId" VALUE="<%= CSOERRORUIR.getPageId() %>">

</FORM></TD></TR></TABLE></TD></TR></TABLE>
</BODY></HTML>

```

Figure 234. Default error page

You could tailor the error page so that a more meaningful response (instead of the message text composed from the `CSOERRORUIRBean`) is provided to the end user (see Chapter 4, “Java Server Pages and the UI Record interface bean API” on page 75 for details of JSP syntax). Your customization could also provide the end user with an HTML hypertext link tag so they can get back to the logon page and enter a new userid and password value. You could even use client side JavaScript in the JSP, or else an HTML META tag to cause a timed re-route to the logon page.

Now you can simulate everything except user and password maintenance. This becomes a problem when you are using CICS-based userid and password validation and the password for a given userid has expired. One example might be finding a way for the end user to change the password when they are not directly connected to the target CICS environment (as when using a TUI system). This has always been a difficult issue to resolve for client/server environments, and the problems are the same for Web Transactions.

The CICS External Security Interface (ESI) provides some features that will allow you to construct a password maintenance approach that does not require a TUI-based CICS login.

The ESI allows a non-CICS application to invoke services provided by advanced program-to-program communication (APPC) password expiration management (PEM).

APPC PEM with CICS provides support for an APPC architected sign-on transaction that signs on user IDs to a CICS server and processes requests for a password change by:

- Identifying a user and authenticating that users identification. Notifying specific users during the authentication that their passwords have expired.
- Letting users change their passwords when (or before) the passwords expire.
- Telling users how long their current passwords will remain valid.
- Providing information about unauthorized attempts to access the server with a particular user identifier.

To use APPC PEM:

- The CICS Universal Client must be connected to the CICS server over APPC.
- An external security manager (ESM), such as resource access control facility (RACF), must also be available to the CICS server.

- ESI calls can be included within your ECI or EPI application.
- Only CICS servers returned by the CICS_EciListSystems and CICS_EpiListSystems functions are acceptable.

A discussion on the use of the APPC PEM support in CICS can be found in the *CICS RACF Security Guide*, SC33-1701-02.

16.3.2 Transparent login to CICS

In this section we will look at configuration, then the pros and cons.

16.3.2.1 Configuration

You need to remove hptLogonPage as an initialization parameter for the Gateway Servlet, and feed through a userid and password from the CICS client on the Web server that all users can run with.

You can feed through userid and password by starting the CICS client in two stages:

```
c:\>CICSCLI /s=cics region name
c:\>CICSCLI /c=cics region name /u=userid /p=password
```

The userid and password referenced in the command were defined in the target Windows NT-based CICS environment (see 13.4.5, “Define CICS user” on page 277).

We recommend, in this situation, that you make the userid and password uppercase. You can have problems with both of these being translated into uppercase, as CICS client attaches to the server depending on the setting for UpperCaseSecurity in the CICSCLI.INI file.

You may wish to apply resource level security to this user we have just defined in a similar way to what we did in 16.3.1.3, “Configuring for user login” on page 357.

16.3.2.2 Considerations

EZEUSRID will have the same value for all your users, so you would need to put some code in your Web Transactions to get the users to identify themselves, if you need to know. You could always add something to your UI Record for userid and password and login. The HTML INPUT TYPE=PASSWORD is not supported as one of the UI Record data item types, however, you can always tailor the JSP to change the HTML tag for your password field to have TYPE=PASSWORD.

You will also need to add code to your Web Transaction if you want to restrict access based on userid.

The benefit is that nothing needs to be done to define and maintain individual users within DCE or CICS, which may not be appropriate if there are large numbers of target users for the Web Transaction system (limited or unlimited potential user base).

16.3.3 Other options

If you simply require authentication to protect entry to the system from the browser end, you can protect the entry point JSP using IBM WebSphere Application Server facilities.

Since the Web Transaction code is all on the VisualAge Generator server, you have the choice of putting the authentication at the Web end or at the VisualAge Generator end; this allows you to determine how security and authentication will be implemented for your system.

If you choose to implement identification, validation, and authorization using WebSphere Application Server facilities, you will still need to find a way of identifying the end user in the Web Transaction program. The EZEUSRID EZEword will only return the CICS signon value, so you may need to design an approach that allows the WebSphere Application Server defined userid to be available to Web Transaction program logic.

For a more detailed review of security and Web browser access to CICS systems, see Chapter 9, "Security", in *Revealed! Architecting Web Access to CICS*, SG24-5466.

16.3.4 Secure HTTP

You can access the Gateway Servlet and hence your Web Transactions over HTTPS rather than HTTP. You just need to change the references to hptGatewayURL in the logon page, entry point page, and all your generated and deployed JSPs to hptSecureGatewayURL.

Appendix A. Sample code and other materials

The code and other materials developed during the residency project that produced this book are available for your use. The materials include:

- VisualAge Generator code
- DB2 database materials
- WebSphere Studio project archives
- Additional runtime files

The SG245636.EXE file packaged on the diskette is a self-extracting ZIP file with directories for the different materials included with this document.

The materials included on the diskette may be updated. Please check for updates in the SG245636 directory at the ITSO redbook materials ftp site at:

<ftp://www.redbooks.ibm.com/redbooks/>

The materials are provided as-is and with limited informal support. If you have questions regarding the materials, please place a question on the VisualAge Generator newsgroup (<news://news.software.ibm.com/ibm.software.vagen>) or send a note to the book owner, Pat McCarthy, at patmc@us.ibm.com.

