IBM VSE/Enterprise Systems Architecture

VSE/ESA e-business Connectors User's Guide

Version 2 Release 7

SC33-6719-04

IBM VSE/Enterprise Systems Architecture

VSE/ESA e-business Connectors User's Guide

Version 2 Release 7

SC33-6719-04

Note!
Before using this information and the product it supports, be sure to read the general information under "Notices" on page xv.

Fifth Edition (March 2003)

This edition applies to Version 2 Release 7 of IBM Virtual Storage Extended/Enterprise Systems Architecture (VSE/ESA), Program Number 5690-VSE, and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the addresses given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Deutschland Entwicklung GmbH
Department 3248
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany

You may also send your comments by FAX or via the Internet:

Internet: s390id@de.ibm.com
FAX (Germany): 07031-16-3456
FAX (other countries): (+49)+7031-16-3456

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, 2003. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures ix

Tables xiii

Notices xv
Trademarks and Service Marks xv

About This Book xvii
Who Should Use This Book xvii
How to Use This Book xvii
Where to Find More Information xix

Summary of Changes xxi
Changes for Fifth Edition (March 2003) xxi
Changes for Fourth Edition (June 2002) xxii
Changes for Third Edition (December 2001) . . . xxii

Part 1. Introduction 1

Chapter 1. Introduction to e-business with VSE/ESA 3
What the VSE/ESA e-business Connectors Provide . 4
Overview of the Java-Based Connector 4
Overview of the DB2-Based Connector 6
Overview of the VSAM Redirector Connector . . 6
Overview of the VSE Script Connector. 7
Overview of VSE/ESA Support for Web Services and SOAP 8
Overview of CICS Connectivity 9
Overview of MQSeries Connectivity 9
Overview of the IBM WebSphere Application Server 9

Chapter 2. Overview of 2- and 3-Tier Environments 11
Overview of 2-Tier Environments 12
Overview of 3-Tier Environments 13

Part 2. Installation & Customization 15

Chapter 3. Choosing the Connectivity You Require 17
Connectivity Possibilities in 2-Tier Environments . . 17
Connectivity Possibilities in 3-Tier Environments . . 18

Chapter 4. Installing the Common Prerequisite Programs 21
Configuring and Activating TCP/IP for VSE/ESA 21
Configuring and Activating the VSE HTTP Server 21
Installing and Configuring Java. 21
Downloading the Java Base Code 22
Deciding Which Java Package to Install 22
Installing the IBM HTTP Server 22
Installing the WebSphere Application Server . . . 23
Installing the WebSphere Application Server on z/OS 23
Installing the WebSphere Application Server on Other Platforms 23

Chapter 5. Installing and Operating the Java-Based Connector 25
Installing the VSE Connector Client 25
Obtaining a Copy of the VSE Connector Client 25
Performing the VSE Connector Client Installation 26
Using the Online Documentation Options . . . 28
Configuring for WebSphere Support 29
Uninstalling the VSE Connector Client 29
Configuring the VSE Connector Server 30
Job SKVCSSTJ – Startup Job 30
Job SKVCSCAT – Catalog Members 31
VSE Library Member SKVCSCFG – General Settings. 32
VSE Library Member SKVCSLIB – Specify Libraries to Be Accessed 33
VSE Library Member SKVCSPLG – Specify Plugins to Be Loaded 33
VSE Library Member SKVCSUSR – Specify Logon Access. 34
VSE Library Member SKVCSSSL – Configure for SSL 35
Starting the VSE Connector Server. 36
Testing the Communication Between VSE Connector Client and Connector Server. 37
Obtaining a List of VSE Connector Server Commands 38
Entering a Command for the VSE Connector Server 38
Maintaining Security Using the VSE Connector Server 38

Chapter 6. Configuring DL/I for Access Via VSE Java Beans 41
Host Installation Activities That Must Be Already Completed 41
Step 1: Skeleton SKDLISMP – Define Sample Database 42
Step 2: Customize CICS TS 42

Chapter 7. Installing the VSE Script Connector 43
Step 1: Download the Install-File and Perform the Installation 43
Step 1.1: Obtain a Copy of the VSE Script Server 43
Step 1.2: Perform the Installation of the VSE Script Server 44
Step 2: Configure the VSEScriptServer Properties File 45
Step 3: Configure the Connections Properties File. . 46

© Copyright IBM Corp. 2000, 2003

Chapter 8. Installing the VSAM Redirector Connector 47
How the VSAM Redirector Connector Works . . . 47
VSAM Integration Considerations 48
Installing and Configuring the VSAM Redirector Client 50
Step 1: Enable the VSAM Redirector Client on VSE/ESA 50
Step 2: Decide Upon Your Redirection Mode . . 51
Step 3 (Optional): Transfer Your VSAM Data . . 56
Step 4: Create the Configuration Phase 56
Installing the VSAM Redirector Server 60
Step 1: Download the Install-File and Perform the Installation 60
Step 2: Configure the Properties File 61
Step 3: Implement a VSAM Request Handler . . 62
IBM-Supplied Example of DB2-Related Handler . . 64
IBM-Supplied Example of HTML-Related Handler 65
Code for HTML-Related Request Handler . . . 65

Chapter 9. Customizing the DB2-Based Connector 71
Host Installation Activities That Must Be Already Completed 71
Step 1: Customize CICS TS 72
Step 2: Customize TCP/IP 72
Step 3: Customize DB2 and Define Sample Database 72
Step 3.1: Define User Catalog 73
Step 3.2: Catalog New ARISIVAR.Z 73
Step 3.3: Job Manager for Preparation / Installation Steps 74
Step 3.4: Activate DRDA Server Support. 75
Step 3.5: Startup Job for Stored Procedure Server 75
Step 3.6: Prepare DB2 Sample Database 75
Step 3.7: Install DB2 Sample Database 77
Step 4: Set Up for DRDA Support 80
Step 5: Set Up Stored Procedure Server and Define to DB2 81
Step 5.1: Set Up the Stored Procedure Server . . 81
Step 5.2: Define Stored Procedure Server to DB2 82
Step 6: Set Up for Stored Procedures 82
Step 7: Customize the DB2-Based Connector for VSAM Data Access 83
Step 8: Customize the DB2-Based Connector for DL/I Data Access 83
Step 9: Start DB2, and Start Stored Procedure Server 84
Step 10: Install DB2 Connect and Establish Client-Host Connection 85

Chapter 10. Configuring the VSAM-Via-CICS Service 87
Configuring the IBM-Supplied CICS System . . . 87
Configuring a Further CICS System for VSAM-Via-CICS 88
How the VSAM-Via-CICS Service Works 89
CICS Transactions for Use with VSAM-Via-CICS . . 89

Chapter 11. Configuring Your VSE/ESA Host for SSL 91
Configuring for SSL Using IBM-Supplied Keys/Certificates 92
Step 1: Activate TCP/IP for VSE/ESA 92
Step 2: Catalog Keyring Set Into the VSE Keyring Library 92
Configuring for SSL Using Your Own Keys/Certificates 95
Step 1: Activate TCP/IP for VSE/ESA 95
Step 2: Install/Configure Utility CIALCLNT On a Web Client 95
Step 3: Generate a Key Pair, Request a Server Certificate 95
Step 4: Obtain a Signed Server Certificate and Copy to Job CIALCERT 98
Step 5: Obtain a Root Certificate and Copy to Job CIALROOT. 101
Step 6: Verify Your Certificates on the Host . . 103
Step 7: Secure Your VSE Keyring Library Entries 103
SSL Examples Provided With the Online Documentation 104

Chapter 12. Configuring the Java-Based Connector for Server Authentication 105
Configuring the VSE Connector Server for Server Authentication 105
Step 1: Configure and Catalog the VSE Connector Server's SSL Profile. 105
Step 2: Activate SSL Profile in Main Configuration File 106
Configuring the VSE Connector Client for Server Authentication 108
Step 1: Set SSL Flag in Class VSEConnectionSpec 108
Step 2: Configure SSL Profile 109
Step 3: Copy a Server Certificate Into Client Keyring File 110
Description of the IBM-Supplied Client Keyring File 111
Currently-Supported SSL Cipher Suites. 112

Chapter 13. Configuring the Java-Based Connector for Client Authentication 115
Configuring the VSE Connector Server for Client Authentication 116
Configuring the VSE Connector Client for Client Authentication 117
Step 1: Generate a Key Pair. 117
Step 2: Generate and Store a Client Certificate 118
Step 3: Import the CA's Root Certificate into the Client Keyring File 120
Step 4: Save Your Client Keyring File 121
Step 5: Define Access Rights for VSE Connector Client to Use Host Resources 121

Chapter 14. Service Functions for Client Authentication 123
Prerequisites. 123

Using the Batch Service Function BSSDCERT . . . 123
Changing the Defaults (Optional). 125
Using the Client-Certificates/User-IDs Dialog . . 125
Step 1: Starting the Dialog 125
Step 2: Selecting an Option 126
Step 3: Creating the Output Job 127
Step 4: Submitting or Storing the Output Job 128

Chapter 15. Mapping VSE/VSAM Data to a Relational Structure 129
Introduction to Mapping VSE/VSAM Data . . . 129
How VSAM Maps Are Structured 130
How Maps Are Stored on the VSE/ESA host . . . 130
Defining a Map Using RECMAP 132
Defining a Map Using the Sample Applet 134
Defining a Map Using a Java Application 134
Defining a Map Using the VSAM MapTool . . . 140

Part 3. Programming 143

Chapter 16. Migrating Your Programs 145
Migrating from CCF to CCI 145
Migrating to Secure Connections Using SSL . . . 147
Migrating to VSAM-Access Via JDBC 149
Migrating Your Applets to JDK 1.3 or Later . . . 151
Using the New Methods in VSE Java Beans . . . 152
Migrating Servlets and EJBs 152

Chapter 17. Using VSE Java Beans to Implement Java Programs 153
Where VSE Java Beans Are Installed and Used . . 153
How JavaBeans and EJBs Compare to VSE Java Beans 154
Contents of the VSE Java Beans Class Library . . 155
Example of a Javadoc for a VSE Java Bean . . . 158
Using the Callback Mechanism of VSE Java Beans 159
Using VSE Java Beans to Connect to a VSE/ESA host 162
Using VSE Java Beans for Accessing VSAM Data 164
Using VSE Java Beans for Accessing DL/I Data 167
Using VSE Java Beans for Accessing VSE/POWER Data 172
Using VSE Java Beans for Submitting Jobs. . . . 173
Using VSE Java Beans for Accessing Librarian Data 175
Using VSE Java Beans for Accessing VSE/ICCF Data 177
Using VSE Java Beans for Accessing the Operator Console 180
Using the VSE Navigator Application 182
Prerequisite for Using the VSE Navigator . . . 183
Migrating From Earlier Versions 183
Installing the VSE Navigator 183
Starting the VSE Navigator Client 183
Adding Your Own VSE Navigator Plug-Ins . . 184

Chapter 18. Using JDBC to Access VSAM Data 187
SQL Statements That Are Supported by JDBC . . 187
Relational and VSE Java Beans Terminology . . . 190
Specifying Table Names 190
Example of Using JDBC to Access VSAM Data . . 191
Step 1. Include the Imports and Class Definition 191
Step 2. Perform the doGet() Method 192
Step 3. Display List of Available Flights . . . 193
Step 4. Display Flight Properties and Entry Form 194
Step 5. Get and Display the Data Field Values 195
Step 6. Update the VSAM Cluster 196
Step 7. Get Order Values and Check Availability 197
Step 8. Create the New Order 198
Step 9. Check Processing and Increase Reserved Seats 199

Chapter 19. Using Java Applets to Access Data 201
How Applets Are Used in 2-Tier Environments . . 202
How Applets Are Used in 3-Tier Environments . . 203
How the VSEAppletServer Is Used 205
Disadvantages and Restrictions Of Using Applets 206
Running the Sample Data-Mapping Applet . . . 207
Description of the Data-Mapping Applet . . . 207
Activities Required on the VSE/ESA Host . . . 208
Deploying the Data-Mapping Applet 209
Calling the Data-Mapping Applet 209
Setting Up the Data-Mapping Applet Class . . 210
Initializing the Data-Mapping Applet 210
Re-Displaying or Leaving an HTML Page . . . 211
Using the Data-Mapping Applet to Add a Map to a VSAM Cluster 211
Using the Data-Mapping Applet to Modify a Map 212
Using the Data-Mapping Applet to Modify a Map's Data Fields 213
Running the Data-Mapping Applet Locally Using the AppletViewer 215
Running the Sample VSAM Applet 216
Description of the VSAM Applet 216
Getting Started With the Sample VSAM Applet 218
Calling the VSAM Applet 221
Description of DB2ConnectorJDBCApplet.java (the Client-Side Program) 223
Description of VSAMSEL 226
Running the Sample DL/I Applet 230
Description of the DL/I Applet 230
Getting Started With the Sample DL/I Applet 232
Calling the DL/I Applet. 233
Description of DB2DLIConnectorJDBCApplet.java (the Client-Side Program) 235
Description of DLIREAD 238

Chapter 20. Using Java Servlets to Access Data 243
How Servlets Are Used in 3-Tier Environments . . 243
Compiling and Calling Servlets 245
How the WebSphere Application Server Stores Session Information 245
Example of How to Implement a Servlet 246
General Description of the Sample Servlet . . . 246

Creating the VSAM Clusters for the Sample . . 247
HTML Constructs Used With the Sample . . . 247

Chapter 21. Using Java Server Pages to Access Data 263
How JSPs Are Used in 3-Tier Environments . . . 263
Example of a Simple Java Server Page 265

Chapter 22. Using EJBs to Represent Data 267
Overview of the EJB Architecture 267
Overview of How EJB Containers are Used . . 268
How EJBs Compare to JavaBeans / Java Servlets 269
Implementing Your Client Applications 270
How an EJB Client Accesses EJBs 271
Example of Using EJBs to Access VSAM Data . . 273
Example of Implementing VSAM-Based EJBs . . . 274
Step 1: Define the Sample's VSAM Cluster . . 275
Step 2: Create the Record Layout for Employees 275
Step 3: Specify the EJB's Home Interface . . . 276
Step 4: Specify the EJB's Remote Interface . . . 276
Step 5: Implement the RecordPK Class 277
Step 6: Implement the EJB Code 277
Step 7: Compile the Java Source Files 282
Step 8: Deploy the EJBs 282
Step 9: Access the EJBs from an EJB Client . . 283

Chapter 23. Extending the Java-Based Connector 287
Implementing a Server Plugin 288
Implementing a PluginMainEntryPoint Function 292
Implementing a SetupPlugin Function 293
Implementing a CleanupPlugin Function . . . 294
Implementing a GetHandledCommands Function 295
Implementing a SetupHandler Function . . . 296
Implementing an ExecuteHandler Function . . 297
Implementing a CleanupHandler Function . . 299
Creating Your Own Plugin Callback Functions 299
Action Codes Supported by the VSE Connector Server 300
Utility Functions Supported by the VSE Connector Server 301
Using the IBM-Supplied Server Plugin Example 302
Registering and Compiling Your Server Plugin 302
Implementing a Client Plugin 303
Using the VSEPlugin class 303
General Considerations When Designing Your Plugin 305
Specifying the Protocol Between VSE Connector Server and Plugin 305
Choosing the Access Method to the Data / Application 306
Considerations for ASCII / EBCDIC and Big / Little Endian 306
Deciding Which Requests / Functions Should Be Supported 306
Transferring Data Over the Network 307
Structuring the Client Plugin's View 307

Chapter 24. Using the DB2-Based Connector to Access Data 309
How You Use DB2 Stored Procedures 310
Grouping Stored Procedure Servers 310
Programming Requirements When Using DB2 Stored Procedures 311
Using DB2 Stored Procedures to Access VSAM Data 312
Overview: Accessing VSAM Data via DB2 Stored Procedures 313
Using the Call Level Interface: Activities on the Requestor 314
Using Call Level Interface: Activities on the VSE/ESA host 315
Example of the Syntax of a CLI Function – VSAMSQLCloseTable 316
Program Flow When Using the VSAMSQL Call Level Interface 317
SQL Statements Supported by VSAMSQL Call Level Interface 318
Using DB2 Stored Procedures to Access DL/I Data 320
Overview of the AIBTDLI Interface 321
Creating Programs That Use AIBTDLI 323
Invoking the AIBTDLI Interface 324
Compiling and Link-Editing Your Programs . . 326
Return and Status Codes 326
Scheduling with Single and Multiple MPS Systems 328
Task Termination and Abend Handling 329
Messages and Return Codes 329

Chapter 25. Using SOAP for Inter-Program Communication 331
Overview of the SOAP Syntax 331
How the VSE/ESA Host Can Act As the SOAP Server 332
How the VSE/ESA Host Can Act As the SOAP Client 334
How the IBM-Supplied SOAP Control Blocks Are Used 335
How the SOAP_PARAM_HDR Control Block Is Used 335
How the SOAP_PROG_PARAM Control Block Is Used 337
How the SOAP_DEC_PARAM Control Block Is Used 338
Description of the IBM-Supplied SOAP Service (getquote.c) 339
Description of the IBM-Supplied SOAP Client (soapclnt.c) 341
Using a Java SOAP Client 343
Running the IBM-Supplied SOAP Sample 344
Step 1: Download and Install the Java SOAP Client Packages on the Client 344
Step 2: Extract and Install the Required Java Programs 344
Step 3: Compile /Link the Sample C Programs, and Define Them to CICS 345
Step 4: Define the SOAP Server to CICS . . . 346
Step 5: Activate the ASCII to EBCDIC Converter 346

Step 6: Compile the Java Sample 346
Step 7: Run the Java SOAP Client Sample . . . 347
Step 8: Run the C-Program SOAP Client Sample 347
Writing Your Own SOAP Programs 348

Chapter 26. Using the VSE Script Connector for Non-Java Access . . . 349
How the VSE Script Connector Is Used. 349
Overview of the Protocol Used Between Client and Server 350
Writing VSE Scripts Using the VSE Script Language. 351
General Rules That Apply to the VSE Script Language. 351
VSE Script Language Built-In General Functions 352
VSE Script Language Built-In String Functions 353
VSE Script Language Built-In Console Functions 353
VSE Script Language Built-In POWER Functions 353
VSE Script Language Built-In VSAM Functions 354
Sample Files You Can Use for Writing VSE Script Clients 355
Example of Writing a VSE Script Client (and Its VSE Script) 356
Step 1: Setup the VSE Script Server Properties File. 356
Step 2: Setup the Connections Properties File 356
Step 3: Define the Sample VSAM Data 357
Step 4: Modify the Sample VSE Script 357
Step 5: Start the VSE Connector Server on the VSE/ESA Host 358
Step 6: Start the VSE Script Server Locally. . . 358
Step 7(a): Open the Sample Lotus 1-2-3 Spreadsheet File 359
Step 7(b): Open the Sample MS Office Spreadsheet 362
Step 7(c): Start a Sample VSE Script from the Command Line. 364

Appendix. AIBTDLI DL/I Messages and Return Codes 365

Glossary 367

Index 371

Figures

1. Overview of Connection Possibilities under VSE/ESA 4
2. Overview of 2-Tier Environments and the Programs You Can Use. 12
3. Overview of 3-Tier Environments and the Programs You Can Use. 13
4. Selecting the Components of the VSE Connector Client That You Require. 27
5. Online Help Options of the VSE Connector Client 28
6. Uninstalling the VSE Connector Client . . . 29
7. Job SKVCSSTJ (for Placing Startup Job in Reader Queue) 31
8. Job SKVCSCAT (for Cataloging Members for VSE Connector Server) 31
9. Member SKVCSCFG (for Specifying General Settings for VSE Connector Server). 32
10. Member SKVCSLIB (for Specifying Libraries to Be Accessed by VSE Connector Server) . . . 33
11. Member SKVCSPLG (for Specifying Plugins for VSE Connector Server) 33
12. Member SKVCSUSR (for Specifying Logon Access to VSE Connector Server) 34
13. Member SKVCSSSL (for Configuring the VSE Connector Server for SSL). 35
14. Startup Job STARTVCS (for Starting the VSE Connector Server) 36
15. Displaying the Commands Provided by the VSE Connector Server 38
16. Example of How the VSAM Redirector Connector Is Used 48
17. Flow of Control for VSAM PUT Request 52
18. Flow of Control for VSAM GET Request 53
19. Flow of Control for VSAM PUT Request 54
20. Flow of Control for VSAM GET Request 55
21. Job to Produce a Configuration Phase for the VSAM Redirector Connector 58
22. Data Layout Used With the HTMLHandler 66
23. Implementation of the HTMLHandler. . . . 66
24. Initialize Method of the HTMLHandler 66
25. Cleanup Method of the HTMLHandler 67
26. Open Method of the HTMLHandler 67
27. Close Method of the HTMLHandler 67
28. Request Method of the HTMLHandler . . . 68
29. Finished Method of the HTMLHandler 68
30. Console Listing from Running the HTMLHandler Request Handler 69
31. Output from Running the HTMLHandler Request Handler 69
32. Job SKSSLKEY to Catalog a Sample Keyring Set into the VSE Keyring Library 93
33. Job to Start Utility CIALSRVR on the VSE/ESA Host 96
34. "TCP/IP for VSE Key Generator" window 96
35. "Generate RSA Private Key" window 97
36. Job CIALCREQ to Request a Server Certificate 98
37. Example Output Listing From Job CIALCREQ 99
38. Thawte "Certificate Signing Request" Window 100
39. A Thawte Signed Server Certificate 100
40. Job CIALCERT to Catalog the Server Certificate. 101
41. Thawte Certificate Authority Web Site 102
42. Job CIALROOT to Catalog the Root Certificate. 103
43. Job CIALSIGV to Verify Certificates 103
44. Skeleton SKVCSSSL (Configure SSL for the VSE Connector Server) 106
45. Skeleton SKVCSCFG (Activate SSL Profile for the VSE Connector Server) 107
46. Set VSE Connector Client's SSL Parameters Using a Properties Object 109
47. Example of Java Properties File for the VSE Connector Client 110
48. Example of a Client Keyring File 112
49. Job to Configure the VSE Connector Server for Client Authentication. 116
50. "Generate Key" Window 118
51. "KM: Request a Certificate" Window – Choose Request Method. 118
52. "KM: Request a Certificate" Window – Enter Personal Details. 119
53. "KM: Import" Window – Import Certificate into Client Keyring File 120
54. "KM: KeyRing" Window – Display Client Certificate. 120
55. "KM: KeyRing" Window – Import Root Certificate. 121
56. Sample Job to Catalog Client Certificate into VSE Keyring Library 122
57. Listing All Client-Certificate/User-ID Pairs 126
58. Adding a Client-Certificate/User-ID Pair 127
59. Hierarchical Structure of VSAM Maps 130
60. Job To Define the Cluster for VSAM.RECORD.MAPPING.DEFS. 131
61. Syntax of RECMAP Command (IDCAMS) 132
62. Example of Creating a Local VSAM Map Object 135
63. Example of Creating Data Fields for a Map 136
64. Example of Displaying the Properties of a Map 136
65. Example of Creating a View for a Map 137
66. Example of Adding Data Fields to a View 137
67. Example of Displaying the Properties of a View 138
68. Example of How to Delete a Map. 138
69. Example of a VSAM MapTool Window 140
70. Code Containing CCF Statements (VSE/ESA 2.5) 145
71. Equivalent Code Containing CCI Statements (VSE/ESA 2.6) 146
72. Code Without SSL Support (VSE/ESA 2.5) 147

© Copyright IBM Corp. 2000, 2003

73. Equivalent Code Containing SSL Support (VSE/ESA 2.6) 148
74. Code Using VSE Java Beans to Access VSAM Data (VSE/ESA 2.5) 149
75. Equivalent Code Using JDBC to Access VSAM Data (VSE/ESA 2.6) 150
76. Example of a Javadoc Belonging to the VSE Java Beans Class Library 158
77. Example of a VSEResourceListener implementation 160
78. Program Flow for Using VSE Java Classes to Obtain a List of VSAM Catalogs 162
79. Creating a VSEConnectionSpec for Connecting to the VSE/ESA Host 162
80. Creating a VSESystem for Connecting to the VSE/ESA Host 163
81. Example of Accessing VSAM Data Using VSE Java Beans 164
82. Example of Accessing DL/I Data Using VSE Java Beans 167
83. Accessing VSE/POWER Data Using VSE Java Beans 172
84. Example of Submitting Jobs Using VSE Java Beans 174
85. Example of Accessing Librarian Using VSE Java Beans 175
86. Example of Accessing ICCF Data Using VSE Java Beans 177
87. Example of Accessing the Operator Console Using VSE Java Beans 180
88. Graphical User Interface, as Provided by the VSE Navigator 182
89. Configure Hosts for the VSE Navigator 184
90. Using the VSE Navigator to Access CICS Data 185
91. JDBC Example: Imports and Class Definition 191
92. JDBC Example: Perform the doGet() Method 192
93. JDBC Example: Display List of Available Flights 193
94. JDBC Example: Display Flight Properties and Entry Form 194
95. JDBC Example: Get and Display the Data Field Values 195
96. JDBC Example: Update the VSAM Cluster 196
97. JDBC Example: Get Order Values and Check Availability 197
98. JDBC Example: Create the New Order 198
99. JDBC Example: Check Processing and Increase Number of Reserved Seats 199
100. How Applets Are Used in the VSE/ESA 2-Tier Environment 202
101. How Applets Are Used in the VSE/ESA 3-Tier Environment 204
102. How the VSEApplet Server Is Used in the 3-Tier Environment 205
103. Window for VSAM Data-Mapping Applet 208
104. Data-Mapping Applet Code for Setting Up the Java Class 210
105. Sample Code for Initializing the Data-Mapping Applet 210
106. Data-Mapping Applet Code for Adding a Map to a VSAM Cluster 211
107. Sample Applet Code for Modifying a Map 212
108. Window for Changing a Map's Properties 212
109. Sample Applet Code for Modifying a Map's Data Fields 213
110. Window for Changing a Map's Data Fields 214
111. Using the Sample VSAM Applet to Access VSAM Data 217
112. Window Displayed by the Sample VSAM Applet 222
113. Using the Sample DL/I Applet to Access DL/I Data 231
114. Window Displayed by the Sample DL/I Applet 234
115. How Servlets Are Used in the VSE/ESA 3-Tier Environment 243
116. How Session Information Is Re-Used by the WebSphere Application Server 246
117. VSAM Structure of FLIGHT.ORDERING.FLIGHTS 247
118. VSAM Structure of FLIGHT.ORDERING.ORDERS 247
119. Example of a Servlet Using Forms to Obtain a User's Input 248
120. Example of Using Forms to Display Window Controls 249
121. Sample Servlet Code for Displaying a List of Flights 250
122. Sample Servlet Code for Getting Flight Instances from the Host 251
123. Flight Order Selection Window, As Generated by the Sample Servlet 252
124. Sample Servlet Code for Displaying Properties of a Flight 253
125. Flight Order Entry Window, As Generated by the Sample Servlet 255
126. Sample Servlet Code for Placing an Order 256
127. Sample Servlet Code for Creating a New Flight 259
128. Sample Servlet Code for Creating a New Order 260
129. Flight Order Confirmation Window, Generated by the Sample Servlet 261
130. How JSPs Are Used in the VSE/ESA 3-Tier Environment 264
131. Example of a Java Server Page /JSP) 265
132. How Containers Are Used To Manage EJBs 269
133. Overview of the Entities Involved in an EJB Method Call 270
134. How an EJB Client Communicates with an EJB 271
135. How EJBs Are Used Together with an Applet in the 3-Tier Environment 273
136. How the EJB Client Accesses EJBs in the Provided Example 284
137. Example of EJB Client Code 285
138. Overview of How a Server Plugin's Functions Are Called 288
139. How a Plugin's Functions Are Called During Startup. 289

140. How a Plugin's Functions Are Called When a Request Is Received 290
141. Overview of How a Plugin's Functions Are Called During Server Shutdown 291
142. Sample Code for Implementing PluginMainEntryPoint Function 292
143. Sample Code for Implementing the SetupPlugin Function 293
144. Sample Code for Implementing the CleanupPlugin Function 294
145. Sample Code for Implementing the GetHandledCommands Function 295
146. Sample Code for Implementing the SetupHandler Function 296
147. Sample Code for Implementing the ExecuteHandler Function 298
148. Sample Code for Distinguishing Between Multiple Requests Within a Plugin 298
149. Example of Calling a Stub Code Written in a Language Other Than C 299
150. How You Use DB2 Stored Procedures To Access VSAM Data 313
151. Typical Program Flow When Performing a VSAMSQL CLI Update 317
152. SQL Statements Supported by VSAMSQL Call Level Interface (CLI) 318
153. How You Use DB2 Stored Procedures To Access DL/I Data 320
154. DL/I Partition Layout for Batch, MPS Batch, CICS/DLI Online, and AIBTDLI Interface . . 322
155. How SOAP Is Used When the VSE/ESA Host Acts As SOAP Server 333
156. How SOAP Is Used When the VSE/ESA Host Acts As SOAP Client 334
157. Contents of the SOAP Parameter 336
158. Possible Values for Type of Value Field 336
159. Fields Contained in SOAP_PROG_PARAM Control Block 337
160. Fields Contained in SOAP_DEC_PARAM Control Block 338
161. Proxy Types That Can Be Used With SOAP_DEC_PARAM Control Block 338
162. Mapping COMMAREA to SOAP_PROG_PARAM Control Block . . . 339
163. Checking Which SOAP Method Has Been Requested. 339
164. Get Input Parameters from CICS Queue 340
165. Put Parameter Into the CICS Output Queue 340
166. Preparing the SOAP Client's Call Parameter 341
167. SOAP Client Prepares the SOAP_DEC_PARAM Structure. 341
168. SOAP Client Inserts Values into the SOAP Server's Input Queue 341
169. SOAP Client Calls SOAP Converter (IESSOAPE) to Handle Requests 342
170. SOAP Client Obtains Results of the SOAP Call. 342
171. SOAP Client Deletes CICS Queues 342
172. 343
173. Using CEDA to Define Sample SOAP Service to CICS 346
174. Using CEDA to Define SOAP Server to CICS 346
175. How the VSE Script Connector is Used 349
176. VSE Script Provided With the VSE Script Connector Example 358
177. Sample Lotus 1-2-3 Spreadsheet for VSE Script Connector Example 359
178. Transferring Data from VSAM Cluster to Lotus 1-2-3 Spreadsheet 359
179. Sample Script As Defined in Lotus 1-2-3 360
180. Visual Basic Script Used With Lotus 1-2-3 Spreadsheet Example 360
181. Sample Spreadsheet for MS Office Spreadsheet Example 362
182. Transferring Data from VSAM Cluster to MS Office Spreadsheet 362
183. Sample Script as Defined in MS Office 363

Tables

1. Connectivity Possibilities in 2-Tier Environments 17
2. Connectivity Possibilities in 3-Tier Environments 18
3. Currently Supported SSL Cipher Suites 112
4. Contents of the VSE Java Beans Class Library 155
5. SQL Statements Supported by JDBC 187
6. Relational Terms and Their VSE Equivalents 190
7. Properties of Session Beans and Entity Beans 267
8. CLI Functions You Can Use for Accessing Mapped VSAM Data 315
9. Files Supplied for Writing VSE Script Clients 355

© Copyright IBM Corp. 2000, 2003

Notices

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 that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

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, U.S.A.

Any pointers in this publication to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement. IBM accepts no responsibility for the content or use of non-IBM Web sites specifically mentioned in this publication or accessed through an IBM Web site that is mentioned in this publication.