A.1 VisualAge Generator code

We exported external source format (ESF) and .dat files for the Developer on Java development platform. When specific versions of the code are required, only the .dat file can be used as the source.

The VisualAge Generator Web Transaction programs implemented in this redbook are included in the following files in the VAGen directory:

- WebTran.dat** Projects and/or packages that contain the different Web Transaction systems discussed in Chapter 8, “Developing Web Transaction programming skills” on page 127.
- WebTran1.ESF** Entry point for the different Web Transaction systems discussed in Chapter 8, “Developing Web Transaction programming skills” on page 127.
- WebTran2.ESF** Solution for the different Web Transaction systems discussed in Chapter 8, “Developing Web Transaction programming skills” on page 127.

- VAGTWeb.dat** Project that contains the VisualAge Generator code generated from the VisualAge Generator Templates model defined in Chapter 9, “VisualAge Generator Templates Web Transactions” on page 173.
- DemoSys.dat** Packages that contain the different Web Transaction programs discussed in Chapter 10, “Demonstration system” on page 181.
- DemoSys.ESF** Source for the Web Transaction programs discussed in Chapter 10, “Demonstration system” on page 181.

The files listed above are Included in the VAGen directory.

A.2 WebSphere Studio

We exported WebSphere Studio archives (.WAR) files at several points during the development process outlined in Chapter 11, “Front-end customization techniques” on page 195. These are in the WStudio directory.

1st-try.war

An initial attempt at using WebSphere Studio.

JSPLevel2_VGBase.war

WebSphere Studio project at the point where the initial Level 2 setup was complete.

JSPLevel2_Ready4css.war

WebSphere Studio project at the point where the CSS exercise was ready to be done. Several of the default JSPs generated by VisualAge Generator have been added to the project.

JSPLevel2_CSSworkDone.war

WebSphere Studio project after the Level 2 exercise was complete.

JSPLevel3htmlint.war

WebSphere Studio project at the point where the Level 3 HTML site was ready to go (HTML design complete).

JSPLevel3htmljsp.war

WebSphere Studio project partway through Level 3 exercises.

JSPLevel3alljsp.war

WebSphere Studio project after Level 3 exercises.

A.3 Database

The VisualAge Generator server programs referenced in this book accessed one or more DB2 tables in the ITSOBank database.

The ITSOBANK database accessed by the ITSO Bank server programs is discussed in Appendix F, “ITSO Bank System Requirements and Database” on page 203.

The following files are included in the Database directory:

BankDB.bat	Command file to create database and load data into the tables. Edit file to adjust database create statements and creator ID value for create table statements.
BankDB.ddl	Data definition statements to create database and tables.
loaddata.sql	SQL statements to insert data into tables.
itsocat.txt	Catalog report of database tables.
itsodb.txt	SQL query for data in tables.

Use the following files to implement the state table referenced in Appendix F., “ITSO Bank System Requirements and Database” on page 203.

WTState.bat Command file to create table used to support state management in an `XFER ' ', UIRec` transfer statement.

WTState.ddl Data definition statements to create database and tables.

A.4 Additional materials

The directory Configs contains some additional files that we used during the development of this redbook.

Appendix B. Special notices

This publication is intended to help programmers and project leaders who will work with Web Transactions to develop new business application systems. The information in this publication is not intended as the specification of any programming interfaces that are provided by VisualAge Generator, VisualAge for Java, or VisualAge for Smalltalk. See the PUBLICATIONS section of the IBM Programming Announcement for VisualAge Generator, VisualAge for Java, and VisualAge for Smalltalk 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 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.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

IBM ®	AIX
AS/400	CICS
CICS/ESA	CICS/VSE
DB2	DB2 Universal Database
IMS	Language Environment
MQ	MQSeries
MVS/ESA	Netfinity
OS/2	OS/390
OS/400	RACF
RS/6000	S/390
System/390	Tivoli
TXSeries	VisualAge
VTAM	WebSphere

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. (For a complete list of Intel trademarks see www.intel.com/tradmarx.htm)

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through The Open Group.

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 C. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

C.1 IBM Redbooks publications

For information on ordering these ITSO publications see “How to get IBM Redbooks” on page 375.

- *WebSphere Studio and VisualAge for Java—Servlet and JSP Programming*, SG24-5755.
- *Revealed! Architecting Web Access to CICS*, SG24-5466
- *The Front of IBM WebSphere Building e-business User Interfaces*, SG24-5488

C.2 IBM Redbooks collections

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.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

C.3 Other resources

These publications are also relevant as further information sources:

- *CICS RACF Security Guide*, SC33-1701-02

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** <http://www.redbooks.ibm.com/>

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 IBM Redbooks fax order form to:

	e-mail address
In United States or Canada	pubscan@us.ibm.com
Outside North America	Contact information is in the "How to Order" section at this site: http://www.elink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site: http://www.elink.ibm.com/pbl/pbl

- **Fax Orders**

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: http://www.elink.ibm.com/pbl/pbl

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 employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at <http://w3.itso.ibm.com/> and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at <http://w3.ibm.com/> for redbook, residency, and workshop announcements.