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 Deutschland Informationssysteme GmbH
Department 0215
Pascalstr. 100
70569 Stuttgart
Germany
```

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

Trademarks and Service Marks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX	ES/9000	S/390
AS/400	IBM	SQL/DS
AT	Language Environment	Tivoli
C/370	MQSeries	VisualAge
CICS	Netfinity	VM/ESA
CICS/VSE	Nways	VSE/ESA
DB2	OS/2	VTAM
DB2 Connect	OS/390	WebSphere
DB2 Server for VSE	OS/400	xSeries
DB2 Universal Database	pSeries	z/OS
Distributed Relational Database Architecture	pSeries	zSeries

© Copyright IBM Corp. 2000, 2003

DRDA RS/6000

Java, Java Beans, and JavaScript are trademarks or registered trademarks of Sun Microsystems, Inc..

Microsoft, Windows, the Windows 95 logo, Windows NT and Windows XP, are trademarks or registered trademarks of Microsoft Corporation.

Visual Basic is a registered trademark of Microsoft Corporation.

Lotus, 1-2-3, Domino, and Notes are trademarks or registered trademarks of the IBM-Lotus Corporation.

SET and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

UNIX is a registered trademark of The Open Group in the United States and other countries.

LINUX is a registered trademark of Linus Torvalds and others.

Other company, product, and service names, may be trademarks or service marks of others.

About This Book

This manual describes how you can use the VSE/ESA e-business Connectors to develop e-business applications that access programs and data based on the VSE/ESA host.

Who Should Use This Book

This manual is intended for systems programmers who install additional VSE/ESA programs, and application programmers who are familiar with the Java object-oriented programming language.

How to Use This Book

This book is divided into four parts, as follows.

- **Part 1** provides you with an introduction to the VSE/ESA e-business Connectors, and the basic possible configurations of e-business solutions.
  - Chapter 1, "Introduction to e-business with VSE/ESA", on page 3 provides an introduction to the IBM Application Framework for e-business, and describes the Java-based connector and the DB2-based connector, which support the Application Framework for e-business. CICS and MQSeries connectivity possibilities under VSE/ESA are also described.
  - Chapter 2, "Overview of 2- and 3-Tier Environments", on page 11 describes the 2-tier and 3-tier environments that you can implement.
- **Part 2** describes the installation and customization activities you might carry out, in order to establish e-business connectivity within your VSE/ESA system.
  - Chapter 3, "Choosing the Connectivity You Require", on page 17 provides you with the information you require to decide which type of connectivity you require.
  - Chapter 4, "Installing the Common Prerequisite Programs", on page 21 provides instructions on how you install and configure the programs you require in order to use the VSE/ESA e-business Connectors.
  - Chapter 5, "Installing and Operating the Java-Based Connector", on page 25 describes how you install the VSE Connector Client on your middle-tier, and how you configure and start the VSE Connector Server on the VSE/ESA host.
  - Chapter 6, "Configuring DL/I for Access Via VSE Java Beans", on page 41 describes how you configure your DL/I system so that it can be accessed via VSE Java Beans.
  - Chapter 7, "Installing the VSE Script Connector", on page 43 describes how you install the *server-part* (the VSE Script Server) of the VSE Script Connector.
  - Chapter 8, "Installing the VSAM Redirector Connector", on page 47 describes how you install and implement the VSAM Redirector connector.
  - Chapter 9, "Customizing the DB2-Based Connector", on page 71 describes how you customize the DB2-based connector.
  - Chapter 10, "Configuring the VSAM-Via-CICS Service", on page 87 describes how you configure the VSAM-via-CICS service to access VSAM data without the restriction that you must use VSAM shareoption 4.

© Copyright IBM Corp. 2000, 2003

- Chapter 11, “Configuring Your VSE/ESA Host for SSL”, on page 91 describes how you configure your VSE/ESA host for Secure Sockets Layer (SSL) support.
- Chapter 12, “Configuring the Java-Based Connector for Server Authentication” , on page 105 describes how you configure the VSE Connector Server and VSE Connector Clients for SSL *server authentication* support.
- Chapter 13, “Configuring the Java-Based Connector for Client Authentication” , on page 115 describes how you configure the VSE Connector Server and VSE Connector Clients for SSL *client authentication* support
- Chapter 14, “Service Functions for Client Authentication”, on page 123 describes the utilities and dialog you can use to manage client certificates, when you implement client authentication.
- Chapter 15, “Mapping VSE/VSAM Data to a Relational Structure”, on page 129 describes how you can map VSAM data to a relational structure.

- **Part 3** provides theoretical and practical information describing how to use the Java-based connector and DB2-based connector to develop your e-business applications.
  - Chapter 16, “Migrating Your Programs”, on page 145 describes the actions you must take to migrate your programs so that they can take advantage of VSE/ESA support for the Common Client Interface (CCI), Secure Sockets Layer, and Java Database Connectivity (JDBC). It also describes how you can migrate applets to JDK 1.3, and new methods for the VSE Java Beans.
  - Chapter 17, “Using VSE Java Beans to Implement Java Programs”, on page 153 describes how you use the VSE Java Beans class library to write applets, servlets, Java Server Pages, and Enterprise Java Beans, that access VSE data.
  - Chapter 18, “Using JDBC to Access VSAM Data”, on page 187 describes how you can setup and issue relational database queries and update requests against VSAM data using a *Java Database Connectivity* (JDBC) driver.
  - Chapter 19, “Using Java Applets to Access Data”, on page 201 describes how you implement applets in 2- and 3-tier environments.
  - Chapter 20, “Using Java Servlets to Access Data”, on page 243 describes how you implement servlets in 3-tier environments.
  - Chapter 21, “Using Java Server Pages to Access Data”, on page 263 describes how you implement Java Server Pages (JSPs) in 3-tier environments.
  - Chapter 22, “Using EJBs to Represent Data”, on page 267 describes how you use Enterprise Java Beans (EJBs) in a 3-tier environment to represent DB2, DL/I, or VSAM data.
  - Chapter 23, “Extending the Java-Based Connector”, on page 287 describes how you can extend the Java-based connector in 2-tier and 3-tier environments, by writing your own “plugins”.
  - Chapter 24, “Using the DB2-Based Connector to Access Data”, on page 309 describes how you use the DB2-based connector to access VSAM and DL/I data, using the DB2 infrastructure.
  - Chapter 25, “Using SOAP for Inter-Program Communication”, on page 331 describes how you use the Simple Object Access Protocol (abbreviated to SOAP) to send and receive information between CICS programs and other modules, over the Internet.
  - Chapter 26, “Using the VSE Script Connector for Non-Java Access”, on page 349 describes how you use the VSE Script connector to access VSE/ESA host data from Java or non-Java platforms.

- "AIBTDLI DL/I Messages and Return Codes", on page 365 lists the new DL/I messages that can be generated by the AIBTDLI interface of the DB2-based connector

Where to Find More Information

Here is a list of IBM publications and publications from other vendors, that you might find useful. The IBM redbooks listed here are usually not kept up-to-date, but on the other hand *at the time they are written* they are at the forefront of the technical areas they describe.

- *VSE/ESA Planning*, SC33-6703
- *TCP/IP for VSE/ESA IBM Program Setup and Supplementary Information*, SC33-6601
- *CICS Transaction Server for VSE/ESA, Enhancements Guide*, GC34-5763
- *e-business Solutions for VSE/ESA (Redbook)*, SG24-5662
- *e-business Connectivity for VSE/ESA (Redbook)*, SG24-5950
- *Getting Started with TCP/IP for VSE/ESA 1.4 (Redbook)*, SG24-5626
- *TCP/IP Tutorial and Technical Overview (Redbook)*, GG24-3376
- *WebSphere Application Servers: Standard and Advanced Editions (Redbook)*, SG24-5460
- *WebSphere Application Server V4 for Linux, Implementation and Deployment Guide (Redpaper)*, REDP0405
- *IBM WebSphere V4.0 Advanced Edition Handbook (Redbook)*, SG24-6176
- *Linux on IBM zSeries and S/390: Server Consolidation with Linux for zSeries (Redpaper)*, REDP0222
- *Linux on IBM zSeries and S/390: High Availability for z/VM and Linux (Redpaper)*, REDP0220
- *Linux for S/390 (Redbook)*, SG24-4987
- *MQSeries for VSE/ESA (Redbook)*, SG24-5647

The online documentation provided by the VSE Connector Client includes a list of useful Internet sites, and online books. See "Using the Online Documentation Options" on page 28 for a description.

In addition, you might refer to the ...

VSE/ESA Home Page

VSE/ESA has a home page on the World Wide Web, which offers up-to-date information about VSE-related products and services, new VSE/ESA functions, and other items of interest to VSE users.

You can find the VSE/ESA home page at:

`http://www.ibm.com/servers/eserver/zseries/os/vse/`

Summary of Changes

Changes for Fifth Edition (March 2003)

VSE/ESA 2.7 includes:

- A change in the way the VSE Connector Client is downloaded and installed. For details, see "Obtaining a Copy of the VSE Connector Client" on page 25.
- A change in the way the `classpath` environment variable is set for the VSE Connector Client. Previously, you had to set the `classpath` environment variable manually. From VSE/ESA 2.7 onwards, it is set *automatically* in the installation batch files. For details, see "Performing the VSE Connector Client Installation" on page 26.
- The option to activate a new configuration of the VSAM Redirector Client while the VSAM system is running (that is, without restarting your VSAM applications). For details, see "Installing and Configuring the VSAM Redirector Client" on page 50.
- Optional parameters when specifying the names of clusters that are to be redirected by the VSAM Redirector connector. You can use these parameters for more exact filtering (for example, to include only clusters that are contained in a specific partition). For details, see "Optional Parameters" on page 56.
- A change in the way the VSAM Redirector Server is downloaded and installed. For details, see "Step 1: Download the Install-File and Perform the Installation" on page 60.
- VSE Connector Client support for the client-certificates dialog and various client-authentication service functions. These were first introduced with VSE/ESA 2.6.1 for use with CICS Web Support. For details, see Chapter 14, "Service Functions for Client Authentication", on page 123.
- A new parameter STA, which shows the status of the client-certificate/User-ID mapping list that is used with client authentication. For details, see "Using the Batch Service Function BSSDCERT" on page 123.
- Support for accessing DL/I data via VSE Java Beans. For details, see Chapter 6, "Configuring DL/I for Access Via VSE Java Beans", on page 41, and (for a code example of how to access DL/I data via VSE Java Beans) "Using VSE Java Beans for Accessing DL/I Data" on page 167.
- A change in the way the VSE Navigator is downloaded and installed. For details, see "Installing the VSE Navigator" on page 183.
- Support for SOAP (the Simple Object Access Protocol) which you can use for communicating between CICS Transaction Server for VSE/ESA programs, and programs running on other platforms. For details, see Chapter 25, "Using SOAP for Inter-Program Communication", on page 331.
- The VSE Script connector, which allows non-Java access to programs and data located on the VSE/ESA host. For details, see Chapter 7, "Installing the VSE Script Connector", on page 43 and Chapter 26, "Using the VSE Script Connector for Non-Java Access", on page 349.
- Support for Windows XP.
- Support for Java Development Kit (JDK) 1.4 and WebSphere 4.5.

The appendixes in this manual that described TCP/IP and SSL have been removed, since this information is now widely available on the internet, and can be obtained for all levels of user skill.

© Copyright IBM Corp. 2000, 2003

Changes for Fourth Edition (June 2002)

VSE/ESA 2.6.1 included:

- Minor changes to the procedure for implementing client authentication for VSE Connector Clients. This was mainly due to difficulties that could arise if a test certificate from the Thawte Corporation was used as a client certificate. For details, see "Configuring the VSE Connector Client for Client Authentication" on page 117.
- A new dialog and various service functions that you could use to help you implement client authentication, and manage client certificates. For details, see Chapter 14, "Service Functions for Client Authentication", on page 123.

Changes for Third Edition (December 2001)

VSE/ESA 2.6 included:

- The VSAM Redirector connector, which handles requests to VSAM datasets and then redirects them to a different Java platform (for example to Linux on zSeries, Windows NT, or Windows 2000) and to a different file system (for example to DB2 or flat files). Your source programs do not need to be changed. For details, see Chapter 8, "Installing the VSAM Redirector Connector", on page 47.
- Access to VSAM files using the VSAM-via-CICS service. This service can be used by your VSE Connector Client applications, or applications that call DB2 Stored Procedures. Using the VSAM-via-CICS service avoids the use of VSAM share option 4 for multiple writes when accessed from *one* CICS. For details, see Chapter 10, "Configuring the VSAM-Via-CICS Service", on page 87.
- SSL support for the Java-based connector (see "Configuring the VSE Connector Server for Server Authentication" on page 105 and "Configuring the VSE Connector Client for Server Authentication" on page 108 for details). This SSL support also includes *client authentication* (see Chapter 13, "Configuring the Java-Based Connector for Client Authentication", on page 115 for details).
- The VSAM MapTool, which allows you to create a map by parsing a COBOL copybook. For details, see "Defining a Map Using the VSAM MapTool" on page 140.
- VSE Java Beans that are based on the Java platform 2 (J2EE 1.3). You must however migrate your applets because of the changes in applet security. For details, see "Migrating Your Applets to JDK 1.3 or Later" on page 151.
- New methods for accessing "in-creation" entries, that have been added to VSE/POWER-related VSE Java. For details, see "Using the New Methods in VSE Java Beans" on page 152.
- JDBC support for the the Java-based connector. This includes a JDBC driver with a limited set of supported SQL constructs. For details, see Chapter 18, "Using JDBC to Access VSAM Data", on page 187.
- The Common Connector Interface (CCI) as a replacement for the IBM Common Connector Framework (CCF). In addition, the reuse of connections is now handled internally by the WebSphere Application Server (Version 4.0 and later). The former CCF package com.ibm.connector is no longer included in the *VSEConnector.jar* file. Instead, the CCI package (javax.resource) is shipped as a separate JAR file *cci.jar*, and must be imported by user applications. As a result of these changes, the chapter "Reusing Connections" that was present in VSE/ESA 2.5, no longer applies from VSE/ESA 2.6 onwards. For details about reusing connections, you should now refer to the online documentation provided with the VSE Connector Client.

Part 1. Introduction

Part 1 contains these chapters:

- Chapter 1, “Introduction to e-business with VSE/ESA”, on page 3
- Chapter 2, “Overview of 2- and 3-Tier Environments”, on page 11

© Copyright IBM Corp. 2000, 2003

Chapter 1. Introduction to e-business with VSE/ESA

To help you be successful in meeting your Internet and e-business requirements, IBM has developed an *Application Framework for e-business*. This framework has been developed with the aim of helping you to protect your investment in existing information assets, while enabling you to exploit the emerging e-business opportunities. The Application Framework for e-business consists of three logical layers:

1. A client, which is usually a workstation that has a standard Web browser installed. However, this can also be a wireless telephone or Personal Digital Assistant (PDA).
2. A Web application server, which is the "hub" that processes requests from clients, controls access to business logic and data. If the logic or data are stored on a different system, the Web application server uses connectors to access them. The Web application server also integrates static and dynamic content, and then returns Web pages to clients.
3. *Connectors*, which provide access to external services such as business logic and data. The connectors provided with VSE/ESA are now introduced.

The Application Framework for e-business covers the most up-to-date security standards such as SSL, SET, and firewall, and supports:

- WebSphere Application Server
- VisualAge for Java
- DB2 Universal Database
- Lotus Domino
- CICS Transaction Gateway
- MQSeries

The VSE/ESA e-business Connectors support the Application Framework for e-business, and provide you with the resources to extend your *core applications* to *e-business applications*. By doing so, you can protect and leverage existing core-application investments.

- *Core applications* (typically CICS, COBOL, VSAM) typically run on the VSE/ESA host, are critical to the company's operations, are expected to remain in production for many years to come, and usually represent an enormous investment of past resources.
- *e-business applications* are typically based upon common standards such as TCP/IP, HTML, XML, Secure Sockets Layer (SSL), Secure Electronic Transaction (SET) and so on, include both the server and client code written in Java, access relational data locally or remotely, and interface with end-users via a standard Web browser.

You can, for example, extend your *core* applications to the Web and combine S/390 with non-S/390 servers (RS/6000, Netfinity, and AS/400), to produce state-of-the-art e-business solutions.

© Copyright IBM Corp. 2000, 2003

What the VSE/ESA e-business Connectors Provide

These are the current VSE/ESA e-business Connectors:

- The Java-based connector, which consists of a client-part (the *VSE Connector Client*) and a server-part (the *VSE Connector Server*).
- The DB2-based connector, including the new DB2 Stored Procedure support.
- The VSAM Redirector connector, which consists of a client-part (the *VSAM Redirector Client*) and a server-part (the *VSAM Redirector Server*).
- The VSE Script connector, which consists of a client-part (any user-written Java or non-Java application called the VSE Script Client) and a server-part (the IBM-supplied VSE Script Server, which is a Java application).

Figure 1 shows the connections you can make between Web clients /the middle-tier and the VSE/ESA host.

Notes:

1. The components of the Java-based connector are shown as shaded.
2. The DB2-based connector uses a connection between DB2 Connect on the middle-tier and DB2 Server for VSE on the VSE/ESA host.

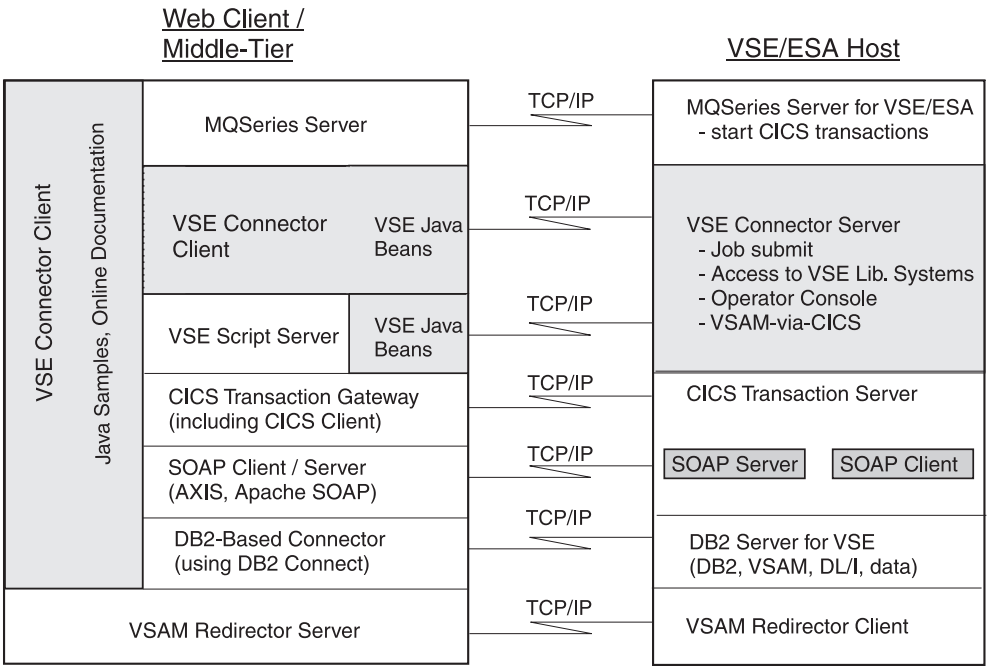


Figure 1. Overview of Connection Possibilities under VSE/ESA

Overview of the Java-Based Connector

The *Java-based connector* consists of a client-part (the *VSE Connector Client*), and a server-part (the *VSE Connector Server*).

Overview of the VSE Connector Client

To install the VSE Connector Client, you install these three components on any Java-enabled middle-tier platform:

- a file **VSEConnector.jar** which contains the VSE Java Beans class library. The VSE Java Beans provide a Java programming interface for communicating with VSE/VSAM, VSE/Librarian, VSE/POWER, VSE/ICCF, and Operator Console, on the VSE/ESA host.
- a set of samples, including Java source code, that show you how to write Java programs that are based upon the use of VSE Java Beans.
- online documentation (a set of HTML pages) describing the various concepts and samples.

To develop your Web applications, you will probably use all the three components listed above. However, the completed Web applications require only the VSE Java Beans class library at run-time. Therefore, when your Web applications are ready for the production environment (the end-user environment), you install only the **VSEConnector.jar** file (which contains the VSE Java Beans class library) on the:
- Middle-tier (for 3-tier environments)
- Web clients (for 2-tier environments).

The difference between 2-tier and 3-tier environment is explained in Chapter 2, “Overview of 2- and 3-Tier Environments”, on page 11.

For a detailed description of how to install the VSE Connector Client, see “Installing the VSE Connector Client” on page 25.

Overview of the VSE Connector Server

The VSE Connector Server is installed on the VSE/ESA host. It is a batch application that runs by default in dynamic class R. After you have configured the VSE Connector Server, it must simply be started in order to become operational. It then provides a TCP/IP *socket listener* which can handle multiple clients.

Java programs running on Web clients or the middle-tier use the VSE Java Beans to build connections to the VSE Connector Server running on the VSE/ESA host.

To start the VSE Connector Server, you use the job **STARTVCS**, which is placed in the POWER reader queue during the installation of the VSE/ESA base. When started, the VSE Connector Server listens to incoming TCP/IP traffic on port 2893 by default. For details of how to start the VSE Connector Server, see “Starting the VSE Connector Server” on page 36.

The VSE Connector Server is pre-configured for your use, and should require no major configuration effort by yourself. However, you can modify various configuration members to specify:
- The VSE/AF libraries that can be accessed by the VSE Connector Server. You may extend or restrict this list according to your needs.
- Plugins to be loaded at VSE Connector Server startup. You can extend the Java-based connector by specifying your own host-side plugins, as described in Chapter 23, “Extending the Java-Based Connector”, on page 287.
- Which users or groups of users are allowed to logon to the VSE Connector Server.

For details of how to modify the configuration members belonging to the VSE Connector Server, see “Configuring the VSE Connector Server” on page 30.

For an overview of where the VSE Connector Client and VSE Connector Server are used in 2-tier and 3-tier environments, see Figure 2 on page 12 and Figure 3 on page 13.

Overview of the DB2-Based Connector

The *DB2-based connector* is an optional feature that allows you to use the *Distributed Relational Database Architecture* (DRDA) to access non-relational data such as VSE/VSAM and DL/I data. Your application programs use standard interfaces such as JDBC, ODBC, or Call Level Interface (CLI) to request data.

The implementation of the DB2-based connector is based upon the use of *DB2 Stored Procedures*, which you can use with the DB2 Server for VSE & VM, Version 6 or later. DB2 Stored Procedures are application programs that you write, and then compile and store on your VSE/ESA host.

You can write DB2 Stored Procedures in any LE (Language Environment)-compliant language (COBOL, C, or PL/I). Local or remote DRDA applications can then invoke these DB2 Stored Procedures.

The DB2-based connector enables you to access VSE/VSAM and DL/I data from within the *same* DB2 Stored Procedure that you use to access DB2 data:

- To access VSE/VSAM data, you use the *VSAM Call Level Interface*. See "Using DB2 Stored Procedures to Access VSAM Data" on page 312 for details.
- To access DL/I data, you use the *AIBTDLI interface*. See "Using DB2 Stored Procedures to Access DL/I Data" on page 320 for details.

For an overview of where the DB2-based connector is used in a 3-tier environment, see Figure 3 on page 13.

Overview of the VSAM Redirector Connector

The VSAM Redirector Connector enables VSE programs to access data on remote systems, in real-time. Using the VSAM Redirector Connector:

- VSAM data can be migrated to other file systems or databases.
- Data can be synchronized on different systems with VSE VSAM data.
- VSE programs can work transparently with data on other file systems or databases.

A Java handler provides access to the specific file system or database on the remote system. For example, you can migrate your VSAM data into DB2 tables residing on a remote system, and your VSE programs will then work with this data, without requiring any changes to these VSE programs.

The VSAM Redirector connector handles requests to VSAM datasets and redirects them to a different:

- Java platform (for example Linux on zSeries, Windows NT, Windows 2000, Windows XP).
- file system (for example DB2 or flat files).

Your existing VSE/ESA host programs that are:

- written in any language (COBOL, PL/I, ASSEMBLER)
- batch or CICS programs

can therefore work with migrated VSAM data without the need to amend and recompile these VSE/ESA host programs. The VSAM Redirector connector manages all connections and data conversions.

The VSAM Redirector connector consists of:

- The *VSAM Redirector Client* (installed on your VSE/ESA host).

- A *VSAM Redirector Server* installed on *each* Java platform.

For further details about the VSAM Redirector connector, see Chapter 8, "Installing the VSAM Redirector Connector", on page 47.

Overview of the VSE Script Connector

As described in "Overview of the Java-Based Connector" on page 4, VSE Java Beans provide direct access to the VSE/ESA host from any kind of Java program (servlets, applets, EJBs, and so on) running on a *Java platform*. In addition, you can use the *VSE Script connector* to access VSE/ESA host data from *non-Java* platforms. This is the main advantage of using the VSE Script connector (although it can also be used to access VSE/ESA host data from Java platforms).

The VSE Script connector is supplied as *part of* the Java-based connector. It can only be used in a 3-tier environment (explained in "Overview of 3-Tier Environments" on page 13), and consists of:

- a VSE Script Client running on a Java or non-Java platform, and which can be either:
  - a user-written Java application (for example a Web-service).
  - a user-written non-Java application (for example a Windows C-program, a Windows CGI-program, or a COBOL application).
  - an office product, such as a word-processing or spreadsheet program (for example Lotus 1-2-3 or Lotus WordPro), where for example, a Visual Basic script is used to call a VSE Script.
- the VSE Script Server running on the middle-tier of a 3-tier environment, which interprets and executes VSE Script files.
- online documentation, including a programming reference manual.

The VSE Script connector works in this general way:

1. The VSE Script Client calls a VSE Script, to make a request for data stored on the VSE/ESA host. These VSE Script (batch) files contain statements written using the VSE Script language, which is a special programming language. The VSE Script language can be used in any environment (even in Visual Basic scripts).
2. The VSE Script Server running on a Java-enabled middle-tier platform then reads, interprets, and translates, the VSE Script file statements into VSE Java Beans requests. The VSE Script Server uses the VSE Java Beans to connect to the VSE Connector Server running on the VSE/ESA host, and to forward the VSE Java Beans requests.
3. The VSE Connector Server accesses the required VSE/ESA data and functions, and sends the reply back to the VSE Script Server.
4. The VSE Script Server converts the data to the format that the VSE Script Client can use, and returns the data to the VSE Script Client.

For details about the VSE Script connector, see Chapter 7, "Installing the VSE Script Connector", on page 43, and Chapter 26, "Using the VSE Script Connector for Non-Java Access", on page 349.

Overview of VSE/ESA Support for Web Services and SOAP

SOAP is a standard, XML-based, industry-wide protocol that allows applications to exchange information over the Internet via HTTP.

XML is a universal format that is used for structured documents and data on the Web. It is independent of both the Web client's operating-system platform and the programming language used. HTTP is supported by all Internet Web browsers and servers.

SOAP combines the benefits of both XML and HTTP into one standard application protocol. As a result, you can send and receive information to/from various platforms.

Using Web browsers, you can view information contained on Web sites. However, using SOAP you can:

- combine the contents of *different* Web sites and services.
- generate a complete view of all the relevant information.

VSE/ESA supports the SOAP protocol and therefore allows you to implement Web services.

An example of using SOAP might be when a travel agent requires a combined view of the Web services covering hotel reservation, flight booking, and car rental. After the travel agent has entered the required data, all three Web services from the three different providers would be processed in one transparent step. This is an example of how a "Business-to-Business" (B2B) relationship can be implemented.

For details of how to implement SOAP in your VSE/ESA system, see Chapter 25, "Using SOAP for Inter-Program Communication", on page 331.

Overview of CICS Connectivity

CICS connectivity in a *2-tier environment* enables CICS applications to be accessed using the CICS Web Support and 3270 Bridge (functions delivered with CICS Transaction Server for VSE/ESA).

CICS connectivity in a *3-tier environment* enables:

- A Java gateway application, that is usually stored on the middle-tier, to communicate with CICS applications running in the CICS TS through the ECI (External Call Interface) or EPI (External Presentation Interface) provided by the CICS Universal Client.
  - The *ECI Interface* enables a non-CICS Client application to call a CICS program synchronously or asynchronously as a subroutine.
  - The *EPI Interface* enables a non-CICS Client application to act as a logical 3270 terminal and so control a CICS 3270 application.

  The CICS Universal Client communicates with the CICS TS via the APPC protocol.
- A CICS Java class library to be used for communication between the Java gateway application and a Java application (applet or servlet). The CICS Java class library also includes classes that provide an application programming interface (API):
  - Java programs can use the *JavaGateway* class to establish communication with the Gateway process, and this class uses Java's sockets protocol.
  - Java programs can use the:
    - *ECIRequest* class to specify the **ECI** calls that are flowed to the gateway.
    - *EPIRequest* class to specify **EPI** calls that are flowed to the gateway.
- A Web browser to be used as an emulator for a 3270 CICS application running on the CICS Transaction Server for VSE/ESA, via a Terminal Servlet.
- A set of Java EPI Beans to be used for creating Java front-ends for existing CICS 3270 applications, without any programming effort.
- The Simple Object Access Protocol (abbreviated to SOAP) to be used to send and receive information between CICS programs and other modules, over the Internet. For further information about SOAP, see Chapter 25, "Using SOAP for Inter-Program Communication", on page 331.

Overview of MQSeries Connectivity

MQSeries connectivity in a 3-tier environment enables:

- Java applets to access MQSeries queues on the VSE/ESA host. By exploiting the trigger facility provided by MQSeries, you can start CICS applications that access CICS data and resources on the VSE/ESA host.
- a Web Client to *participate* in transactions, instead of simply providing and receiving information.

For further information about MQSeries connectivity in a 3-tier environment, see Table 2 on page 18.

Overview of the IBM WebSphere Application Server

You implement the WebSphere Application Server on the middle-tier of the *3-tier* Application Framework for e-business environment shown in Figure 3 on page 13. It is used together with a Web server (such as the IBM HTTP Server, or Apache server).

e-business with VSE/ESA

The WebSphere Application Server is fully compatible with industry standards such as Enterprise Java Beans (EJBs), eXtensible Markup Language (XML), and Common Object Request Broker Architecture (CORBA), and provides you with a solid framework for implementing your e-business applications.

Complementary programs include the:

1. WebSphere Studio, which is a powerful set of application development tools and facilities.
2. WebSphere Performance Pack.

In addition, you can use Tivoli's TME-10 network management together with the WebSphere Application Server.

Using the WebSphere Application Server, you can write applications (using Web development tools) for the middle-tier that can access data and programs stored on the VSE/ESA host (CICS, DB2, and so on). These applications can be written to take advantage of the full benefits of Java and Internet technologies.

For an overview of where the WebSphere Application Server is used in 3-tier environments, see Figure 3 on page 13.

For more information about the WebSphere Application Server range, you might also refer to this Internet address:

www-4.ibm.com/software/webservers/appserv

Chapter 2. Overview of 2- and 3-Tier Environments

You can choose between 2-tier and/or 3-tier environments for the communication between Web clients, and the programs and data stored on the VSE/ESA host. These environments are illustrated in Figure 2 on page 12 and Figure 3 on page 13 respectively:

- In 2-tier environments, the Web client and VSE/ESA host communicate directly with each other.
- In 3-tier environments, the Web client or non-Java client, and VSE/ESA host communicate with each other via an intermediate tier called the *middle-tier*.

Notes:

1. The 2-tier environment is *not* the typical environment under which you will develop your Java programs, since it is not:
   - part of the IBM *Application Framework for e-business* (it does not, for example, use the IBM WebSphere Application Server on the middle-tier).
   - secured by the state-of-the-art security services (firewall, and so on) provided by the IBM Application Framework for e-business.

   The 2-tier environment is generally suitable for *intranet* solutions only.
2. The Java-based connector is normally used in a 3-tier environment, but the DB2-based connector can *only* be used in a 3-tier environment. The 3-tier environment requires a middle-tier server (such as Netfinity, or RS/6000) on which the WebSphere Application Server is installed.

This chapter contains these main sections:

- "Overview of 2-Tier Environments" on page 12
- "Overview of 3-Tier Environments" on page 13

© Copyright IBM Corp. 2000, 2003

Overview of 2-Tier Environments

In 2-tier environments as shown in Figure 2:

1. The VSE Java Beans class library, which is part of the VSE Connector Client, must be accessible from each Java program running on a Web client, that communicates with the VSE Connector Server. This is achieved by copying the file **VSEConnector.jar** (which contains the VSE Java Beans) to each Web client on which your Java programs are to run.
2. The VSE Java Beans are used for establishing connections between the Java program running on the Web client and the VSE Connector Server running on the VSE/ESA host. Java applications or applets running on the Web clients can then use standard Web browsers to communicate directly with the VSE HTTP Server and VSE Connector Server running on the VSE/ESA host. When the work is complete, replies are sent from the VSE Connector Server to the Web client.
3. You can redirect VSAM requests to any Java-enabled platform using the VSAM Redirector connector.

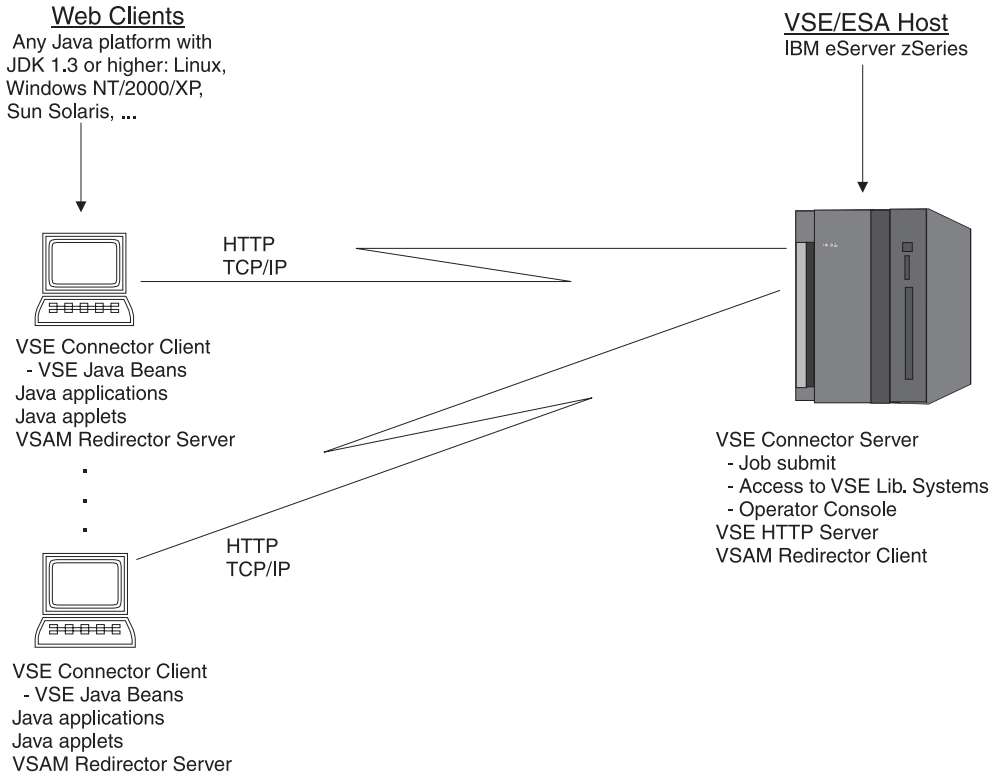


Figure 2. Overview of 2-Tier Environments and the Programs You Can Use

An example of a 2-tier implementation might be a Java application that directly communicates with the VSE Connector Server, to access POWER data.

For a detailed description of how applets can be used in 2-tier environments, see Figure 100 on page 202

Overview of 3-Tier Environments

In your 3-tier environments as shown in Figure 3, a WebSphere Application Server running for example, on an IBM eServer zSeries, Netfinity, or RS/6000, is the central "hub":

1. Either:
   - Web clients use standard Web browsers to communicate with an application that is based upon the WebSphere Application Server on the middle-tier.
   - Non-Java clients (for example a spreadsheet application running under Windows) use a VSE Script to communicate with the VSE Script Server running on the middle-tier.
2. WebSphere applications, such as servlets, EJBs, or the VSE Script Server, use the VSE Java Beans to access VSE data or start an application on the VSE/ESA host.
3. Replies are sent from the VSE Connector Server to the application running on the WebSphere Application Server. When the activities are completed, the application packages the data with other information, and sends a reply back to either the Web client or to the non-Java client.

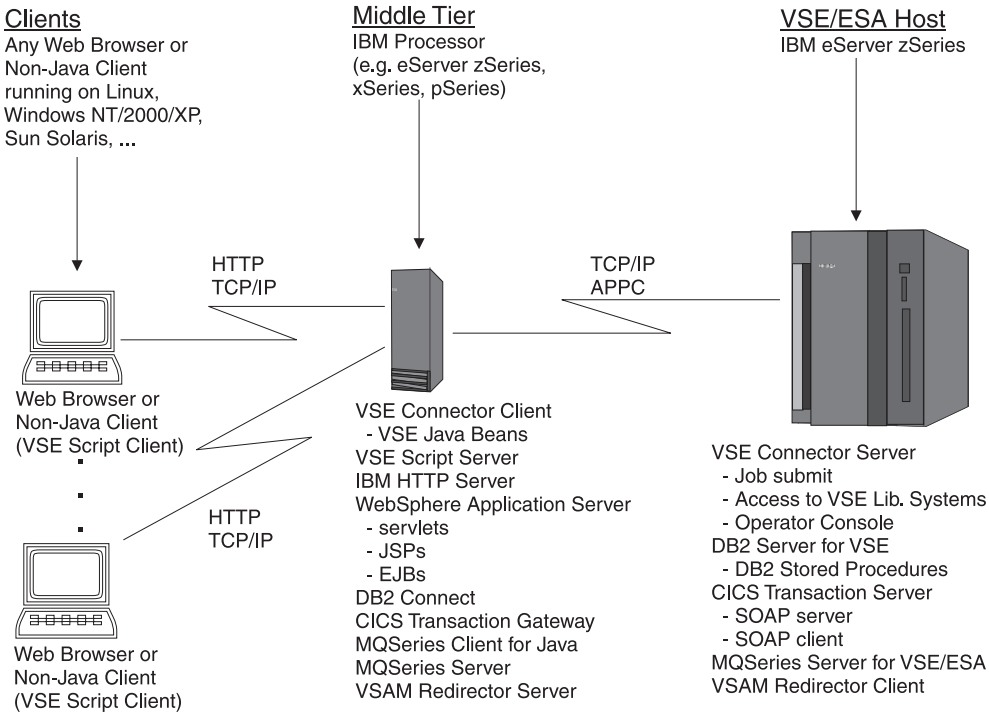


Figure 3. Overview of 3-Tier Environments and the Programs You Can Use

Note: In this manual, a *firewall* used in a 3-tier environment is *not* considered as a separate tier.

An example of a 3-tier implementation might be a Java servlet running on the Web Application Server (WebSphere) that provides access to VSAM data. The servlet might allow regular customers to enter orders over the Internet, using the required security layers.

Another (**non-Java**) example of a 3-tier implementation might be a Lotus 1-2-3 application running under Windows (the VSE Script Client) that obtains VSAM

data stored on the VSE/ESA host, via the VSE Script Server running on the middle-tier. VSE Script connector is the term used to refer to both the VSE Script Client and the VSE Script Server.

For a detailed description of how:

- Applets can be used in 3-tier environments, see Figure 101 on page 204.
- Servlets can be used in 3-tier environments, see Figure 115 on page 243.
- Java Server Pages (JSPs) can be used in 3-tier environments, see Figure 130 on page 264.
- Enterprise Java Beans (EJBs) can be used in 3-tier environments, see Figure 135 on page 273.
- DB2 Stored Procedures can be used to access VSE/VSAM data, see Figure 150 on page 313
- DB2 Stored Procedures can be used to access DL/I data, see Figure 153 on page 320.
- The VSAM Redirector Connector can be used to redirect requests for VSAM data, see Figure 16 on page 48.
- The Simple Object Access Protocol (abbreviated to SOAP) can be used to send and receive information between CICS programs and other modules over the Internet, see Chapter 25, “Using SOAP for Inter-Program Communication”, on page 331.
- The VSE Script connector can be used for non-Java access to functions and data stored on the VSE/ESA host, see Chapter 26, “Using the VSE Script Connector for Non-Java Access”, on page 349.

Part 2. Installation & Customization

Part 2 contains these chapters:

- Chapter 3, “Choosing the Connectivity You Require”, on page 17
- Chapter 4, “Installing the Common Prerequisite Programs”, on page 21
- Chapter 5, “Installing and Operating the Java-Based Connector”, on page 25
- Chapter 6, “Configuring DL/I for Access Via VSE Java Beans”, on page 41
- Chapter 7, “Installing the VSE Script Connector”, on page 43
- Chapter 8, “Installing the VSAM Redirector Connector”, on page 47
- Chapter 9, “Customizing the DB2-Based Connector”, on page 71
- Chapter 10, “Configuring the VSAM-Via-CICS Service”, on page 87
- Chapter 11, “Configuring Your VSE/ESA Host for SSL”, on page 91
- Chapter 12, “Configuring the Java-Based Connector for Server Authentication”, on page 105
- Chapter 13, “Configuring the Java-Based Connector for Client Authentication”, on page 115
- Chapter 14, “Service Functions for Client Authentication”, on page 123
- Chapter 15, “Mapping VSE/VSAM Data to a Relational Structure”, on page 129

© Copyright IBM Corp. 2000, 2003

Chapter 3. Choosing the Connectivity You Require

For 2-tier and 3-tier environments, this chapter describes the types of connectivity you can establish using the Java-based connector, the DB2-based connector, the VSAM Redirector connector, the VSE Script connector, CICS (including the SOAP server and SOAP client), and MQSeries.

This chapter contains these main sections:

- "Connectivity Possibilities in 2-Tier Environments"
- "Connectivity Possibilities in 3-Tier Environments" on page 18

Connectivity Possibilities in 2-Tier Environments

For 2-tier environments, you can use:

- The Java-based connector
- The CICS *Web Support* feature
- The VSAM Redirector connector

to carry out the functions described in Table 1.

Table 1. Connectivity Possibilities in 2-Tier Environments

Connectivity	Java Applications	Servlets / Java Server Pages	Applets	Enterprise Java Beans
Java-Based Connector	Implement a Java application on the client, that uses the VSE Beans class library to access all VSE file systems (see **Note** below), submit jobs, issue console commands, etc.	Not possible	Download an applet from VSE to your Web browser. Then use the VSE Beans class library to access all VSE file systems (see **Note** below), submit jobs, issue console commands, and so on. The VSE Beans classes can be put into the same JAR file together with the applet code.	Not possible
CICS	Use the *Web Support* feature provided by the CICS Transaction Server for VSE/ESA, to access CICS transactions directly from a Web browser. The CICS transactions generate the required Web pages, which are then displayed by the Web browser. However, since the use of the Web Support feature in a VSE/ESA environment is generally outside the scope of this manual, for details you should refer to the *CICS Transaction Server for VSE/ESA, CICS Internet Guide*, SC34-5765.			
VSAM Redirector connector	The VSAM Redirector connector handles requests to VSAM datasets and then redirects them to a different Java platform (for example to Linux on zSeries, Windows NT, Windows 2000, or Windows XP) and to a different file system (for example to DB2 or flat files). Your source programs do not need to be changed. For details, see Chapter 8, "Installing the VSAM Redirector Connector", on page 47.			

Note: The term *VSE file systems* includes VSE/VSAM, VSE/POWER, VSE/Librarian, and VSE/ICCF.

© Copyright IBM Corp. 2000, 2003

Connectivity Possibilities in 3-Tier Environments

For 3-tier environments, you can use:

- The Java-based connector
- The DB2-based connector
- The VSE Script connector
- CICS connectivity, including the Simple Object Access Protocol (abbreviated to SOAP)
- MQSeries connectivity

to carry out the functions described in Table 2.

Table 2. Connectivity Possibilities in 3-Tier Environments

Connectivity	Java Applications	Servlets / Java Server Pages	Applets	Enterprise Java Beans
Java-based connector	Not applicable.	Implement a servlet using the VSE Beans class library to access the VSE/ESA host data and display it through HTML pages. Refer to the online documentation (described on page 28) for details.	Download an applet from the middle tier's Web server and use the *VSEAppletServer* (see page 205 for a description) to connect to a remote VSE/ESA host to access VSE file systems (see **Note** at end of this table). Refer to the online documentation (described on page 28) for details.	Write an EJB that uses the VSE Java Beans class library to communicate with the VSE/ESA host and access your database on the VSE/ESA host. Normally, this will be either DB2 data or VSE/VSAM data. Refer to the online documentation (described on page 28) for details.
DB2-based connector	Implement a Java application on the Web client, that uses DB2 Connect on the middle-tier to access VSAM and DL/I data by calling a DB2 Stored Procedure on the VSE/ESA host.	Write a servlet or JSP that uses DB2 Connect to access VSAM and DL/I data by calling a DB2 Stored Procedure on the VSE/ESA host. The servlet or JSP can be accessed from any Web client through WebSphere.	Download an applet from the middle-tier's web server. The applet connects to DB2 Connect on the middle-tier, which accesses VSAM and DL/I data by calling a DB2 Stored Procedure on the VSE/ESA host.	Write an EJB that uses DB2 Connect to access VSAM and DL/I data by calling a DB2 Stored Procedure on the VSE/ESA host.
VSE Script connector	Implement a non-Java program to call a VSE Script to access VSE functions and data. For example, write a Visual Basic™ script within an Office product such as Lotus 1-2-3, to include VSE data in a spreadsheet or document.	Not applicable.	Not applicable.	Not applicable.

Table 2. Connectivity Possibilities in 3-Tier Environments (continued)

Connectivity	Java Applications	Servlets / Java Server Pages	Applets	Enterprise Java Beans
CICS	Implement a Java application on the Web client, that connects to the CICS Transaction Gateway on the middle-tier. Use the ECI/EPI interface to communicate with the CICS TS.	Write a servlet or JSP that uses the CICS ECI/EPI interface to communicate with the CICS TS. The servlet or JSP can be accessed from any Web client through WebSphere.	Download an applet from the middle-tier's Web server. The applet connects to the CICS Transaction Gateway on the middle-tier, which then communicates with the CICS TS.	Write an EJB that uses the CICS client's ECI/EPI interface to communicate with the CICS TS.
MQSeries	Implement a Java application on the Web client, that uses the MQSeries Client for Java to connect to (for example) the MQSeries Server for Windows 2000 on the middle-tier. The MQSeries Server for Windows 2000 (for example) on the middle-tier in turn connects to the MQSeries Server for VSE/ESA on the VSE/ESA host. You can then start any CICS transaction on the VSE/ESA host from a Java application on the Web client.	Write a servlet or Java Server Page (JSP) that uses the MQSeries Client for Java to start any CICS transaction on the VSE/ESA host. The servlet or JSP can be accessed from any Web client through WebSphere.	Download an applet from the middle-tier's Web server and use a ″router″ to connect to a remote VSE/ESA host to access VSE based data.	Write an Enterprise Java Bean (EJB) that uses the MQSeries Client for Java to communicate with the VSE/ESA host and access a database there.

Note: The term *VSE file systems* includes VSE/VSAM, VSE/POWER, VSE/Librarian, and VSE/ICCF.

Connectivity Possibilities

Chapter 4. Installing the Common Prerequisite Programs

This chapter describes the activities you must perform and which are independent of your choice of connectors. It describes how you:

- Activate and configure TCP/IP on your VSE/ESA host (see "Configuring and Activating TCP/IP for VSE/ESA").
- Install the VSE HTTP Server on your VSE/ESA host (see "Configuring and Activating the VSE HTTP Server").
- Install Java on your middle-tier (see "Installing and Configuring Java").
- Install the IBM HTTP Server on your middle-tier (see "Installing the IBM HTTP Server" on page 22).
- Install the WebSphere Application Server on your middle-tier (see "Installing the WebSphere Application Server" on page 23).

Configuring and Activating TCP/IP for VSE/ESA

You require TCP/IP for VSE/ESA in order to use the:

- Java-based connector
- DB2-based connector
- VSAM Redirector connector
- VSE Script connector

TCP/IP for VSE/ESA is supplied with VSE/ESA, but requires a key to be activated, which you must purchase. For details on how to activate and configure TCP/IP for VSE/ESA, refer to the *TCP/IP for VSE/ESA IBM Program Setup and Supplementary Information*. This manual is also available online at **http://www.ibm.com/servers/s390/os390/bkserv/vse.html .**

Note: Where you can use VTAM / APPC instead of TCP/IP, is noted in this manual.

Configuring and Activating the VSE HTTP Server

The TCP/IP for VSE/ESA *Application Pak* includes the VSE HTTP Server. You should therefore refer to the *TCP/IP for VSE/ESA Installation Guide*, SC33-6741, for details of how to configure and activate the VSE HTTP Server. This manual is available as a *PDF file only*. You can obtain this manual from:

- Disk 3 (PDFs) of the *VSE Collection* Online Library, SK2T-0060.
- URL **http://www.ibm.com/servers/s390/os390/bkserv/vse/vsepdf/**

Installing and Configuring Java

Java is a programming language in which *bytecode* is created from Java source files. This bytecode is stored in Java *class files.* These class files are read and executed by the Java interpreter (for Windows and OS/2, this is the program **java.exe**).

© Copyright IBM Corp. 2000, 2003

Common Prerequisite Programs

As stated previously, the VSE/ESA Java-based connector consists of the VSE Connector Client and the VSE Connector Server. To develop Web applications that use the *VSE Connector Client*, you must install the Java Development Kit 1.3 or later on your development platform.

Notes:

1. You can install different versions of Java on the same middle-tier platform, but to do so you must ensure that the correct paths are set for the Java version that is to run.
2. To use some of the samples supplied with the Java-based connector, you require the Java *Swing* classes. These are supplied when you install the VSE Connector Client (described in "Installing the VSE Connector Client" on page 25).

Downloading the Java Base Code

You can download Java-related code from these IBM web sites:

- **http://www.ibm.com/java/jdk/** from which you can download, for example, the Java Development Kits (JDKs) for AIX, OS/2, z/OS (Unix services), OS/400, VM/ESA, Linux, and Windows. You can download the code (without payment) either from this site or from other related sites.
- **http://www.ibm.com/developer/java/** from which you are given access to various libraries containing both tools and documentation.

Deciding Which Java Package to Install

You can install Java in one of two ways on your middle-tier server:

- As a Java Development Kit (JDK) installation, in which you use **java.exe** to run your Java programs. The JDK contains the run-time environment, together with tools and facilities such as debugger, compiler, and so on. You will require the JDK if you are developing your own Java programs.
- As a Java Runtime Environment (JRE) installation, in which the JRE contains only the runtime library required to run Java applications (development tools are not included). Here, you use **jre.exe** to run your Java programs.

Installing the IBM HTTP Server

On your middle-tier, you must install a *Web Server* which can be, for example, the:

- IBM HTTP Server
- Lotus Domino Go Webserver
- Apache Server

The IBM HTTP Server is part of the WebSphere Application Server package. During the installation of the WebSphere Application Server, you are asked if you wish to:

- install the IBM HTTP Server as your Web Server.
- use another Web Server (for example, the Apache Server).

For details of how to install the IBM HTTP Server on your middle-tier, refer to the installation instructions provided with the WebSphere Application Server.

Installing the WebSphere Application Server

On your middle-tier you must install an *Application Server*, which can be the IBM WebSphere Application Server, or any other vendor's application server.

The WebSphere Application Server is supplied in these editions:

- The Standard Edition, which supports servlets and Java Server Pages (JSPs).
- The Advanced Edition or the Enterprise Edition, which both support servlets, JSPs, and Enterprise Java Beans (EJBs).

Because the installation of the WebSphere Application Server on various platforms is both complex and subject to change, this section only provides documentation references and/or a description of the general installation steps.

Installing the WebSphere Application Server on z/OS

Refer to the relevant IBM publications for information on how to install the WebSphere Application Server and Supporting Products on the z/OS platform.

Installing the WebSphere Application Server on Other Platforms

These are the *general* steps you might follow to install the WebSphere Application Server on platforms such as Windows (NT/2000/XP ...), Linux, AIX or Sun Solaris:

1. Make sure you meet the hardware and software prerequisites for the platform and configuration you plan to install.
2. Decide on which of the installation steps you need to follow. For example, if your system does not already have the WebSphere Application Server, IBM Java Development Kit, IBM HTTP Server, and DB2 Universal Database installed, you will follow a different procedure than if your system already has these products installed.
3. Install the IBM Java Development Kit by:
   a. running an **exec** file
   b. following the instructions displayed on each window.
4. Install the IBM HTTP Server (or another Web Server, such as Apache).
5. Install the IBM DB2 Universal Database (or another database system, such as Oracle).
6. Test the installation of the prerequisite products for your configuration.
7. Install the WebSphere Application Server on the platform you have chosen.
8. Test the installation of the WebSphere Application Server, together with the prerequisite products.

However, your main source of information for installing the WebSphere Application Server on platforms such as Windows (NT/2000/XP ...), Linux, AIX, or Sun Solaris, should be the IBM Web pages that are kept up-to-date. You can find the IBM Web pages at:

http://www.ibm.com/software/webservers/appserv/library.html

Common Prerequisite Programs

Chapter 5. Installing and Operating the Java-Based Connector

This chapter describes how you:

- Install the *client-part* of the VSE/ESA Java-based connector, that is the VSE Connector Client, on your middle-tier (see page 25). A section is also provided describing how you can *uninstall* the VSE Connector Client, if this should be necessary (see page 29).
- Configure the *server-part* of the VSE/ESA Java-based connector, that is the VSE Connector Server, on the VSE/ESA host (see page 30).
- Start the VSE Connector Server on the VSE/ESA host (see page 36).
- Test the communication between the VSE Connector Client and VSE Connector Server (see page 37).
- Obtain a list of the commands that you can enter for the VSE Connector Server on the VSE/ESA host (see page 38).
- Enter commands for the VSE Connector Server on the VSE/ESA host (see page 38).
- Use the VSE Connector Server to maintain security for the VSE/ESA host resources (see page 38).

Installing the VSE Connector Client

This section describes how you install the VSE Connector Client on the middle-tier server of a 3-tier environment.

If you plan to implement a 2-tier environment using applets and Java applications that run on the Web clients, you must copy the VSE Java Beans part of the VSE Connector Client (file **VSEConnector.jar**) to each Web client on which the applets and Java applications are to run. The VSE Java Beans are required in order to establish connections between Web clients and the VSE Connector Server running on the VSE/ESA host.

The VSE Connector Client is included in VSE/ESA Central Functions and consists of one file **iesincon.w**.

Obtaining a Copy of the VSE Connector Client

Before you begin, you must already have installed the Java Development Kit (JDK) 1.3 or higher on the development platform where you plan to install the VSE Connector Client. If you do not have JDK 1.3 or higher installed, refer to "Installing and Configuring Java" on page 21 for details of how to install it.

To obtain a copy of the VSE Connector Client, you must decide if you wish to obtain the client from the Internet, or from VSE/ESA library PRD1.BASE.

To obtain the client *from the Internet*, start your Web browser and go to URL:

http://www.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/conmain.htm

From within the VSE Connector Client section, select (for the VSE/ESA version you require) **Details and Download**, then download the file **vsecon***nnn***.zip** to the

© Copyright IBM Corp. 2000, 2003

directory where you wish to install the VSE Connector Client. **Note:** *nnn* refers to the current VSE version (for example, **vsecon270.zip**).

To obtain the client *from the VSE/ESA library PRD1.BASE*, you use the FTP (file transfer program) utility of TCP/IP for VSE/ESA to download **iesincon.w** to the directory where you wish to install the VSE Connector Client.

Notes:

1. You must download **iesincon.w** in *binary*.
2. Make sure that Unix mode is ***turned off***. Otherwise **iesincon.w** will be downloaded in ASCII mode, even when you specify *binary*. *Unix mode* is one parameter of your VSE FTP daemon. Some FTP clients might *force* Unix mode to be turned on!. The example below shows how a successful transfer of **iesincon.w** was made using a batch FTP client. The place where the UNIX mode is set, is shown as bold.