Glossary

CSOGW.properties. The file that contains the definitions for how a Web Transaction program will be invoked by the Gateway Servlet. Similar to, but architecturally different from, a linkage table.

Gateway Servlet. The VisualAge Generator provided servlet that interacts with the generated JSPs, Java Beans, and Web Transactions. These components are generated for a UI Record used in a Web Transaction.

Java Server Page. Standardized language for building dynamic Web pages using Web servers and their associated application servers. A JSP can contain a mix of HTML and Java. The JSP source is compiled into a servlet that runs in the application server and produces dynamic Web pages.

linkage table. A VisualAge Generator control file that defines how a call to another program will be implemented. The call could be local, or part of a client/server environment. The linkage table is not used to control how a Web Transaction program is invoked, but can be used to control how a Web Transaction program calls a server program.

User Interface Type. Definition associated to a data item in a UI Record that controls how the data item will be presented in a Web browser.

User Interface Record (UI Record). A type of VisualAge Generator record that can be used in a CONVERSE or XFER statement to interact with a Web browser.

Web Transaction. VisualAge Generator program that can use a UI Record to interact with a Web browser.

Abbreviations and acronyms

APPC	advanced program-to-program communication	VAGen	VisualAge Generator
CICS	Customer Information Control System	WSRec	working storage record
CTG	CICS Transaction Gateway		
ECI	external call interface		
ESI	external security interface		
ESF	external source format		
ESM	external security manager		
IBM	International Business Machines Corporation		
IMS	Information Management System		
ITSO	International Technical Support Organization		
HTML	Hypertext Markup Language		
JDK	Java development kit		
JNI	Java Native Interface		
JSP	Java Server Page		
JVM	Java Virtual Machine		
MVS	Multiple Virtual Storage		
PEM	password expiration management		
RCT	resource control table		
SIDM	session ID manager		
UIRec	User Interface Record		
UI Type	User Interface Type		
URI	Universal Resource Indicator		
URL	Universal Resource Locator		

Index

A

- alias 318
- ANCHOR 6
- Apache 301
- APPC 296
- application 78, 264
- architecture
 - considerations 103
 - Web Transaction 5
- ASP 4
- ATTACHSEC 323
- authentication 354

B

- back button 52
- bean tag
 - application 78
 - class 78
 - in JSP 77
 - page 78
 - request 78
 - scope 78
 - session 78
 - type 78
- beans 32
- Block ID 281
- BODY 56, 198
- BORDER 202
- business object 175
- BUTTON 57, 60

C

- cascading style sheet
 - Master.css 222
 - use in JSP 222
- CCL3105 323
- CCL7053E 323
- CEDA 296
- CEDF 277, 351
- CELLPADDING 202
- CELLSPACING 202
- CHECKBOX 58
- CHECKED 58
- CICS
 - listener definition 275, 321

- login 360
- program definition 338
- security 299
- support for WebSphere Application Server 319
- Transaction Gateway 319
- WebSphere Test Environment 325
- CICS for NT
 - CEDF 277
 - CPMI 277
 - csogw.properties 279
 - mirror transaction 277, 279
 - runtime 250, 270
 - software
 - software requirements 250
 - TCP/IP listener 275
 - telnet 277
 - USER 277
 - XA definition 275
- CICS/ESA
 - connection 296
 - csogw.properties 299
 - implementation tasks 255
 - runtime 279
 - software requirements 254
- CICSCLI.INI 320
- CICSECI 266
- CICSTERM 322
- classes 31
- classpath
 - CICS Transaction Gateway 320
 - ctgserver.jar 320
 - runtime 341
 - WebSphere Application Server 305
 - WebSphere Test Environment 328
- CNOS Initialize 298
- comdtype 265
- complete state 89
- configuration
 - CICS Client 320
 - CICS for NT 270
 - CICS in WebSphere Application Server 319
 - CICS/ESA 279
 - code deployment 341
 - csogw.properties 263
 - GatewayServlet 45, 312, 314, 328
 - GIF 317
 - hpt.jar 306

- hptDateMask 315
- hptEntryApp 315
- hptEntryPage 315
- hptErrorLog 315
- hptErrorPage 315
- hptGateway.jar 306
- hptGatewayProperties 315
- hptIDManagerHost 315
- hptLinkageProperties 315
- hptLogonPage 315
- introduction 247
- Nanny path 320
- overview
 - CICS for NT 250
 - Windows NT 247
- security 352
- Vagen1EntryPage.jsp 342
- vgj.properties 318
- VisualAge Generator table 341
- Web Transaction runtime 257