```
c:\temp>ftp 9.164.155.2
Connected to 9.164.155.2.
220-TCP/IP for VSE -- Version 01.04.00 -- FTP Daemon
    Copyright (c) 1995,2000 Connectivity Systems Incorporated
220 Service ready for new user.
User (9.164.155.2:(none)): sysa
331 User name okay, need password.
Password:
230 User logged in, proceed.
ftp> cd prd1
250 Requested file action okay, completed.
ftp> cd base
250 Requested file action okay, completed.
ftp> binary
200 Command okay.
ftp> get iesincon.w
200 Command okay.
150-File: PRD1.BASE.IESINCON.W
    Type: Binary Recfm: FB Lrecl:    80 Blksize:      80
    CC=ON  UNIX=OFF RECLF=OFF TRCC=OFF CRLF=ON  NAT=NO
150 File status okay; about to open data connection
226-Bytes sent:       4,756,400
    Records sent:        59,455
    Transfer Seconds:        16.52 (    290K/Sec)
    File I/O Seconds:         3.94 (  1,548K/Sec)
226 Closing data connection.
4756400 bytes received in 17,12 seconds (277,91 Kbytes/sec)
ftp> bye
221 Service closing control connection.
c:\temp>ren iesincon.w vsecon.zip
```

Performing the VSE Connector Client Installation

To perform the installation of the VSE Connector Client, you must:

1. Unzip the file **vsecon.zip**, which contains these files:
   - install.class (contains the VSE Connector Client code)
   - install.bat (an install batch file for Windows)
   - install.cmd (an install batch file for OS/2)
   - install.sh (an install script for Linux/Unix)
2. Start the batch file (by double-clicking the file) that is applicable to your operating-system platform.
3. The installation process now begins, and you are guided through various installation menus.
4. The *Select Components to Install* window provides you with a choice:

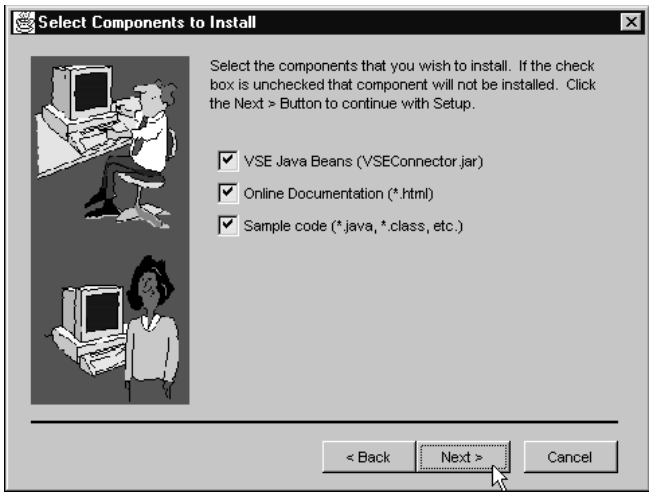


Figure 4. Selecting the Components of the VSE Connector Client That You Require

Select ...	**If you wish to ...**
VSE Java Beans	develop and run Java programs in 2- or 3-tier environments. *You will normally select this component to be installed.*
Online Documentation	use the online documentation (Javadoc and HTML-based) with the development of your own Java programs. This documentation is not for the end-users, only for developers.
Samples	use the online samples for developing your Java programs. These samples provide you with an introduction to developing your programs based upon the use of the Java-based connector and DB2-based connector. In addition, there are samples showing how you can develop programs that use the CICS Transaction Gateway and MQSeries Client for Java.

After making your selection, click **Next**.

5. The *Setup Complete* window is now displayed:

Select ...	**If you wish to ...**
Yes, I want to view the ReadMe file	open the ReadMe file that contains instructions on how to use the client.
Add VSEConnector.jar to local classpath	add the file VSEConnector.jar to your local Windows classpath. If you are installing on a platform other than Windows (NT/2000/XP), you will receive the message “please append the file VSEConnector.jar to your local classpath”. You must therefore manually perform this action yourself for the platform onto which you are installing.

After making your selection, click **Finish**. The installation now completes.

Note: Desktop icons are only created for Windows and OS/2 platforms (not for AIX, Linux, or z/OS).

6. To access the HTML-based documentation, you can now use your Web browser to open the file **VSEConnectors.html**.

Using the Online Documentation Options

After installing the VSE Connector Client on your middle-tier server, you can then access all information (including samples and source code) *via the online documentation*.

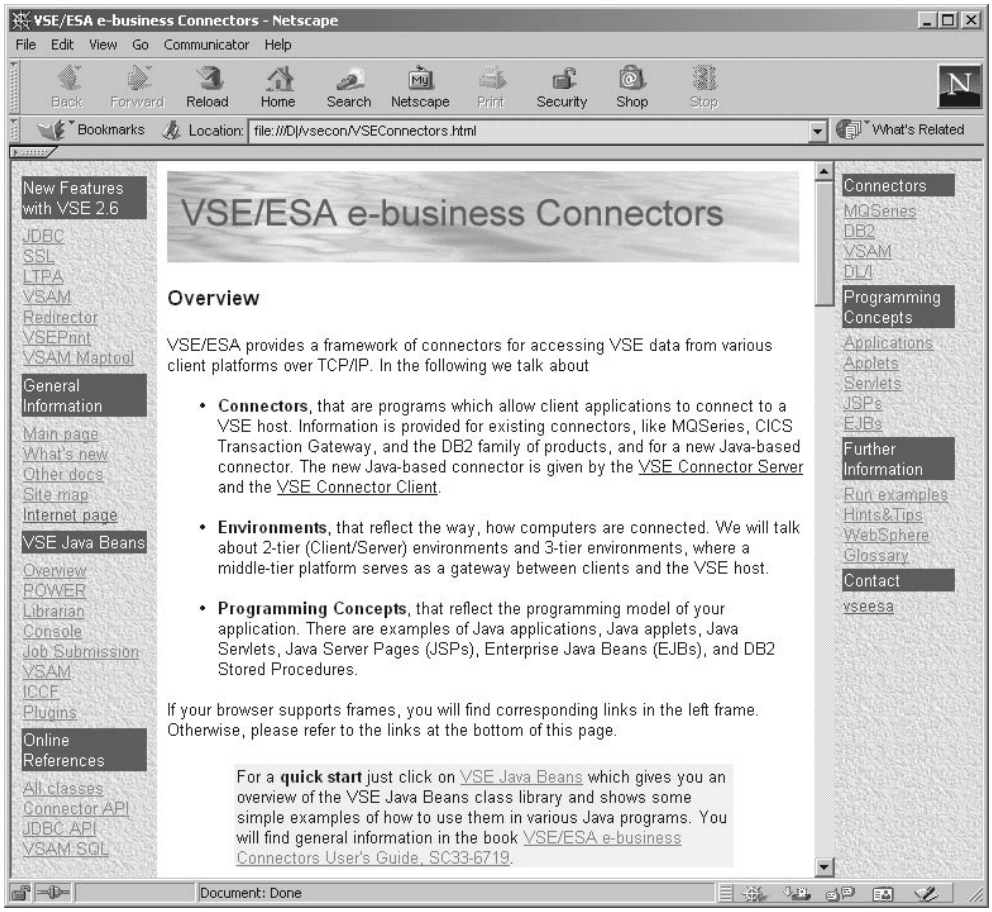


Figure 5. Online Help Options of the VSE Connector Client

The main window of the Online Documentation is shown in Figure 5, and contains links (on the left of this window) to:

- The Java-based connector and DB2-based connector, but also to the CICS connectivity and MQSeries connectivity setups. Here you find information and examples for these connectors and setups.
- Explanations of the concepts, such as Java applets, servlets, Java Server Pages (JSPs), Enterprise Java Beans (EJBs), and JDBC. Here you find information and examples related to these topics.
- Online reference (under **All classes**), including references to information about how you can extend the VSE Java Beans by writing your own *Plugins*.
- Samples that you can run directly from a Web browser. To run these samples, you must have TCP/IP for VSE/ESA and the VSE Connector Server running on your VSE/ESA host.

  Note: Before you can run any of the samples, or write your own Java programs that use the VSE Java Beans, you must include the **VSEConnector.jar** file in your local CLASSPATH variable.

Configuring for WebSphere Support

If you plan to implement the IBM WebSphere Application Server on your middle-tier, you should now complete the steps contained in the online documentation which describe how to configure your middle-tier for WebSphere. To find this information, click **Further Information** on the right of the window shown in Figure 5 on page 28, then select **WebSphere4**.

Now follow the steps described in the **WebSphere4** section.

Uninstalling the VSE Connector Client

You have three possible methods to uninstall the VSE Connector Client:

- Using the **Uninstall** option provided by the VSE Connector Client. From the Windows **Start**, select **Programs — VSE Connector — Uninstall**, and the *Uninstall* window is displayed:

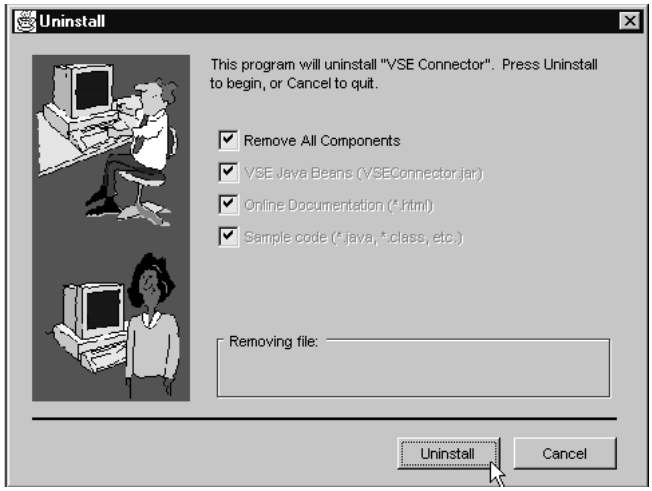


Figure 6. Uninstalling the VSE Connector Client

  Then select the components you wish to uninstall. If you select **Remove All Components**, after all components have been uninstalled, the icons on your desktop will also be removed.

  Note: For Windows (NT/2000/XP) and OS/2 systems, you are recommended to use this method to uninstall the client.

- Running the batch uninstall program **juninst** manually. Using this method, you can *only uninstall all components.* Your desktop icons will also be removed. Under Windows NT for example, this uninstall batch file is contained in the directory **\WINNT\Profiles**, and under OS/2 it is placed in the directory **\os2**.
- A third method is available under Windows NT/2000/XP only. From the Windows NT/2000/XP **Start**, select **Settings — Control Panel — Add/Remove Programs**. Now you can select **juninst.bat** and click **Install/Uninstall** to remove the client.

Configuring the VSE Connector Server

The VSE Connector Server is an application that runs in batch in one of your VSE/ESA partitions, and which implements a TCP/IP connection. A description is provided in "Overview of the VSE Connector Server" on page 5.

This section describes the jobs you use to configure your VSE Connector Server:

SKVCSSTJ A skeleton startup job.

SKVCSCAT A job to catalog the VSE Connector Server's configuration members. These are the skeletons contained within job SKVCSCAT:

SKVCSCFG
A VSE library member in which you specify the general settings for the VSE Connector Server

SKVCSLIB
A VSE library member in which you specify the VSE libraries that can be accessed by the VSE Connector Server

SKVCSPLG
A VSE library member in which you specify the server plugins to be loaded during startup of the VSE Connector Server.

SKVCSUSR
A VSE library member in which you specify the users, or groups of users, who can logon to the VSE Connector Server.

SKVCSSL
A VSE library member in which you configure the VSE Connector Server for Secure Sockets Layer (SSL) security.

Job SKVCSSTJ – Startup Job

You can find the skeleton job SKVCSSTJ in the VSE/ICCF library 59.

You use SKVCSSTJ to place a startup job (for starting the VSE Connector Server) in the VSE/POWER reader queue. SKVCSSTJ is also available as a Z book, which is loaded into the POWER reader queue during initial installation of VSE/ESA, or during a cold startup of VSE/ESA.