- WebSphere Studio 211
- WebSphere Test Environment 324, 328
- Windows NT 268
- contable 265
- control point 289
- controlled state 91
- CONV_UI_RECORD.jsp 235
- conversation id 46
- CONVERSE 117
 - DXFR 89
 - runtime 47
 - state 87, 89, 150, 163
 - structure 85, 89, 128
 - TUI comparison 86
- CONVMOD 182, 184
- Corporate1 228
- CP name 281
- CPMI 277
- CSOERRORUIR.jsp 218, 317
- csogw.properties
 - application 264
 - CICS for NT 279
 - CICS server name 320
 - CICS/ESA 299
 - CICSECI 266
 - commtype 265
 - configuration 263
 - contable 265
 - destid 265
- GatewayServlet 45
 - groupid 265
 - hptErrorPackage 264
 - hptGateway 264
 - hptLinkageProperties 315
 - hptRuntimeProperties 264
 - javaProperty 265
 - location 265
 - overriding 266
 - propertiesRefreshInterval 264
 - serverid 265
 - serverLinkage 265, 266
 - TCP/IP 270
 - TCPIMS 266
 - TCPIP 266
 - Windows NT 270
- CSOTCPUI 344
- CSP 115
- CSTCNS 151
- CSTCNS2 163
- CSTCNV 128, 195
- CSTXB1 142
- CSTXBS 159
- CSTXBS2 168
- CSTXP 134
- CSTXP1 138
- CSTXP2 138
- CSTXPS 156
- CSTXPS2 165
- CTG.INI 320
- ctgclient.jar 320
- CTGJNI.DLL 320
- ctgserver.jar 320
- CUSTUI 129, 198, 222
- CUSTUI.jsp 235
- CUSTUI_I 145
- CUSTUI_IN 146
- CUSTUI_IO 143
- CUSTUI_O 145
- CUSTUI1 138
- CUSTUI2 139

D

- data item
 - help text 72
- data transfer
 - PROGRAMLINK 99
 - UI Record 98

- user interface type 100
- database
 - ITSOBANK 174
 - state 96
- date masks 318
- DB2
 - CICS attachment 271
 - WebSphere Application Server 301
- DB2DBDFT 268, 272, 275
- DB2INSTANCE 273
- definition
 - overview 9
 - UI Record 19
- demonstration system
 - components 181
 - CONVMOD 182, 184
 - data management 183
 - FRSTFRM 182, 185
 - FRSTPGM 182, 184
 - FRSTPLK 182, 185
 - Input validation 185
 - introduction 181
 - processing 182
 - state management 184
 - testing 186
 - transfer processing 183
- dependent LU 283
- design
 - considerations 98
 - data transfer 98, 100
 - development process 105
 - FORM 98
 - PROGRAMLINK 99
 - SUBMIT 99
 - user interface type 100
 - Web Transaction 85
- destid 265
- development
 - JSP modification 108
 - level 1 109
 - level 2 111
 - level 3 112
 - process 105
 - roles 106
 - skills 106
 - Web site 105
 - Web Transaction 18
- directive
 - errorPage 79

- import 79
- JSP 79
- DISABLED 58, 72
- docRoot 217, 327
- DXFR 89

E

- edit table 35
- edits 100
 - modification 101
 - processing 101
 - rules 100
 - UI Record 73
- elementNamed(String name) 80
- entry point 45
- environment variable
 - DB2DBDFT 268, 272, 275
 - DB2INSTANCE 273
 - EZERGRGL_xxx 268, 278
 - EZERGRGS_xxx 268, 278
 - EZERJULL_xxx 268, 278
 - EZERJULS_xxx 268, 278
 - EZERSQLDATE 268, 278
 - EZERSQLDB 268, 272, 275, 278
 - FCWDBNOOP 275
 - FCWDBPASSWORD 268, 272, 275
 - FCWDBPATH 268
 - FCWDBUSER 268, 272, 275
 - FCWDPATH 278
 - FCWRSC 268, 278
 - FCWTRDB_ 272, 275
 - FCWTROPT 268, 278
 - FCWTROUT 268, 278
 - HPTCLASSDIR 335
 - HPTJSPDIR 336
- Error 400 240
- error page
 - gateway servlet 45
 - JSP 79
- ERZ014016E 323
- ERZ042004E 323
- exercises
 - code base 128
 - CSTCNS 151
 - CSTCNS2 163
 - CSTCNV 128
 - CSTXB1 142
 - CSTXBS 159

CSTXBS2 168
 CSTXP 134
 CSTXP1 138
 CSTXP2 138
 CSTXPS 156
 CSTXPS2 165
 CUSTUI 129
 CUSTUI_I 145
 CUSTUI_IN 146
 CUSTUI_IO 143
 CUSTUI_O 145
 CUSTUI1 138
 CUSTUI2 139
 introduction 127
 program structure
 CONVERSE 128
 introduction 127
 XFER'' 142
 XFER program 134
 state management
 CONVERSE 150, 163
 introduction 150
 XFER'' 159, 168
 XFER program 155, 165
 VisualAge Generator Templates 173
 WTSPGM 168
 expression 76
 EZE Aid 71
 EZEAPP 93
 EZE DAYLC 318
 EZE DTELC 318
 EZE LTERM 47, 193
 EZE RGRGL_xxx 268, 278
 EZE RGRGS_xxx 268, 278
 EZE RJULL_xxx 268, 278
 EZE RJULS_xxx 268, 278
 EZE RSQLDATE 268, 278
 EZE RSQLDB 268, 272, 275, 278
 EZEUSR 47, 95, 193
 EZEUSRID 47, 193, 360

F

FACE 209
 FCWDBNOOP 275
 FCWDBPASSWORD 268, 272, 275
 FCWDBUSER 268, 272, 275
 FCWDPATH 268, 278
 FCWRSC 268, 278

FCWTRDB_ 272, 275
 FCWTROPT 268, 278
 FCWTROUT 268, 278
 fields, protecting 200
 FIELDSET 57, 60
 FILE 59
 folders 220
 FONT 209
 FORM
 customization 209
 default 98
 defined 98
 design 98
 HTML tag 57
 JSP customization 203
 protecting fields 200
 returning data 65
 TABLE 208
 UI Record definition 62
 user interface type 21, 65
 FRAME
 FRAMESET 233
 hptEntryApp 236
 hptEntryPage 236
 hptLogonPage 236
 top.html 235