```
* $$ JOB JNM=CATSTVCS,DISP=D,CLASS=0
// JOB CATSTVCS       CATALOG STARTVCS AND LDVCS, LOAD STARTVCS
// EXEC LIBR,PARM='MSHP'
ACC S=IJSYSRS.SYSLIB
CATALOG  STARTVCS.Z        REPLACE=YES
$$$$ JOB JNM=STARTVCS,DISP=L,CLASS=R
$$$$ LST CLASS=A,DISP=D
// JOB STARTVCS START UP VSE CONNECTOR SERVER
// ID USER=VCSRV
// LIBDEF *,SEARCH=(PRD2.CONFIG,PRD1.BASE,PRD2.SCEEBASE)
// OPTION SYSPARM='00'
// EXEC IESVCSRV,PARM='DD:PRD2.CONFIG(IESVCSRV.Z)'
$$/*
$$/&
$$$$ EOJ
/+
CATALOG  LDVCS.PROC     REPLACE=YES DATA=YES
// EXEC DTRIINIT
   LOAD STARTVCS.Z
/*
/+
/*
// EXEC PROC=LDVCS        TO LOAD  VCS STARTUP  INTO RDR QUEUE
/&
* $$ EOJ
```

Figure 7. Job SKVCSSTJ (for Placing Startup Job in Reader Queue)

Job SKVCSCAT – Catalog Members

You can find the job SKVCSCAT in the VSE/ICCF library 59.

You use SKVCSCAT to catalog the configuration members listed in this section.

```
* $$ JOB JNM=VCSCAT,DISP=D,CLASS=0
// JOB VCSCAT CATALOG VCS CONFIGURATION MEMBERS
// EXEC LIBR,PARM='MSHP'
ACCESS S=PRD2.CONFIG
CATALOG IESVCSRV.Z REPLACE=Y
* $$ SLI ICCF=(SKVCSCFG),LIB=(YY)
/+
CATALOG IESLIBDF.Z REPLACE=Y
* $$ SLI ICCF=(SKVCSLIB),LIB=(YY)
/+
CATALOG IESUSERS.Z REPLACE=Y
* $$ SLI ICCF=(SKVCSUSR),LIB=(YY)
/+
CATALOG IESPLGIN.Z REPLACE=Y
* $$ SLI ICCF=(SKVCSPLG),LIB=(YY)
/+
CATALOG IESSSLCF.Z REPLACE=Y
* $$ SLI ICCF=(SKVCSSSL),LIB=(YY)
/+
/*
/&
* $$ EOJ
```

Figure 8. Job SKVCSCAT (for Cataloging Members for VSE Connector Server)

VSE Library Member SKVCSCFG – General Settings

You can find the skeleton SKVCSCFG in the VSE/ICCF library 59, which you use to specify your general settings for the VSE Connector Server.

```
; ****************************************************************
;      MAIN CONFIGURATION MEMBER FOR VSE CONNECTOR SERVER
; ****************************************************************
; TRACING SPECIFIC SETTINGS:
; - TRACEON    : A 32 BIT HEX VALUE PREFIXED WITH '0X'
;                0X00000000 IS OFF, 0XFFFFFFFF IN ON
; - TRACEFILE  : DESTINATION FOR TRACE MESSAGES
;                DD:SYSLOG, DD:SYSLST OR DD:LIB.SLIB(NAME.TYPE)
; ****************************************************************
  TRACEON      = 0X00000000  ; TRACE IS OFF
  TRACEFILE    = DD:SYSLOG   ; TRACE GOES TO SYSLOG
; ****************************************************************
; TCP/IP - SERVER SPECIFIC CONFIGURATIONS
; - SERVERPORT : THE TCP PORT WHERE THE SERVER IS LISTENING
; - MAXCLIENTS : THE MAXIMUM NUMBER OF CONCURRENT CLIENTS
; - SSLENABLE  : YES/NO - USE SECURE SOCKET LAYER
; ****************************************************************
  SERVERPORT   = 2893
  MAXCLIENTS   = 256
  SSLENABLE    = NO
; ****************************************************************
; DEFAULT CLASS FOR JOBS
; - DEFAULTCLASS : THE JOB CLASS WHERE UTILITY JOBS SHOULD RUN
; ****************************************************************
  DEFAULTCLASS = P
; ****************************************************************
; CODE PAGE CONVERSIONS
; - ASCII_CP  : ASCII CODE PAGE
; - EBCDIC_CP : EBCDIC CODE PAGE
; ****************************************************************
  ASCII_CP     = IBM-850
  EBCDIC_CP    = IBM-1047
; ****************************************************************
; SYSTEM LANGUAGE
; - LANGUAGE : THY SYSTEM'S LANGUAGE (E, G, S, J)
; ****************************************************************
  LANGUAGE = E
; ****************************************************************
; DESCRIPTION SENT AS IDENTIFY
; - DESCRIPTION : THIS STRING IS SENT AS IDENTIFY TO THE CLIENT
;                 CHANGE '<YOUR SYSTEM>' TO YOUR SYSTEM'S NAME
; ****************************************************************
; SUB CONFIGURATION MEMBERS NEEDED FOR VSE CONNECTOR SERVER
; - LIBRCFGFILE   : LIBRARY DEFINITION FILE. CONTAINS THE LIBRARIES
;                   THAT ARE VISIBLE FOR THE VSE CONNECTOR SERVER.
; - USERSCFGFILE  : USER/SECURITY CONFIG FILE. DEFINES ADDITIONAL
;                   SECURITY FOR USERS AND IP ADDRESSES.
; - PLUGINCFGFILE : PLUGIN CONFIG FILE. DEFINES THE PLUGINS THAT
;                   ARE LOADED AT SERVER STARTUP.
; - SSLCFGFILE    : SSL CONFIG FILE. DEFINES SSL PARAMETERS
; NOTE: YOU HAVE TO CHANGE THE NAMES AND LOCATIONS OF THESE MEMBERS
;       IN THIS MEMBER IF YOU MOVE THEM TO ANOTHER LIBRARY
; ****************************************************************
  LIBRCFGFILE   = DD:PRD2.CONFIG(IESLIBDF.Z)
  USERSCFGFILE  = DD:PRD2.CONFIG(IESUSERS.Z)
  PLUGINCFGFILE = DD:PRD2.CONFIG(IESPLGIN.Z)
  SSLCFGFILE    = DD:PRD2.CONFIG(IESSSLCF.Z)
```

Figure 9. Member SKVCSCFG (for Specifying General Settings for VSE Connector Server)

VSE Library Member SKVCSLIB – Specify Libraries to Be Accessed

You can find the skeleton SKVCSLIB in the VSE/ICCF library 59.

You use SKVCSLIB to specify the libraries that can be accessed by the VSE Connector Server. This skeleton consists of a list of libraries, which you can extend or restrict according to your own requirements. You must, however, enter each library name on a separate line.