 Web Transaction 234
 FRAMESET 233
 front-end
 customization 195
 development 107
 level 1 195
 presentation 107
 FRST_FRM_RECV_UI.jsp 235
 FRST_FRM_UI_RECORD.jsp 235
 FRST_PGM_UI_RECORD.jsp 235
 FRST_PLK_UI_RECORD.jsp 235
 FRSTFRM 182, 185
 FRSTPGM 182, 184
 FRSTPLK 182, 185
 FTP 259

G

GatewayServlet
 configuration 45, 328
 connecting to custom JSPs 241
 csogw.properties 45, 263
 definition 312

- edits 100
- entry point 45
- Error 400 240
- error page 45
- FRAME 235
- hptDateMask 315
- hptEntryApp 236, 315
- hptEntryPage 236, 237, 241, 315
- hptErrorLog 315
- hptErrorPage 315
- hptGatewayProperties 315
- hptIDManagerHost 315
- hptLinkageProperties 315
- hptLogonPage 236, 240, 315
- initialization parameters 314
- invocation 345
- logon page 45
- processing 44, 347
- PROGRAMLINK 69
- runtime 37
- top.html 235
- generation
 - FTP 259
 - HPTCLASSDIR 335
 - HPTJSPDIR 336
 - JSP 35
 - outputs 37
 - overview 10, 29
 - UI Record 33
 - VisualAge Generator Templates 177
 - Web Transaction 32, 335
- get() 80
- getAppID() 80
- getEditTableValues() 81
- getErrorMessage() 81
- getGatewayURL() 80, 81
- getHelpText() 80, 81
- getIndex() 81
- getLabel() 81
- getPageID() 80
- getSecureGatewayURL() 80
- getSession() 12
- getSessionID() 80
- getTextValue() 81
- getTextValuesTable() 81
- getTitle() 80
- GIF 317, 331
- groupid 265

H

- hasInputError() 80, 81
- HEAD 56, 196, 198
- HEIGHT 202
- help text
 - data item 72
 - UI Record 20
 - using in JSP 198
- HIDDEN 21, 59, 93, 200
- hpt.jar 306, 327
- hptAppld 239
- HPTCLASSDIR 335
- hptDateMask 315
- hptEntryApp 236, 315
- hptEntryPage 236, 237, 241, 315
- hptErrorLog 315
- hptErrorPackage 264
- hptErrorPage 315
- hptExec 239
- hptGateway 25
- hptGateway.jar 306, 326
- hptGatewayProperties 315
- hptGatewayURL 219
- hptIDManagerHost 315
- HPTJSPDIR 336
- hptLinkageProperties 315
- hptLogonPage 236, 240, 315, 352
- hptLogonPage hptEntryPage hptEntryApp 236
- hptRuntimeProperties 264
- HTML
 - ANCHOR 6, 68
 - basic structure 198
 - basics 55
 - BODY 56, 198
 - BUTTON 57, 60
 - cascading style sheet 222
 - CHECKBOX 58
 - CHECKED 58
 - customization 206
 - designer 113
 - DISABLED 58, 72
 - FACE 209
 - FIELDSET 57, 60
 - FILE 59
 - FONT 209
 - FORM 12, 26, 57, 209
 - FRAME 234
 - FRAMESET 233
 - HEAD 56, 196, 198

- HIDDEN 59, 200
- IMAGE 59
- index.html 232, 240
- INPUT 12, 26, 57, 200
- JavaScript 59
- JSP customization 196
- LABEL 57, 60
- layout 70
- LEGEND 57, 61
- LINK 223
- META 56
- OPTGROUP 57, 60
- OPTION 57, 60
- PASSWORD 58, 360
- RADIO 58
- READONLY 58, 72
- RESET 58
- SCRIPT 56
- scriptlet 75
- SELECT 57, 60
- SIZE 209
- STYLE 56
- SUBMIT 58
- TABLE
 - BORDER 202
 - CELLPADDING 202
 - CELLSPACING 202
 - customization 206
 - HEIGHT 202
 - WIDTH 202
- TARGET 68
- TEXTAREA 57, 59
- TITLE 56, 198
- top.html 231, 238
- UI Record mapping 20, 55
- VBScript 59
- WRAP 59
- httpd.cnf 318
- httpPort 217, 327
- HTTPS 361
- HttpServletRequest 12

I

- IBM HTTP 301
- IMAGE 59
- imports 195
- index.html 232, 240, 241
- index.jsp 241

- INPUT 21, 57, 200
 - edits 100
- INPUT/OUTPUT 21
 - edits 100
- instance 31
- interface bean
 - elementNamed(String name) 80
 - get() 80
 - getAppID() 80
 - getEditTableValues() 81
 - getErrorMessage() 81
 - getGatewayURL() 80, 81
 - getHelpText() 80, 81
 - getIndex() 81
 - getLabel() 81
 - getPageID() 80
 - getSecureGatewayURL() 80, 81
 - getSessionID() 80
 - getter and setter methods 80
 - getTextValue() 81
 - getTextValuesTable() 81
 - getTitle() 80
 - hasInputError() 80, 81
 - isDisplayable() 81
 - isEmpty() 82
 - isSelected() 82
 - Java API 79
 - occurrences() 82
 - set(String value) 80
 - subElements() 82
 - VGDataElement 81
- interface unit 176
- isDisplayable() 81
- isEmpty() 82
- isSelected() 82
- ITSOBANK 174

J

- Java
 - bean tag 77
 - beans 32
 - classes 31
 - data bean 33
 - expression 76
 - instance 31
 - interface bean 33, 34
 - Java Virtual Machine 31
 - packages 31

- PrintWriter 75
- resource bundle 37
- terminology 30
- JAVA_HOME 301
- javaProperty 265
- JavaScript 55, 59
 - ANCHOR 68
- javascript 199
 - index.jsp 242
 - location 242
- JDK 301
- JSP
 - application 78
 - bean tag 77
 - cascading style sheet 222
 - correcting errors 195
 - CSOERRORUIR.jsp 317
 - customization
 - adding to WebSphere Studio 217
 - advanced techniques 206
 - cascading style sheet 222
 - CSTCNV 195
 - FONT 209