```
PRD1
PRD2
PRIMARY
IJSYSRS
```

Figure 10. Member SKVCSLIB (for Specifying Libraries to Be Accessed by VSE Connector Server)

VSE Library Member SKVCSPLG – Specify Plugins to Be Loaded

You can find the skeleton SKVCSPLG in the VSE/ICCF library 59.

You use SKVCSPLG to specify the VSE Connector Server plugins to be loaded, when the VSE Connector Server is started.

This is the syntax of the PLUGIN statement:

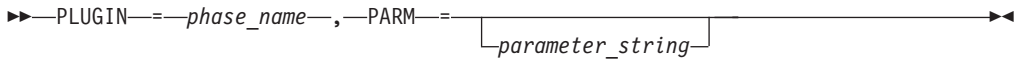


Figure 11 shows the member you use to specify VSE Connector Server plugins:

```
* ****************************************************************
* VSE CONNECTOR SERVER PLUGIN CONFIGURATION MEMBER
* ****************************************************************
* THE FOLLOWING PLUGINS ARE LOADED DURING STARTUP OF THE SERVER
* UNCOMMENT THE SAMPLES BELOW TO CHANGE THEM
* ****************************************************************
  PLUGIN=IESSAPLG,PARM=CICS=F2,CONS=IESA,TRANS=IEXM,EXIT=IESSAEXT
  PLUGIN=IESCOMPH,PARM=
  PLUGIN=IESHTOHP,PARM=
* PLUGIN=SAMPLE,PARM=MY PARAMTER STRING
* PLUGIN=<PHASE NAME>,PARM=<PARAMETER STRING>
```

Figure 11. Member SKVCSPLG (for Specifying Plugins for VSE Connector Server)

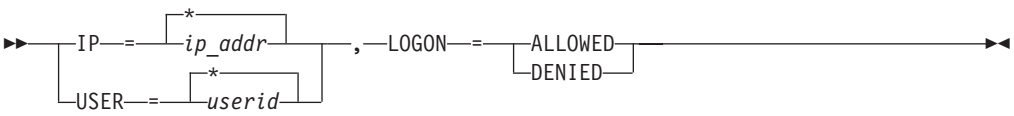
For details of how to write your own plugins, see Chapter 23, “Extending the Java-Based Connector”, on page 287.

VSE Library Member SKVCSUSR – Specify Logon Access

You can find the skeleton SKVCSUSR in the VSE/ICCF library 59.

You use SKVCSUSR to specify the users, or groups of users, who can logon to the VSE Connector Server. Using this skeleton, you can also prevent specific users or IP addresses from being able to access the VSE Connector Server.

This is the syntax of this member:




```
* ****************************************************************
* VSE CONNECTOR SERVER USER SECURITY CONFIGURATION MEMBER
* YOU CAN EITHER ALLOW OR DENY THE LOGON FOR A SPECIFIED
* USER OR IP OR GROUP OF USERS AND IP ADDRESSES.
* ****************************************************************
* USERS FROM THIS IP'S ARE ALLOWED TO LOGON
* UNCOMMENT THE SAMPLES AND MODIFY THEM
* ****************************************************************
IP   = *,             LOGON = ALLOWED
* IP = 9.164.123.456, LOGON = DENIED
* IP = 9.165.*      , LOGON = DENIED
* IP = 10.0.0.*     , LOGON = ALLOWED
* ****************************************************************
* THIS USERS ARE ALLOWED TO LOGON
* UNCOMMENT THE SAMPLES AND MODIFY THEM
* ****************************************************************
USER = *,             LOGON = ALLOWED
* USER = BOBY,        LOGON = ALLOWED
* USER = SYS*,        LOGON = DENIED
```

Figure 12. Member SKVCSUSR (for Specifying Logon Access to VSE Connector Server)

Notes:

1. If you do not make any entries in Figure 12, *no* access authorizations will be defined!.
2. You can use the wildcard (*) within an IP address or user name.

VSE Library Member SKVCSSSL – Configure for SSL

You can find the skeleton SKVCSSSL in the VSE/ICCF library 59.

You use SKVCSSSL to specify the:

- Version of SSL to be used.
- Name of the *VSE Keyring Library* to be used by the VSE Connector Server (see "Step 2: Catalog Keyring Set Into the VSE Keyring Library" on page 92 for details).
- Name of the *server certificate* to be used by the VSE Connector Server (see "Step 4: Obtain a Signed Server Certificate and Copy to Job CIALCERT" on page 98 for details).
- *Session timeout* in seconds – the number of seconds that the VSE Connector Server allows a VSE Connector Client to reconnect, without requiring a full *handshake*.

```
; ****************************************************************
;      SSL CONFIGURATION MEMBER FOR VSE CONNECTOR SERVER
; ****************************************************************

; ****************************************************************
; SSLVERSION   SPECIFIES THE MINIMUM VERSION THAT IS TO BE USED
;              POSSIBLE VALUES ARE:   SSL30 AND TLS31
; KEYRING      SPECIFIES THE SUBLIBRARY WHERE THE KEY FILES ARE
;              STORED.
; CERTNAME     NAME OF THE CERTIFICATE THAT IS USED BY THE SERVER
; SESSIONTIMEOUT NUMBER OF SECONDS THAT THE SERVER WILL USE TO
;              ALLOW A CLIENT TO RECONNECT WITHOUT PERFORMING A
;              FULL HANDSHAKE. (86440 SEC = 24 HOURS)
; AUTHENTICATION TYPE OF AUTHENTICATION. POSSIBLE VALUES ARE:
;              SERVER - SERVER AUTHENTICATION ONLY
;              CLIENT - SERVER AND CLIENT AUTHENTICATION
; ****************************************************************
SSLVERSION     = SSL30
KEYRING        = CRYPTO.KEYRING
CERTNAME       = SAMPLE
SESSIONTIMEOUT = 86440
AUTHENTICATION = SERVER

; ****************************************************************
; CIPHERSUITES SPECIFIES A LIST OF CIPHER SUITES THAT ARE ALLOWED
; ****************************************************************
CIPHERSUITES =   ; COMMA SEPARATED LIST OF NUMERIC VALUES
               01,  ; RSA512_NULL_MD5
               02,  ; RSA512_NULL_SHA
               08,  ; RSA512_DES40CBC_SHA
               09,  ; RSA1024_DESCBC_SHA
               0A,  ; RSA1024_3DESCBC_SHA
               62   ; RSA1024_EXPORT_DESCBC_SHA
```

Figure 13. Member SKVCSSSL (for Configuring the VSE Connector Server for SSL)

Starting the VSE Connector Server

You start the VSE Connector Server by releasing the startup job STARTVCS from the VSE/POWER reader queue. By default, the server then runs in a partition of dynamic class R.

STARTVCS is placed in the POWER reader queue either:

- During initial installation of VSE/ESA
- During a cold startup of VSE/ESA

TCP/IP Must Be Running

Before you start the VSE Connector Server, TCP/IP must be running. Refer to *TCP/IP for VSE/ESA IBM Program Setup and Supplementary Information* for details of how to start TCP/IP. Alternatively, you can refer to the online copy of this book at **http://www.ibm.com/servers/s390/os390/bkserv/vse.html**

When started, the VSE Connector Server listens to incoming TCP/IP traffic on port **2893**:

- The server's security subsystem uses RACROUTE requests to check logon and resource authorization.
- The server can handle multiple concurrent clients.

Figure 14 shows the startup job STARTVCS:

```
* $$ JOB JNM=STARTVCS,CLASS=R,DISP=L
* $$ LST CLASS=A,DISP=D
// JOB STARTVCS START UP VSE CONNECTOR SERVER
// LIBDEF *,SEARCH=(PRD2.CONFIG,PRD1.BASE,PRD2.SCEEBASE)
// ID USER=VCSRV
// OPTION SYSPARM='00'
// EXEC IESVCSRV,PARM='DD:PRD2.CONFIG(IESVCSRV.Z)'
/*
/&
* $$ EOJ
```

Figure 14. Startup Job STARTVCS (for Starting the VSE Connector Server)

Testing the Communication Between VSE Connector Client and Connector Server

Once you have completed the installation of the VSE Connector Client and VSE Connector Server, you can test if these two parts of the Java-based connector communicate correctly with each other by running some pre-compiled sample Java applications.

Before you begin, you must ensure that:

- TCP/IP for VSE/ESA is running on your VSE/ESA host.
- The VSE Connector Server is running on your VSE/ESA host.
- A TCP/IP connection exists between your local workstation and the VSE/ESA host.

The VSE Connector Client installation includes a set of batch files that run on Windows, OS/2, and Unix/Linux operating systems. They are stored in **samples**, which is a sub-directory below the directory where you installed the VSE Connector Client.

Each of the batch files contained in **samples** runs one sample *Java application* or *sample applet*. To run a sample, you should:

1. Open a command prompt.
2. Proceed to the **samples** sub-directory.
3. Run any batch file (you are not required to enter any parameters).

To obtain a list (with descriptions) of all the samples you can run, you should:

1. Start the VSE Connector Client.
2. Click **Run examples** in the left frame of this window.

Obtaining a List of VSE Connector Server Commands

To obtain a list of the VSE Connector Server commands that you can use, simply enter **Help** or **?** at the operator console. Figure 15 shows what will be displayed.

```
SYSTEM:  VSE/ESA              VSE/ESA 2.6      TURBO (01)       USER:  JSCH
                                                                TIME:  14:25:53
F7-0112 IPN300I Enter TCP/IP Command
msg startvcs,data=?
AR 0015 1I40I  READY
R1 0045 IESC1043I HELP COMMAND
R1 0045   HELP|?                                    PRINTS THIS MESSAGE
R1 0045   STATUS [ALL|CONFIG|CLIENTS|PLUGINS|VSAM] PRINTS STATUS INFORMATION
R1 0045   SENDMSG <USER(S)> <MESSAGE TEXT>          SENDS A MESSAGE TO A USER
R1 0045   SHUTDOWN [NOPROMPT]                      SHUTS DOWN THE SERVER
R1 0045   SETTRACE <TRACEFILE> <TRACELEVEL>        SET TRACING ON/OFF
R1 0045   STOP CLIENT <CLIENT-ID|ALL>              STOPS THE SPECIFIED CLIENT
R1 0045   CLOSE VSAM <SLOT-ID|ALL>                 CLOSES A VSAM CLUSTER


==>

1=HLP 2=CPY 3=END 4=RTN 5=DEL 6=DELS 7=RED 8=CONT 9=EXPL 10=HLD         12=RTRV
```

Figure 15. Displaying the Commands Provided by the VSE Connector Server

Entering a Command for the VSE Connector Server

To enter a command that the VSE Connector Server should process, you should use this command syntax:

```
msg <jobname>,data=<command>
```

where:

- *<jobname>* is the actual name of the VSE Connector Server startup job.
- *<command>* is one of the command strings shown in Figure 15.

For example:

```
msg startvcs,data=status
```

Maintaining Security Using the VSE Connector Server

When a Web application uses the VSE Java Beans class library (described in "Using VSE Java Beans to Connect to a VSE/ESA host" on page 162) to connect to the VSE/ESA host, the Web application must first perform a logon to the VSE/ESA host (supplying a valid user ID and password). Providing this logon is successful, the Web application obtains its access rights from the supplied user ID.

When the VSE Connector Server receives a request from a Web application, it passes this request to the currently-active VSE security manager (either the Basic Security Manager or External Security Manager). The security manager then checks whether or not the Web application should be allowed to access the requested resources or data.

The VSE Connector Server also uses a configuration file SKVCSUSR, which contains two lists of user IDs and IP addresses that:

1. *are* allowed to connect to the VSE Connector Server.
2. *are not* allowed to connect to the VSE Connector Server.

This is explained in "VSE Library Member SKVCSUSR – Specify Logon Access" on page 34.

When you write *servlets*, you might use a special VSE/ESA user ID which then allows the servlet to connect to the VSE Connector Server, without forcing the end-user to logon to the VSE Connector Server. Using this user ID, your servlets can restrict the type of requests, and also restrict access to data.

When you write *applets*, you should ***never*** "hard-code" any user IDs and passwords in the applet code: when the applet is downloaded to a Web browser and is stored in the Web browser's cache, this information could possibly be displayed by unauthorized persons (hackers).

TCP/IP for VSE/ESA also contains its own security functions. However, these functions are applicable only when using TCP/IP applications such as FTP, HTTP, or Telnet daemons. The VSE Connector Server *always* communicates with the VSE security manager.

Note: Security between the middle-tier and Web clients is established using SSL. SSL is, however, *not* used between the middle-tier and your VSE/ESA host: however, since this is an *intranet* connection, it should (by definition) be secure.

Chapter 6. Configuring DL/I for Access Via VSE Java Beans

This chapter describes how you configure your DL/I system so that it can be accessed via VSE Java Beans, and contains these main sections:

- "Host Installation Activities That Must Be Already Completed"
- "Step 1: Skeleton SKDLISMP – Define Sample Database" on page 42
- "Step 2: Customize CICS TS" on page 42

Three VSE Java Beans are used for DL/I access: VSEDli, VSEDliPsb, and VSEDliPcb. For descriptions of these VSE Java Beans, see "Contents of the VSE Java Beans Class Library" on page 155.

For an example of the coding you can use to access DL/I data via VSE Java Beans, see "Using VSE Java Beans for Accessing DL/I Data" on page 167.

Host Installation Activities That Must Be Already Completed

This section provides a summary of the installation activities that must already be completed on the VSE/ESA host, before you use VSE Java Beans to access DL/I data.

- The AIBTDLI interface must be installed for accessing DL/I data via VSE Java Beans. To use the AIBTDLI interface:
  - DL/I VSE 1.11 or later must be installed.
  - APAR PQ39683 for DL/I must be applied.
  - Your CICS/DLI system must have all databases (DBDs) that you wish to use defined to CICS, together with the AIBTDLI interface.
  - Your CICS/DLI system must have:
    - all PSBs defined in the DL/I online nucleus DLZNUCxx
    - an active MPS system.
  - The DL/I task termination exit *DLZBSEOT* (described in "Task Termination and Abend Handling" on page 329) must be resident in the SVA.

© Copyright IBM Corp. 2000, 2003

Step 1: Skeleton SKDLISMP – Define Sample Database

Use skeleton SKDLISMP (available in VSE/ICCF library 59) to define and load a DL/I sample database, if you:

- wish to use the IBM-supplied sample database for testing and learning purposes
- have not already defined and loaded this database during previous installations.

The skeleton SKDLISMP contains these jobs:

1. STJDBDGN (generate the DBDs for the sample database).
2. STJPSBGN (generate the PSBs for the sample database).
3. STJACBGN (generate the ACBs for the sample database).
4. STJPREOR (a pre-reorganization utility).
5. STJDFINV (define the cluster for the sample database).
6. STJLDCST (load the sample database).
7. STJPRRES (prefix resolution).
8. STJPRUPD (prefix update).

Step 2: Customize CICS TS

To access DL/I data via VSE Java Beans, you must customize a CICS TS - DL/I online system (providing you have not already done so during previous installation activities):

1. Configure your CICS/DLI online system, as described in:
   - Part 6 of the *DL/I Resource Definition and Utilities* manual.
   - The section "CICS – DL/I Tables – Requirements" of the *DL/I Resource Definition and Utilities* manual.
   - Section "Migrating to DL/I VSE 1.11 and the CICS Transaction Server for VSE/ESA 1.1" of the *DL/I 1.11 Release Guide*.
2. Define in the CICS FCT the sample database STDIDBP and other databases you wish to access.
3. Provide labels for the sample database STDIDBP (// DLBL STDIDBC ...) and other databases you wish to access.
4. Create a new DL/I online nucleus (DLZACT generation), by including all DL/I online programs and PSBs that you wish to use. The CICS/DLI mirror program DLZBPC00 must be authorized for PSB STBICLG, used for accessing the DL/I sample database. The CICS/DLI mirror program DLZBPC00 must also be authorized for any other PSBs you wish to use for accessing other DL/I databases.
5. Account for an increased number of concurrent DLZBPC00 mirror tasks in the CICS/DLI online system: you must accordingly adjust the MAXTASK and CMAXTSK parameters in the DLZACT generation.
6. Load the DL/I exit routine DLZBSEOT into the SVA.
7. Start an MPS system.

DLZMPX00 is SVA-eligible, and is used for accessing DL/I data via the *AIBTDLI interface* (see "Overview of the AIBTDLI Interface" on page 321 for an explanation of DLZMPX00 and the AIBTDLI interface). The AIBTDLI interface uses DLZMPX00 from the SVA (if it resides there), or loads DLZMPX00 into partition space and uses it from there.

Chapter 7. Installing the VSE Script Connector

This chapter describes how you install the *server-part* (the VSE Script Server) of the VSE Script Connector. The *client-part* (the VSE Script Client) is not installed here, but can be either:

- a user-written Java application.
- a user-written non-Java application.
- an existing office product, such as a word-processing or spreadsheet program.

This chapter consists of these main sections:

- "Step 1: Download the Install-File and Perform the Installation"
- "Step 2: Configure the VSEScriptServer Properties File" on page 45
- "Step 3: Configure the Connections Properties File" on page 46

Notes:

1. The VSE Script Server must run on a Java-enabled platform.
2. The client-part (the VSE Script Client) can run on either a Java-enabled platform **or** a non-Java-enabled platform.
3. You are **not** required to write your own Java code in order to use the VSE Script Connector. Instead, you simply write your own VSE Scripts, using the IBM-supplied VSE Script Language.

Related Sections:

- "Overview of the VSE Script Connector" on page 7
- Chapter 26, "Using the VSE Script Connector for Non-Java Access", on page 349

Step 1: Download the Install-File and Perform the Installation

You install the VSE Script Server on a Java-enabled platform.

Before you begin, you must already have installed the Java Development Kit (JDK) 1.3 or higher on the development platform where you plan to install the VSE Script Server. If you do not have JDK 1.3 or higher installed, refer to "Installing and Configuring Java" on page 21 for details of how to install it.

Step 1.1: Obtain a Copy of the VSE Script Server

You must decide if you wish to obtain a copy of the VSE Script Server either from the Internet, or from VSE/ESA library PRD1.BASE:

- To obtain the VSE Script Server from the Internet, start your Web browser and go to URL:

  `http://www.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/conmain.htm`

  From within the VSE Script Server section, select **Details and Download**, then download the file **vsescript***nnn***.zip** to the directory where you wish to install the VSE Script Server. **Note:** *nnn* refers to the current VSE version (for example, **vsescript270.zip**).
- To obtain the VSE Script Server from the VSE/ESA library PRD1.BASE, use the FTP (file transfer program) utility of TCP/IP for VSE/ESA to download **iesscrpt.w** to the directory where you wish to install it.

© Copyright IBM Corp. 2000, 2003

Notes:

1. You must download **iesscrpt.w** in *binary*.
2. Make sure that Unix mode is ***turned off***. Otherwise **iesscrpt.w** will be downloaded in ASCII mode, even when you specify *binary*. *Unix mode* is one parameter of your VSE FTP daemon. Some FTP clients might *force* Unix mode to be turned on!. The example below shows how a successful transfer of **iesscrpt.w** was made using a batch FTP client. The place where the UNIX mode is set, is shown as bold.

```
c:\temp>ftp 9.164.155.2
Connected to 9.164.155.2.
220-TCP/IP for VSE -- Version 01.04.00 -- FTP Daemon
    Copyright (c) 1995,2000 Connectivity Systems Incorporated
220 Service ready for new user.
User (9.164.155.2:(none)): sysa
331 User name okay, need password.
Password:
230 User logged in, proceed.
ftp> cd prd1
250 Requested file action okay, completed.
ftp> cd base
250 Requested file action okay, completed.
ftp> binary
200 Command okay.
ftp> get iesscrpt.w
200 Command okay.
150-File: PRD1.BASE.IESSCRPT.W
    Type: Binary Recfm: FB Lrecl:    80 Blksize:      80
    CC=ON  UNIX=OFF RECLF=OFF TRCC=OFF CRLF=ON  NAT=NO
150 File status okay; about to open data connection
226-Bytes sent:       2,378,200
    Records sent:        29,728
    Transfer Seconds:         8.26 (    290K/Sec)
    File I/O Seconds:         1.97 (  1,548K/Sec)
226 Closing data connection.
2378200 bytes received in  8,56 seconds (277,82 Kbytes/sec)
ftp> bye
221 Service closing control connection.
c:\temp>ren iesscrpt.w vsescript.zip
```

Step 1.2: Perform the Installation of the VSE Script Server

To perform the installation of the VSE Script Server, you must:

1. Unzip the file **vsescript.zip**, which contains these files:
   - install.class (contains the VSE Script Connector code)
   - install.bat (an install batch file for Windows)
   - install.cmd (an install batch file for OS/2)
   - install.sh (an install script for Linux/Unix)
2. Start the batch file (by double-clicking the file) that is applicable to your operating-system platform.
3. The installation process now begins, and you are guided through various installation menus.
4. The *Setup Complete* window is displayed when the installation is complete. After making your selection, click **Finish**. The installation now completes.

   Note: Desktop icons are only created for Windows and OS/2 platforms (not for AIX, Linux, and so on).
5. To access the HTML-based documentation, you can now use your Web browser to open the file **server.html** in the **/doc** subdirectory.

Step 2: Configure the VSEScriptServer Properties File

The properties file for the VSE Script Server is called **VSEScriptServer.properties**, which is a text file that you can edit using any text editor.

Comment lines begin with a # character in the first column.

These are the settings that you define in **VSEScriptServer.properties**:

messages= on | off
: If you define **messages= on**, all messages will be printed. If you define **messages= off**, messages will not printed (and "quiet mode" will be active).

listenport = *TCP/IP portnumber*
: Port number which the VSE Script Server uses to listen for requests.

maxconnections = *number*
: Maximum number of simultaneous connections that are allowed from VSE Script Clients.

scriptdirectory = ./scripts
: Root directory to contain the Scripts.

connectionconfig = Connections.properties
: Name of the connection configuration file (described in "Step 3: Configure the Connections Properties File" on page 46).

If you wish to use a different properties file than the default, you must specify the file name of your properties file as a parameter, using this command.

```
java com.ibm.vse.script.VSEScriptServer MyPropertiesFile.properties
```

Step 3: Configure the Connections Properties File

You define each connection from the VSE Script Server to the VSE/ESA host using a set of *five* properties (`connection.1.name`, up to `connection.1.password`) contained in a properties file. This is a text file that you can edit using any text editor. For an overview of how these connections are used, see Figure 175 on page 349.

Here is an example of a Connections properties file:

```
#Connection.properties

connection.1.name=vsecon
connection.1.ip=9.164.155.2
connection.1.port=2893
connection.1.userid=fran
connection.1.password=mypasswd

connection.2.name=vsefran
connection.2.ip=9.164.155.95
connection.2.port=2893
connection.2.userid=sysa
connection.2.password=mypasswd

...
connection.timeout=100
...
```

Comment lines begin with a # character in the first column.

These are the settings that you define in **Connections.properties** for *each* connection.

name = *name*
: The logical name you give to the VSE/ESA host. Your VSE Scripts will refer to this name when accessing the VSE/ESA host. For an example of how **name** is used, see "Step 4: Modify the Sample VSE Script" on page 357.

ip = *IP address*
: The TCP/IP address of the VSE/ESA host to which the VSE Script Server is to connect.

port = *number*
: The port number used by VSE Connector Server to listen for incoming requests.

userid = *user-ID name*
: The VSE/ESA user ID used by the VSE Script Server to build a connection to the VSE/ESA host.

password = *password*
: The password you wish to assign to the connection from the VSE Script Server to the VSE/ESA host. During the initial startup of the VSE Script Server, the password is encrypted and stored using the property **connection.n.encpassword**.

In addition, you can define in **Connections.properties** this global setting:

connection.timeout = *seconds*
: The time in seconds before an unused connection in the pool (to the VSE/ESA host) is closed and destroyed.

Chapter 8. Installing the VSAM Redirector Connector

This chapter describes how you install and implement the VSAM Redirector connector. It consists of these main sections:

- "How the VSAM Redirector Connector Works"
- "Installing and Configuring the VSAM Redirector Client" on page 50
- "Installing the VSAM Redirector Server" on page 60
- "IBM-Supplied Example of DB2-Related Handler" on page 64
- "IBM-Supplied Example of HTML-Related Handler" on page 65

For a general description of the VSAM Redirector connector, see "Overview of the VSAM Redirector Connector" on page 6.

How the VSAM Redirector Connector Works

The VSAM Redirector connector consists of:

- The *VSAM Redirector Client* which is installed on your VSE/ESA host. It is responsible for communication and redirection of VSAM requests.
- The *VSAM Redirector Server* which is installed on each remote *Java platform*, as shown in Figure 16 on page 48. It is a Java program that:
  1. Is responsible for connection-handling.
  2. Is responsible for data conversion.
  3. Builds the interface to the different file system *request handlers*.
  4. Generates error messages that are the same as those generated before VSAM datasets were migrated. Therefore your application programs do not need to be amended for changes in error-message handling.

VSAM request handlers (referred to simply as *request handlers*) are also stored on the Java platform, and have a common interface. They are specific to the file system with which they work. For all connections, information about the file and the request are sent to the request handler.

Figure 16 on page 48 provides an overview of how the VSAM Redirector connector can be used to redirect VSAM requests from an application running on the VSE/ESA host, to a DB2 database stored on a Java platform. It uses the IBM-supplied *DB2Handler* on the Java platform. Each handler can decide at each of the above requests, which processing is required for the remote data.

© Copyright IBM Corp. 2000, 2003

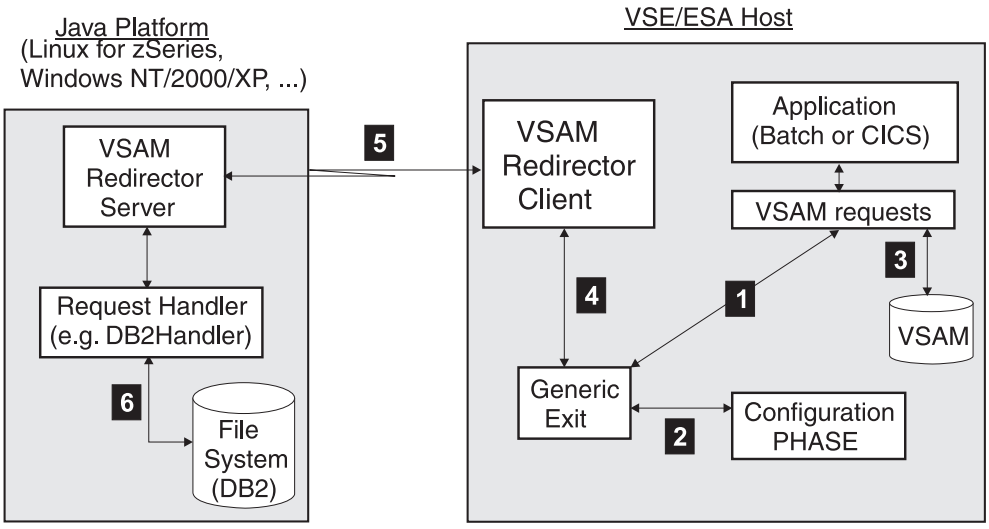


Figure 16. Example of How the VSAM Redirector Connector Is Used

The general processing shown in Figure 16 is as follows:

1. An application running on the VSE/ESA host issues a VSAM command (for example, to open a VSAM file). The request is passed to the *generic exit* (IKQVEX01.PHASE).
2. The generic exit checks whether the VSAM file has been set up to be redirected. To do so, it checks the configuration phase (IESRDCFG.PHASE).
3. If the VSAM file has not been redirected to another Java platform (and is therefore still stored as a VSAM record on the VSE/ESA host), the generic exit returns and indicates that the VSAM file has not been redirected. It also indicates that the generic exit should not be called again for any request against *this* VSAM file. Normal VSAM processing then continues.
4. If the VSAM file *has* been redirected to another Java platform, the generic exit (IESREDIR.PHASE) calls the VSAM Redirector Client.
5. The VSAM Redirector Client establishes a connection to the VSAM Redirector Server running on the Java platform, and forwards the VSAM file request, together with any data (such as a VSAM record contents) to the VSAM Redirector Server.
6. The VSAM Redirector Server uses the request handler that is specified in the configuration phase, to perform access to the target file system or database. In Figure 16, the specified request handler is *DB2Handler* (which is supplied by IBM during the installation of the VSAM Redirector Server). *DB2Handler* implements access to a DB2 database.

VSAM Integration Considerations

For VSAM internal processing (such as the `POINT to END OF FILE`) changes have been made to VSAM so that the VSAM Redirector Client can perform its processing. The original VSAM cluster of a redirected file must, however, still exist on the VSE/ESA host. It must also contain a dummy record (which you can insert using, for example, the DITTO utility).

In order for all VSAM requests to be redirected, the exit phase (IKQVEX01.PHASE) must return a return code of -1. This indicates to VSAM that no VSAM processing at all is required against this file.

If the exit phase cannot open the configuration phase (IESRDCFG.PHASE), or if a TCP/IP connection is not available, the exit phase reports this to VSAM using two error definitions:

- If the exit phase returns -3 to VSAM, this is converted to a DDNAME NOT FOUND error message, which indicates that the exit phase was unable to connect to the specified Java platform.
- The second error code is -4, converted to a `UNABLE TO CDLOAD` error, which means that the exit phase was unable to load either the VSAM Redirector Client or the configuration phase.

For further details about processing and return-code changes, refer to the online documentation provided with the VSAM Redirector Server.

Installing and Configuring the VSAM Redirector Client

The VSAM Redirector Client is automatically installed on your *VSE/ESA host* during the installation of VSE/ESA, and consists of three PHASE files:
- IKQVEX01.PHASE (the *exit phase*)
- IESREDIR.PHASE (the *VSAM Redirector Client*)
- IESRDCFG.PHASE (the *configuration phase*)

IKQVEX01 is called each time a VSAM OPEN request is made. IKQVEX01 then decides if a cluster is to be redirected or not:

- If a cluster *is* to be redirected, IKQVEX01 loads and starts IESREDIR.PHASE. This phase handles the network operations, and communicates with the VSAM Redirector Server. All relevant information must be specified in the configuration phase IESRDCFG.PHASE (see "Step 1: Enable the VSAM Redirector Client on VSE/ESA" for details).
- If a cluster *is not* to be redirected, normal VSAM processing takes place.

An overview of this processing is shown in Figure 16 on page 48.

To configure the VSAM Redirector Client, you must follow these steps:
- "Step 1: Enable the VSAM Redirector Client on VSE/ESA"
- "Step 2: Decide Upon Your Redirection Mode" on page 51
- "Step 3 (Optional): Transfer Your VSAM Data" on page 56
- "Step 4: Create the Configuration Phase" on page 56

Fast Service Upgrade Considerations

If you are performing a Fast Service Upgrade (FSU), you must run the configuration job SKRDCFG to update the IKQVEX01.PHASE in library PRD2.CONFIG.

Step 1: Enable the VSAM Redirector Client on VSE/ESA

The VSAM Redirector connector is based upon the existing VSAM Data Access (VDA) exit. The VDA exit is represented by the dummy exit phase IKQVEX01.PHASE, which is shipped in library IJSYSRS.SYSLIB.

The VSAM Redirector Client also uses the VDA exit. Therefore to avoid over-writing any existing changes you have made to IKQVEX01.PHASE, all phases that belong to the VSAM Redirector Client are shipped in library PRD1.BASE.

To enable the VSAM Redirector Client, you must use skeleton SKRDCFG in Library 59 to configure your VSAM Redirector Client. You use this skeleton to
1. Assemble/link the member IESRDCFG.PHASE, and store it in library PRD2.CONFIG.
2. Load the IESRDCFG.PHASE into the SVA (optional).
3. Copy IESVEX01.PHASE to the library PRD2.CONFIG with the name IKQVEX01.PHASE, to activate the exit phase.
4. Load the IKQVEX01.PHASE into the SVA (optional).

   Note: Normally, your original IKQVEX01.PHASE will have been loaded into the SVA. To enable the VSAM Redirector Client, you must replace it in the SVA with the new phase loaded from PRD2.CONFIG. Alternatively, to replace the IKQVEX01 phase you can re-IPL your VSE/ESA system.

If you wish to activate a new configuration while the system is running (that is, without restarting your VSAM applications), after completing the steps above you must then:

5. Load the IESRDANC.PHASE into the SVA (you need to do this only once, of course).
6. Execute program IESRDLDA. However, IESRDCFG.PHASE must be *already loaded* into the SVA (as in Step 2, above). Program IESRDLDA will then activate the new copy of IESRECFG so that it is the current configuration, and the changes become immediately active. However, VSAM files will not be changed if they are already open. To activate the changes for any VSAM files that are open, you must close and then reopen such files.

The VSAM Redirector connector uses TCP/IP as its communication protocol. Therefore, any of your applications that you wish to use with the VSAM Redirector connector must be modified accordingly. You must add the statement:

```
// OPTION SYSPARM='nn'
```

to the JCL for these jobs. The value `'nn'` (the system ID) is contained in the:

```
// EXEC IPNET,SIZE=IPNET,PARM='ID=nn,INIT=... '
```

statement of your TCP/IP startup job.

Note: The value of `'00'` is the default for the system ID. If you accept this default, you are *not* required to add the statement `// OPTION SYSPARM='00'`.

Step 2: Decide Upon Your Redirection Mode

There are two possible modes in which you can use the VSAM Redirector Client:

1. Working with data that resides on another platform.
2. Synchronizing your *existing* VSAM data with data that resides on another platform.

You set the mode of operation using the OWNER parameter within the configuration phase. See “Step 4: Create the Configuration Phase” on page 56 for details.

Notes:

1. For each redirection mode, the VSAM cluster must be defined to the VSE/ESA system. Therefore, when the VSAM `OPEN` request is executed for this cluster, the VSAM Redirector Client must obtain the following information from the VSAM cluster:
   - Cluster type
   - Key position
   - Key length
   - Maximum record length
2. When you open a redirected file (on a Java platform) for `READ` processing, the original VSAM cluster must still be defined on the VSE/ESA host, and must contain at least one ‘dummy’ record. Otherwise, a VSAM error will occur when the `OPEN` request is processed.
3. `READ` requests will not generate any data transfer to the remote Java platform.

Mode 1. Working With Data Residing On Another Platform

If you use this redirection mode (`OWNER=REDIRECTOR`), your programs that work with VSAM data will never perform any VSAM access operations. All requests are redirected to the VSAM Redirector Client (**iesredir.phase**), which then connects to the VSAM Redirector Server running on a Java platform. The VSAM Redirector Server then performs the request.

Note: You cannot chain exits if any one of these exits has `OWNER` set to `REDIRECTOR`.

Figure 17 shows the flow of control when a VSAM `PUT` is executed.

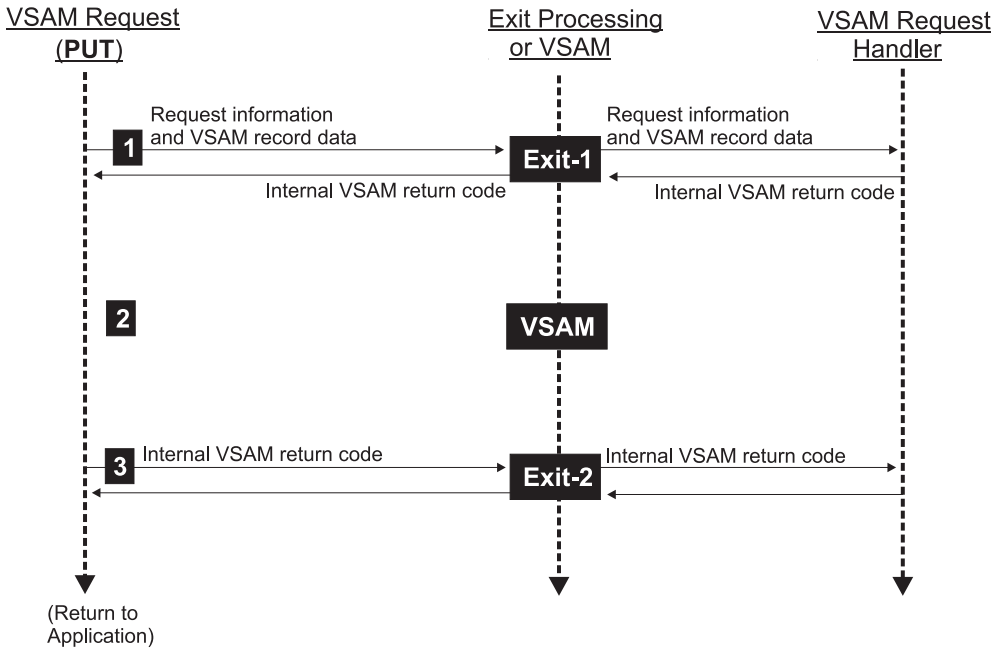


Figure 17. Flow of Control for VSAM PUT Request

1. VSAM calls **Exit-1** *before* VSAM starts to process the request.
2. VSAM does not process any requests.
3. VSAM calls **Exit-2** *after* VSAM has finished processing the request.

Figure 18 on page 53 shows the flow of control when a VSAM `GET` is executed.

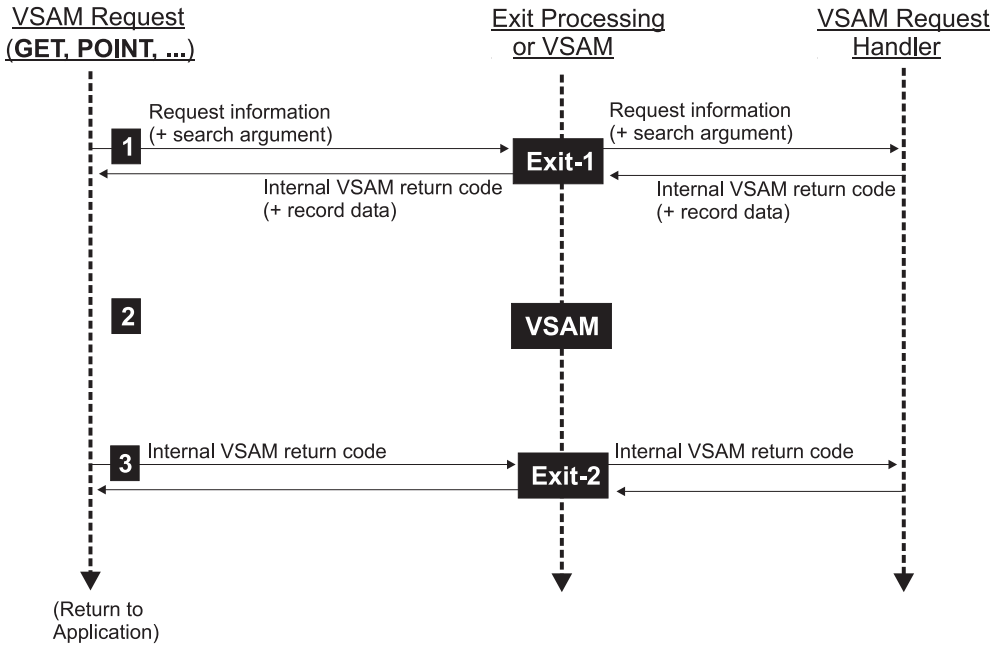


Figure 18. Flow of Control for VSAM GET Request

1. VSAM calls `Exit-1` *before* VSAM starts to process the request.
2. VSAM does not process any requests.
3. VSAM calls `Exit-2` *after* VSAM has finished processing the request.

Mode 2. Synchronizing Your Existing VSAM data

If you use this mode (`OWNER=VSAM`), your programs that work with VSAM data will perform a VSAM access and redirected access. Each VSAM request issues two requests to the VSAM Redirector Client (**iesredir.phase**).

Figure 19 shows the flow of control when a VSAM `PUT` is executed.

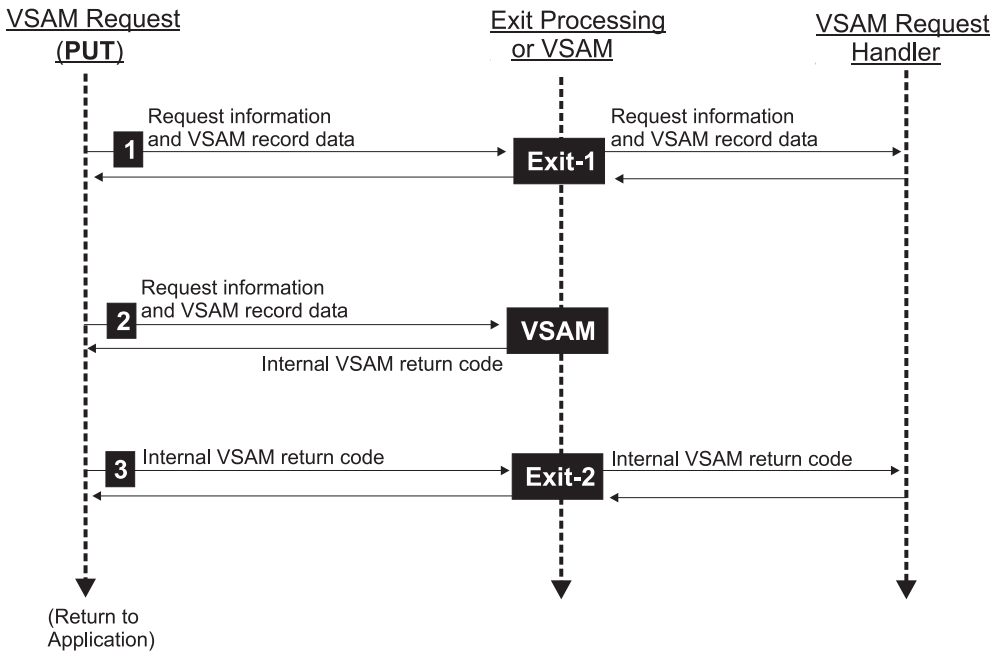


Figure 19. Flow of Control for VSAM PUT Request

1. VSAM calls `Exit-1` *before* VSAM starts to process the request.
2. VSAM processes the request.
3. VSAM calls `Exit-2` *after* VSAM has finished processing the request.

Figure 20 on page 55 shows the flow of control when a VSAM `GET` is executed.

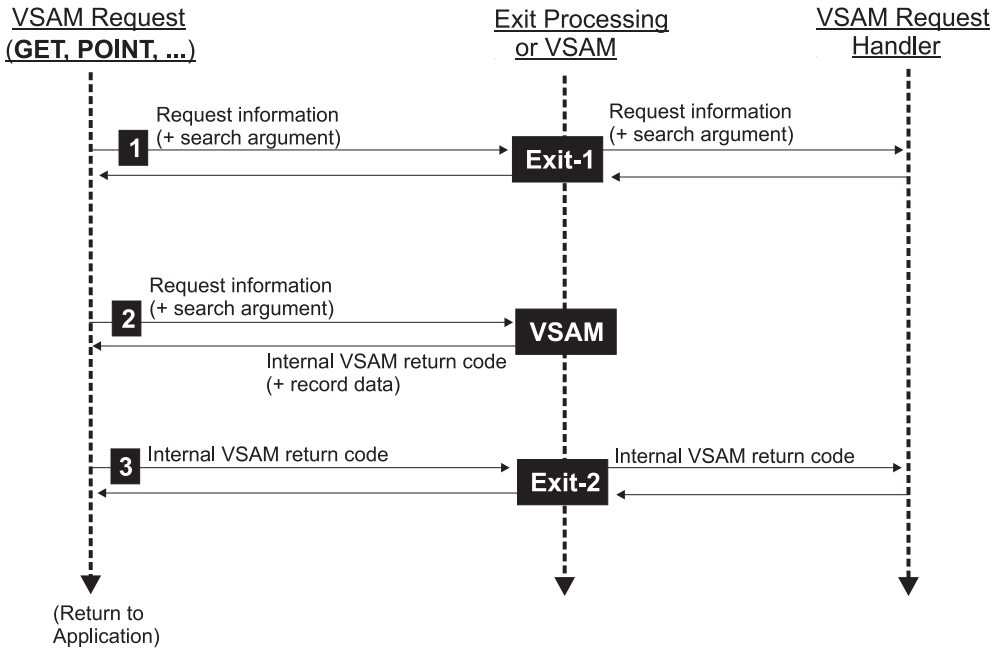


Figure 20. Flow of Control for VSAM GET Request

1. VSAM calls `Exit-1` *before* VSAM starts to process the request.
2. VSAM processes the request.
3. VSAM calls `Exit-2` *after* VSAM has finished processing the request.

Step 3 (Optional): Transfer Your VSAM Data

To use your existing VSAM datasets with the VSAM Redirector connector, you can migrate your VSAM data to the file system you require (for example, DB2 format). For details of how to map VSAM data to a relational structure suitable for DB2 processing, refer to Chapter 15, “Mapping VSE/VSAM Data to a Relational Structure”, on page 129.

To transfer your data from the VSE/ESA host to your target Java platform’s file system, you can use the IDCAMS REPRO utility:

1. Define a VSAM cluster (that has the same properties as the source cluster) as the target for your redirection / transfer process.
2. Change the configuration phase, so that your VSAM cluster is redirected (`OWNER=REDIRECTOR`). For details, see “Step 4: Create the Configuration Phase”.
3. Ensure that the VSAM Redirector Server, and the handler you have defined in the configuration phase, are both running on your your target Java platform.
4. Copy the data into your redirected VSAM cluster, using the IDCAMS REPRO utility. After completing this action, your data will now be stored on the target Java platform’s file system.

Step 4: Create the Configuration Phase

VSE library 59 contains an example skeleton SKRDCFG which you can use to create a configuration phase. Figure 21 on page 58 shows the sample job skeleton SKRDCFG. Below are are the parameters you set in this job.

Mandatory Parameters

CATALOG= VSAM catalog name of the file to be redirected. You can use the wildcard * in your parameter definition. In this case, you must also set `CLUSTER=*`. However, be aware that if you use `CATALOG=*` this will redirect *all* catalogs (and clusters contained in those catalogs).

Notes:

1. If the master catalog is redirected, you might not be able to startup your VSE/ESA system!.
2. Entries that contain wildcards will only used providing no other matching entry can be found.

CLUSTER= VSAM cluster name. You can use the wildcard * in your parameter definition. However, be aware that if you use `CLUSTER=*` this will redirect *all* clusters belonging to the specified catalog.

EXIT= Name of the exit phase to use.

- If you specify `EXIT=IESREDIR`, all the parameters listed below apply.
- If you specify another value for `EXIT` (for example, a vendor-provided exit), no further parameters apply.

Optional Parameters

Using these optional parameters, you can specify additional filters when specifying the clusters that are to be redirected.

CATDD= The label name of the catalog. The default is CATDD='*'. If you enter a value for CATDD (other than the default), both the CATALOG and CATDD will be used for checking if the file is to be redirected.

CLUDD= The label name of the cluster. The default is CLUDD='*'. If you

specify a value for CLUDD (other than the default), both the CLUSTER and CLUDD are used to check if the file is to be redirected.

PART= The partition ID (for example F4) of the partition from which redirection is *only* possible. The default value is PART='*'.

NOTPART= The partition ID (for example F4) of the partition from which redirection is *not* possible. The default is that *all* partitions are available for redirection.

Parameters That Apply When EXIT=IESREDIR

OWNER= Can take one of these values:

REDIRECTOR

All requests are redirected to the VSAM Redirector Client (**IESREDIR.PHASE**), which then connects to the VSAM Redirector Server running on a Java platform. The VSAM Redirector Server then performs the request.

Note: You cannot chain exits if any one of these exits has `OWNER=REDIRECTOR`.

VSAM

Dual processing occurs (both VSAM processing and redirecting requests to the VSAM Redirector Client).

IP= A mandatory parameter. The IP address of the server where the VSAM Redirector Server is installed, and to which a connection is to be made.

PORT= An optional parameter. The port number of the server where the VSAM Redirector Server is installed, and to which a connection is to be made. The default port number used with the VSAM Redirector Server is **2387**, which has been assigned by the Internet Assigned Numbers Authority (IANA).

HANDLER= A mandatory parameter. The name of the Java class to be started, which represents the request handler to be used with this configuration entry.

OPTIONS= A mandatory parameter. A string containing data to be transferred to the request handler, as an option string. You can insert your own settings here. In the example shown in Figure 21 on page 58, this string contains information required by the IBM-supplied *DB2Handler* (DB/2 system, username, password, and so on). If you specify a blank value for this parameter (‘ ’), the request handler will receive blanks as the option string.

Parameters That Apply When OWNER=VSAM

IGNOREERROR=

An optional parameter, whose default is `IGNOREERROR=NO`. If you set `IGNOREERROR=YES`, a `VSAM OPEN` request will not return an error, even if the VSAM Redirector Server cannot be accessed.

PUTREQONLY=

An optional parameter, whose default is `PUTREQONLY=NO`. If you set `PUTREQONLY=YES`, only `INSERT` and `UPDATE` requests will be redirected to the VSAM Redirector Server. You might find this parameter useful if you want to collect statistics about your request handler, excluding requests such as `POINT`, `GET`, and so on.