 - FORM 203, 209
 - FRAME 234
 - GatewayServlet 241
 - hptAppIId 239
 - hptEntryApp 236
 - hptEntryPage 236, 237, 241
 - hptExec 239
 - hptLogonPage 236, 240
 - implementing help 198
 - imports 317
 - index.html 240, 241
 - index.jsp 241
 - interleaved TABLE and FORM 208
 - introduction 195
 - javascript 199, 242
 - level 1 195
 - level 2 205
 - level 3 226
 - presentation 201
 - protecting form fields 200
 - rules 205
 - suggested changes 196
 - TABLE 202, 206
 - top.html 235, 238
 - Vagen1EntryPage.jsp 342
 - WebSphere Studio 210, 216

- deploy 317
- developer
 - level 2 112, 205
 - level 3 114
- directive 79
- errorPage 79
- errors 317
- Execution Monitor 332
- expression 76
- front-end customization 195
- generated for UI Record 35
- import 79
- imports error 195
- interface bean 75
- JavaScript 55
- javascript 199
- modification 107, 108, 123
- page 78
- Page Compile Generated Code 334
- presentation 201
- request 78
- scriptlet 75
- session 78
- support for 1.0 330
- user interface type 21
- VBScript 55
- WebSphere Test Environment 195, 331
- JSP presentation 107

K
key ring 302

L
LABEL 57, 60
LEGEND 57, 61
level 1

- front-end development 195
- introduction 109

level 2

- front-end development 205
- HTML designer 113
- introduction 111
- JSP developer 112
- Web Transaction developer 111

level 3

- FRAME 234
- front-end development 226
- introduction 112

JSP developer 114
Web Transaction developer 113
LINK 223
listener definition 321
location 242, 265
logon page 45

M

Master.css 222
META 56
Microsoft Internet Explorer 211
Microsoft Peer Web Services 259
mirror transaction 277, 279

N

Nanny path 320
NetName 320
NONE 21, 71, 200

O

occurrences() 82
OPTGROUP 57, 60
OPTION 57, 60
OUTPUT 21, 57, 63

P

packages 31
page 78
parser 219, 221
PASSWORD 58
password 302, 352, 360
performance 306
Personal Communications 280
physical unit 281
PrintWriter 75
PROGRAMLINK
 GatewayServlet 69
 images 70
 passed data 69
 TUI migration 120
 user interface type 21, 68
propertiesRefreshInterval 264
pseudo-conversational 5, 8, 86
PU 281
publishing 212, 230

R

RADIO 58
READONLY 58, 72, 201
request 78
RESET 58
RSL 357
runtime
 CEDF 351
 CICS for NT 250, 270
 CICS/ESA 279
 code deployment 341
 csogw.properties 263
 CSOTCPUI 344
 edit table 35
 GatewayServlet 37, 44, 347
 hptLogonPage 352
 implementation tasks
 CICS for NT 253
 CICS/ESA 255
 Windows NT 250
 overview 10, 29, 247
 program 32
 resource clean up 53
 security 352
 session ID manager 37, 46, 344
 software requirements
 CICS for NT 250
 CICS/ESA 254
 Windows NT 247
 system implementation 36
 system start up 344
 user message table 36
 Vagen1LogonPage.jsp 352
 VisualAge Generator table 341
 web browser 37
 Web Transaction 47
 CONVERSE 47
 setup 257
 tasks 341
 XFER '' 51
 XFER program 47
 Windows NT 247, 268

S

scope 78
SCRIPT 56
scriptlet 75
security

- authentication 354
- CICS 323
- considerations 352
- EZEUSERID 360
- HTTPS 361
- RSL 357
- transparent CICS login 360
- SELECT 57, 60
- SERunner.properties 327
- serverid 265
- serverLinkage 265, 266
- session 78
- session data 227
- session ID manager
 - conversation id 46
 - EZELTERM 47
 - EZEUSR 47
 - EZEUSRID 47
 - runtime component 37
 - runtime role 46
 - starting 318, 344
 - user id 46
- set(String value) 80
- SIZE 209
- software
 - Apache 301
 - CICS for NT
 - DB2 301
 - DB2 client 257
 - IBM HTTP 301
 - JDK 301
 - Microsoft Peer Web Services 259
 - Personal Communications 280
 - TXSeries
 - VisualAge for C++ 259, 302
 - VisualAge Generator Common Services 259, 302
 - VisualAge Generator Server 259, 302
 - Web server 301
- state
 - complete 89
 - controlled 91
 - conversation 94
 - CONVERSE 87, 89
 - data 42
 - EZEUSR 95
 - global 94
 - HIDDEN 93
 - implementation 94
 - management
 - CONVERSE 150, 163
 - demonstration system 184
 - exercises 150, 155, 159, 163, 165, 168
 - implementation 168
 - using UI Record 150
 - using working storage record 163
 - WTSPGM 168
 - XFER '' 159, 168
 - XFER program 155
 - options 87
 - self-managed
 - implementation 94
 - program design 97
 - XFER '' 93
 - stateless 93
 - timestamp 95
 - XFER '' 87, 93, 94
 - XFER program 87, 91
- stateless 93
- structure
 - CONVERSE 85, 89
 - options 89
 - Web Transaction 85
 - XFER '' 85, 89
 - XFER program 85, 89
- STYLE 56
- subElements() 82
- SUBMIT 63
 - button in HTML 58
 - user interface type 21, 99
- SUBMIT VALUE ITEM 99
- submit value item 20
- SUBMITBYPASS 21, 63

T

TABLE

- BORDER 202
- CELLPADDING 202
- CELLSPACING 202
- customization 206
- FORM 208
- HEIGHT 202
- TD 202
- TR 202
- WIDTH 202

table

- VisualAge Generator 341

- TCP/IP 275
- TCPIIMS 266
- TCPIP 266
- TD 202
- telnet 277
- templates
 - index.html 232
 - top.html 231
- templatesWebSphere Studio 228
- terminology
 - Java 30
- testing
 - default HTML 21
 - overview 9, 27
 - WebSphere Test Environment 195
- TEXTAREA 57, 59
- timestamp 95
- TITLE 56, 198
- top.html 231, 235, 238
- TR 202
- TUI
 - comparison to Web Transaction 86
 - converting to Web Transactions 115
 - creating UI Record from MAP 117
- TWASIZE 299
- TXSeries
 - See CICS for NT*

U

- UI Record
 - creating from MAP 117
 - CUSTUI 129, 198, 222
 - CUSTUI_I 145
 - CUSTUI_IN 146
 - CUSTUI_IO 143
 - CUSTUI_O 145
 - CUSTUI1 138
 - CUSTUI2 139
 - definition 19
 - edit table 35, 65
 - edits 73, 100, 101
 - EZEAIID 71
 - FORM definition 62
 - forms 25
 - generated JSP 35, 55
 - generation 33
 - help 198
 - help text 20, 71

- introduction 7
- Java data bean 33
- Java interface bean 33, 34
- javascript 199
- mapping to HTML 20, 22, 55
- PROGRAMLINK 99
- properties 71
- state management 150
- submit value item 20, 71
- user interface type 21
- VGDataElement 81
- WebSphere Test Environment 214
- UpperCaseSecurity 320
- user interface type 21
 - data management 100
 - edits 73, 100
 - FORM 21, 65, 98
 - HIDDEN 21, 62, 93, 200
 - INPUT 21, 62
 - INPUT/OUTPUT 21, 62
 - NONE 21, 71, 200
 - OUTPUT 21, 57, 63
 - PROGRAMLINK 21, 68
 - SUBMIT 21, 63, 99
 - submit bypass 72
 - SUBMIT VALUE ITEM 99
 - SUBMITBYPASS 21, 63
- user message table 36
- userid 302, 352

V

- Vagen1EntryPage.jsp 218, 237, 342
- Vagen1ErrorPage.jsp 218
- Vagen1LogonPage.jsp 218, 352
- vawcg-wp.gif 218, 221
- VBScript 55, 59
- VGDataElement interface 81
- vgj.properties 318
- visage.gif 218, 221
- VisualAge for C++ 259, 302
- VisualAge for Java
 - components 325
 - features 325
- VisualAge Generator
 - table 341
 - V4 overview 3
- VisualAge Generator Common Services 259
- VisualAge Generator Server 259

VisualAge Generator Templates 173
 business object 175
 customization 179
 generation 177
 interface unit 176
VTAM 296

W

web browser
 back button 52
Web site
 development 105
 planning 227
 WebSphere Studio 228
web system
 options
 ASP 4
 Java servlet and JSP programming 12
 JSP 4
 servlet 4
 Web Transaction 13
 processing
 application server 38
 Java servlet 38
 JSP 40
 state data 42
 steps 41
Web Transaction
 architecture 5, 103
 CSTCNS 151
 CSTCNS2 163
 CSTCNV 128, 195
 CSTXB1 142
 CSTXBS 159
 CSTXBS2 168
 CSTXP 134
 CSTXP1 138
 CSTXP2 138
 CSTXPS 156
 CSTXPS2 165
 definition 9, 18, 26
 demonstration system 181
 design 85, 98, 103
 developer
 level 2 111
 level 3 113
 development
 level 1 109

 level 2 111
 level 3 112
 process 105
 roles 106
 simultaneous 226
 Web site 105
edits 101
exercises
 code base 128
 CONVERSE 128
 introduction 127
 program structure 127
 state management 150
 VisualAge Generator Templates 173
 XFER'' 142
 XFER program 134
FRAME 234
generation 10, 29, 32
introduction 3
programming model 7
running 341
runtime
 CICS for NT 250, 270
 CICS/ESA 279
 configurations 14
 CONVERSE 47
 generation for 29
 overview 10, 247
 scenario 47
 setup 257
 tiers 14
 Windows NT 247, 268
 XFER'' 51
 XFER program 47
simultaneous development 226
state 87
 complete 89
 controlled 91
 stateless 93
state management
 CONVERSE 150, 163
 XFER'' 159, 168
 XFER program 155, 165
structure 89
 options 26
 state 87
support for
 definition 9
 generation 9

- runtime 9
- testing 9
- testing 9, 27
- top.html 238
- TUI 115
- VisualAge Generator Templates 173
- WebSphere
 - application server 4
- WebSphere Application Server
 - Application Server 304
 - CICS support 319
 - classpath 305
 - GatewayServlet 312
 - GIF 317
 - installation 302
 - password 302
 - performance 306
 - startup 303
 - userid 302
 - WebSphere Studio 211
- WebSphere Page Designer
 - corrected the errors 208
 - templates 231
- WebSphere Studio
 - adding JSPs 217
 - configuration 213
 - Corporate1 228
 - CSOERRORUIR.jsp 218
 - folders 220
 - hptGatewayURL 219
 - index.html 232
 - JSP
 - customization 210
 - modification 216
 - suggested changes 196
 - Microsoft Internet Explorer 211
 - parser 221
 - project 217
 - project creation 216
 - publishing 212, 216, 230
 - setup and configuration 211
 - templates 228
 - top.html 231
 - use parser 219
 - Vagen1EntryPage.jsp 218
 - Vagen1ErrorPage.jsp 218
 - Vagen1LogonPage.jsp 218
 - vawcg-wp.gif 218, 221
 - visage.gif 218, 221
 - Web site development 228
 - WebSphere Application Server 211
 - WebSphere Test Environment 211, 214, 216
 - WebSphere Test Environment
 - classpath 328
 - docRoot 217, 327
 - GatewayServlet 328
 - generated UI Records 214
 - GIF 331
 - hpt.jar 327
 - hptGateway.jar 326
 - httpPort 217, 327
 - introduction 324
 - JSP 331
 - JSP 1.0 330
 - JSP customization 195
 - publishing support 216
 - SERunner.properties 327
 - setup 324
 - WebSphere Studio 211, 214, 216
 - WIDTH 202
 - Windows NT
 - csogw.properties 270
 - runtime 247, 268
 - software requirements 247
 - working storage record
 - state management 163
 - WRAP 59
 - WTSPGM 168

X

 - XA definition 275
 - XFER ''
 - EZEAPP 93
 - runtime 51
 - state 87, 93, 94, 159, 168
 - structure 85, 89, 142
 - XFER program
 - runtime 47
 - state 87, 91, 155, 165
 - structure 85, 89, 134
 - XID 296

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at ibm.com/redbooks
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number	SG24-5636-00
Redbook Title	Building Enterprise Web Transactions using VisualAge Generator JavaBeans and JSPs
Review	
What other subjects would you like to see IBM Redbooks address?	
Please rate your overall satisfaction:	<input type="radio"/> Very Good <input type="radio"/> Good <input type="radio"/> Average <input type="radio"/> Poor
Please identify yourself as belonging to one of the following groups:	<input type="radio"/> Customer <input type="radio"/> Business Partner <input type="radio"/> Solution Developer <input type="radio"/> IBM, Lotus or Tivoli Employee <input type="radio"/> None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	<input type="checkbox"/> Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. ibm.com/privacy/yourprivacy/



Redbooks

Building Enterprise Web Transactions using VisualAge Generator JavaBeans and JSPs

0.5" spine 250
<-> 459 pages



Building Enterprise Web Transactions

using VisualAge Generator JavaBeans and JSPs



Redbooks

Develop Web Transaction design and programming skills

This redbook explains how VisualAge Generator V4 can be used to implement Web-based transaction systems that access enterprise server platforms. The implementation process includes Web Transaction program development, generation of JavaBeans, JSPs, and programs for the selected target runtime environment, and configuration of the runtime system.

Customize generated JSPs and perform Web-site integration

In Part 1 we describe how VisualAge Generator uses the Web Transaction programming model to implement Web-based systems. The inner workings of the generated JSPs and JavaBeans and the VisualAge Generator GatewayServlet are fully described and contrasted with hand-crafted JSP and servlet systems.

For IBM WebSphere, Windows NT, TX Series, and CICS/ESA

The exercises in Part 2 allow you to develop an advanced understanding of how Web Transactions work, supported system structure options, state management implementation options, and how VisualAge Generator Templates can be used to rapidly implement fully functional Web Transaction systems.

In Part 3 you learn how to customize the generated JSPs, provide support for cascading style sheets (CSS) in a Web Transaction system, and integrate HTML-based Web sites and the generated JSPs for Web Transactions. An integrated approach for the use of WebSphere Studio and the WebSphere Test Environment provided by VisualAge for Java is also defined.

Part 4 provides you with a step-by-step guide to the process of installing, configuring, building, and running a Web Transaction system using the IBM WebSphere Application Server platform on Windows NT and Web Transactions running on Windows NT, TX Series (Windows NT) and CICS/ESA.

After reading the redbook you will fully understand how to use VisualAge Generator to build and implement an IBM WebSphere Application Server-based system that accesses enterprise transactions and data.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by IBM's International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-5636-00

ISBN 0738416495