```
* $$ JOB JNM=RDCONFIG,CLASS=A,DISP=D
// JOB RDCONFIG GENERATE REDIRECTOR CONFIG PHASE
* ********************************************************************
* STEP 1: ASSEMBLE AND LINK THE CONFIG TABLE                         *
* ********************************************************************
// LIBDEF *,CATALOG=PRD2.CONFIG
// LIBDEF *,SEARCH=PRD1.BASE
// OPTION ERRS,SXREF,SYM,NODECK,CATAL,LISTX
   PHASE IESRDCFG,*,SVA
// EXEC ASMA90,SIZE=(ASMA90,64K),PARM='EXIT(LIBEXIT(EDECKXIT)),SIZE(MAXC
               -200K,ABOVE)'
IESRDCFG CSECT
IESRDCFG AMODE ANY
IESRDCFG RMODE ANY
*
         IESRDENT CATALOG='VSESP.USER.CATALOG',                       X
               CLUSTER='MY.TEST.CLUSTER1',                            X
               EXIT='IESREDIR',                                       X
               OWNER=REDIRECTOR,                                      X
               IP='10.0.0.1',                                         X
               HANDLER='com.ibm.vse.db2handler.DB2Handler',           X
               OPTIONS='db2url=jdbc:db2:redir;db2user=hugo;           X
                        db2password=hugospw;db2table=mydata'
*
         IESRDENT CATALOG='VSESP.USER.CATALOG',                       X
               CLUSTER='MY.TEST.CLUSTER2',                            X
               EXIT='VENDOREX'
*
         END
/*
// IF $MRC GT 4 THEN
// GOTO NOLINK
// EXEC LNKEDT,PARM='MSHP'
/. NOLINK
/*
* ********************************************************************
* STEP 2: LOAD THE IESRDCFG.PHASE INTO THE SVA (OPTIONAL)            *
* ********************************************************************
* LIBDEF *,SEARCH=PRD2.CONFIG
* SET SDL
* IESRDCFG,SVA
* /*
* ********************************************************************
* STEP 3: COPY IESVEX01.PHASE INTO PRD2.CONFIG AS IKQVEX01           *
* ********************************************************************
// EXEC LIBR,PARM='MSHP'
  CONNECT S=PRD1.BASE:PRD2.CONFIG
  COPY IESVEX01.PHASE:IKQVEX01.PHASE REPLACE=YES
/*
```

Figure 21. Job to Produce a Configuration Phase for the VSAM Redirector Connector (Part 1 of 2)

```
* ********************************************************************
 * STEP 4: LOAD THE IKQVEX01.PHASE INTO THE SVA (OPTIONAL)            *
 * ********************************************************************
 * LIBDEF *,SEARCH=PRD2.CONFIG
 * SET SDL
 * IKQVEX01,SVA
 * /*
 * ********************************************************************
 * STEP 5: LOAD THE IESRDANC.PHASE INTO THE SVA (OPTIONAL)            *
 *         THIS SHOULD BE DONE ONLY ONCE  !!                          *
 * ********************************************************************
 * // LIBDEF *,SEARCH=PRD2.CONFIG
 * SET SDL
 * IESRDANC,SVA
 * /*
 * ********************************************************************
 * STEP 6: REGISTER THE CURRENT CONFIGURATION PHASE                    *
 * ********************************************************************
 * // LIBDEF *,SEARCH=PRD1.BASE
 * // EXEC IESRDLDA
 * /*
 /&
 * $$ EOJ
```

Figure 21. Job to Produce a Configuration Phase for the VSAM Redirector Connector (Part 2 of 2)

Installing the VSAM Redirector Server

The main activities that you must perform on each Java platform where you plan to install migrated datasets, are:

- "Step 1: Download the Install-File and Perform the Installation"
- "Step 2: Configure the Properties File" on page 61
- "Step 3: Implement a VSAM Request Handler" on page 62

Step 1: Download the Install-File and Perform the Installation

You install the VSAM Redirector Server on a Java-enabled platform.

Before you begin, you must already have installed the Java Development Kit (JDK) 1.2.2 or higher on the development platform where you plan to install the VSAM Redirector Server. If you do not have JDK 1.2.2 or higher installed, refer to "Installing and Configuring Java" on page 21 for details of how to install it.

Step 1.1: Obtain a Copy of the VSAM Redirector Server

You must decide if you wish to obtain a copy of the VSAM Redirector Server either from the Internet, or from VSE/ESA library PRD1.BASE:

- To obtain the VSAM Redirector Server from the Internet, start your Web browser and go to URL:

  http://www.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/conmain.htm

  From within the VSAM Redirector Server section, select **Details and Download**, then download the file **redir***nnn***.zip** to the directory where you wish to install the VSAM Redirector Server. **Note:** *nnn* refers to the current VSE version (for example, **redir270.zip**).

- To obtain the VSAM Redirector Server from the VSE/ESA library PRD1.BASE, use the FTP (file transfer program) utility of TCP/IP for VSE/ESA to download **iesvsmrd.w** to the directory where you wish to install it.

  Notes:

  1. You must download **iesvsmrd.w** in *binary*.
  2. Make sure that Unix mode is ***turned off***. Otherwise **iesvsmrd.w** will be downloaded in ASCII mode, even when you specify *binary*. *Unix mode* is one parameter of your VSE FTP daemon. Some FTP clients might *force* Unix mode to be turned on!. The example below shows how a successful transfer of **iesvsmrd.w** was made using a batch FTP client. The place where the UNIX mode is set, is shown as bold.

```
c:\temp>ftp 9.164.155.2
Connected to 9.164.155.2.
220-TCP/IP for VSE -- Version 01.04.00 -- FTP Daemon
    Copyright (c) 1995,2000 Connectivity Systems Incorporated
220 Service ready for new user.
User (9.164.155.2:(none)): sysa
331 User name okay, need password.
Password:
230 User logged in, proceed.
ftp> cd prd1
250 Requested file action okay, completed.
ftp> cd base
250 Requested file action okay, completed.
ftp> binary
200 Command okay.
ftp> get iesvsmrd.w
200 Command okay.
150-File: PRD1.BASE.IESVSMRD.W
```

```
    Type: Binary Recfm: FB Lrecl:    80 Blksize:      80
    CC=ON  UNIX=OFF RECLF=OFF TRCC=OFF CRLF=ON  NAT=NO
150 File status okay; about to open data connection
226-Bytes sent:       4,756,400
    Records sent:        59,455
    Transfer Seconds:        16.52 (    290K/Sec)
    File I/O Seconds:         3.94 (  1,548K/Sec)
226 Closing data connection.
4756400 bytes received in 17,12 seconds (277,91 Kbytes/sec)
ftp> bye
221 Service closing control connection.
c:\temp>ren iesvsmrd.w redir.zip
```

Step 1.2: Perform the Installation of the VSAM Redirector Server

To perform the installation of the VSAM Redirector Server, you must:

1. Unzip the file **redir.zip**, which contains these files:
   - install.class (contains the VSAM Redirector Server code)
   - install.bat (an install batch file for Windows)
   - install.cmd (an install batch file for OS/2)
   - install.sh (an install script for Linux/Unix)
2. Start the batch file (by double-clicking the file) that is applicable to your operating-system platform.
3. The installation process now begins, and you are guided through various installation menus.
4. The *Setup Complete* window is displayed when the installation is complete:

Select ...	If you wish to ...
Yes, I want to view the ReadMe file	open the ReadMe file that contains instructions on how to use the VSAM Redirector Server.
Add VSAMRedir.jar to local classpath	add the file VSAMRedir.jar to your local Windows classpath. If you are installing on a platform other than Windows NT/2000/XP, you will receive the message "please append the file VSAMRedir.jar to your local classpath". You must therefore manually perform this action yourself for the platform onto which you are installing.

   After making your selection, click **Finish**. The installation now completes.

   Note: Desktop icons are only created for Windows and OS/2 platforms (not for AIX, Linux, or z/OS).
5. To access the HTML-based documentation, you can now use your Web browser to open the file **...** **(***Redirector root directory***)/doc/redir.html**.

Step 2: Configure the Properties File

The properties file for the VSAM Redirector Server is called **VSAMRedirectorServer.properties**, which is a text file that you can edit using any text editor.

Comment lines begin with a # character in the first column.

These are the settings that you define in **VSAMRedirectorServer.properties**:

messages= on | off

If you define **messages= on**, all messages will be printed. If you define **messages= off**, messages will not printed (and "quiet mode" will be active).

listenport = *TCP/IP portnumber*
Port number which the VSAM Redirector Server uses to listen for requests.

maxconnections = *number*
Maximum number of connections that are allowed from VSAM Redirector Clients.

codepagetranslator = com.ibm.vse.server.DefaultTranslator
Codepage translator class to be used for converting Strings from:
- EBCDIC into ASCII
- ASCII into EBCDIC

The IBM-supplied default is shown above.

Step 3: Implement a VSAM Request Handler

VSAM request handlers (referred to simply as *request handlers*) are used by the VSAM Redirector Server as shown in Figure 16 on page 48, and are programmed in Java. This section describes how request handlers are implemented, under these headings:
- "Coding VSAM Logic and Parameters"
- "Calling a VSAM Request Handler" on page 63
- "Error Reporting" on page 63
- "Datatype Conversions" on page 63
- "Getting a Map Dynamically Into Your Request Handler" on page 64
- "IBM-Supplied Example of DB2-Related Handler" on page 64
- "IBM-Supplied Example of HTML-Related Handler" on page 65

Note: Refer to "Code for HTML-Related Request Handler" on page 65 for an explanation of the code used to implement this request handler.

Coding VSAM Logic and Parameters

To redirect VSAM data into other data formats (SQL database, flat file, and so on) so that this data is transparent to existing applications, you must simulate all VSAM behavior in your request handler. Therefore you must include positioning and error-reporting logic in your request handler.

You can program your own request handlers and include them for use by the VSAM Redirector connector. You must also code an interface which your request handler provides, by implementing *case methods* to handle:
- OPEN requests
- CLOSE requests
- Record requests, which are:
  - GET
  - PUT (UPDATE+INSERT)
  - ENDRQ
  - POINT

If you wish to write a request handler that is to copy VSAM files (in the same way as the IBM-supplied *DB2Handler*), your request handler must:

1. Copy the VSAM logic.
2. Respond in the same way as VSAM itself would.

For details of how to code VSAM logic and parameters, refer to the:
- *VSE/VSAM User's Guide and Application Programming*, SC33-6732
- *VSE/VSAM Commands*, SC33-6731

- Javadoc for *VSAMRequestInfo.java* and *VSAMFileInfo.java*.

Calling a VSAM Request Handler

The VSAM Redirector Server is implemented in Java and is not delivered with source code. The VSAM Redirector Server can be started after the:

1. properties file has been configured,
2. request handler code has been copied into the directory where the VSAM Redirector Server is installed.

When the VSAM Redirector Server has been started:

1. The VSAM Redirector Server reads the properties file.
2. The VSAM Redirector Server listens for VSAM Redirector Clients on the TCP/IP port that you defined in "Step 2: Configure the Properties File" on page 61.
3. When the VSAM Redirector Server receives an OPEN request:
   a. The request handler that was defined in configuration.phase, is instantiated.
   b. The request handler's OPEN method is called.
4. The request handler terminates in the way that is defined by the *CLOSE method* call.

Error Reporting

Request handlers report an error by producing an *Exception*. The VSAM Redirector Server intercepts the Exception and sends the error code to the VSAM Redirector Client running on the VSE/ESA host. *No data is sent together with the error code*.

Error codes that are sent to the VSAM Redirector Client:

- must be an internal VSAM return code
- are returned to VSAM into register 15 (R15).

A new error code has been created for use with the VSAM Redirector connector: when a request handler sends a `DUPLICATE RECORD` warning to a VSAM application (when more than one record is found in alternate index access/path access and when other records follow) it produces an Exception. However, in this case the record data is not sent!. To return this warning message, you must therefore use error code 255 (which will be converted in the server-part to the required warning message).

Also refer to "VSAM Integration Considerations" on page 48 for further error-reporting considerations.

Datatype Conversions

Record data that is transferred from the VSAM Redirector Client running on the VSE/ESA host, consists of EBCDIC characters. Therefore, all data that is interpreted as Strings must be converted to *ASCII characters*.

An instance of the *CodepageTranslator* class is used for converting EBCDIC to ASCII. You can also write your own translators and configure (in the properties file) the VSAM Redirector Server to use these translators.

The *VSAMRequestInfo* class contains many methods that are implemented to get and set data areas in a record. You must specify the part (*offset* plus *length*) of the record and the method's return the data in this area interpreted as:

- String
- Number (Signed/Unsigned)
- Packed Decimal
- Binary bytes

The get/set String method always uses the CodepageTranslator.

Getting a Map Dynamically Into Your Request Handler

These are the ways in which you can get a map into your request handler *dynamically*:

- Define the map as an option string in the **config.phase** and parse it.
- Use the VSE/ESA Connector framework to get a VSAM map from the cluster.
- Use an XML file and then parse it (you can create such a file using the *MapTool* – see *CreateDB2Tables.java* for an example of how to do so).
- Store the map in another location (for example, in a database table – see *DB2Handler.java* for an example of how to do so).

IBM-Supplied Example of DB2-Related Handler

This example (*DB2Handler*) shows how you can redirect all VSAM requests for a specific file, to a remote DB2 database. To use this example, you must have installed on the remote system:

- DB2 database
- Java Development Kit

DB2Handler is represented by the Java class **com.ibm.vse.db2handler.DB2Handler**, whose option string delivers:

- username
- tablename
- password
- systemname

To run *DB2Handler* you must:

- Include the DB/2 JDBC 1.2 driver (**db2java.zip**) in your CLASSPATH variable.
- Obtain the *xerces* package, which is required by *CreateDB2Tables* in order to parse the XML file. You can download the *xerces* package from the Apache Foundation (address **xml.apache.org**).

DB2Handler starts various sub-handlers that handle ESDS/KSDS and RRDS/VRDS files. Each sub-handler then:

1. Opens a database table containing the field definitions.
2. Reads the database table.
3. Prepares the database table for record requests.
4. Analyzes each incoming request, and processes each request.
5. Sends the result back to the VSAM Redirector Client.

For further information, refer to the source code for each sub-handler, which contain detailed comments.

These are the restrictions for using *DB2Handler*:

- Share option 1 and 2 only are supported. Concurrent updates to the same DB2 table are not supported. However, your request handler can be coded to respond to database locks.
- STRING fields only are supported as KEY fields in KSDS files (for BASE Cluster and AIX).
- There is no support for NON-UNIQUE AIX access. You should avoid this type of access completely.

- Fields that contain 0x00 (`NULL` fields) are *not* supported as KEY fields (for BASE Cluster and AIX), because Key columns are defined as `NOT NULL`.

The *CreateDB2Tables* program (**com.ibm.vse.db2handler.create.CreateDB2Tables**) creates the field definition table on the DB/2 system. To use *CreateDB2Tables* you should:

1. Create an XML file containing the field definitions required to create your tables. A sample XML file is supplied with the DTD, which you can use as a template. You can define the field types STRING, UNSIGNED, SIGNED, PACKED and BINARY. In the database:
   - STRING is a CHAR
   - UNSIGNED, SIGNED and PACKED are INT
   - BINARY is mapped to a BLOB (binary large object).
2. Start program *CreateDB2Tables* and follow the instructions that this program provides.
3. Your tables (including field definitions and data table) will be created and, if required, indexes built.

IBM-Supplied Example of HTML-Related Handler

The IBM-supplied example of an HTML-related VSAM request handler is *htmlhandler.java*, which does not require any configuration actions. Furthermore, all parameters are set in the source code.

htmlhandler.java is an application that is automatically started by VSE/ESA in order to collect daily statistics about the status of flight bookings. During the night, a new cluster is created, into which a VSAM data cluster is "copied":

1. The empty cluster is redirected to a Java system.
2. The request handler on the Java system then extracts the data from the record, and enters a statistic in the HTML file.

The logic for the above steps is contained in the Java *request handler*. Therefore each morning, an updated HTML file is automatically placed in the intranet.

Code for HTML-Related Request Handler

You implement a VSAM Request Handler by implementing the Java interface *VSAMRequestHandler* (documented in the Javadoc of the VSAM Redirector Server installation). This Java interface describes methods to access the file system or database to which the VSAM requests are redirected.

The example in this section describes the implementation of the *HTMLHandler*, which is part of the VSAM Redirector Server installation. It is a sample VSAM Request Handler, which writes the received data into an HTML file.

The sample request handler processes only INSERT requests (only writes to the target HTML file). It cannot, for example, delete already-written data from the file. The data used with the *HTMLHandler* is taken from a sample VSAM cluster which is also used in the *FlightOrderingServlet* example (see "Example of How to Implement a Servlet" on page 246 for details).

You must perform an IDCAMS REPRO of the data from the VSAM cluster FLIGHT.ORDERING.ORDERS to a second VSAM file, which will be redirected to the HTML request handler on the Java platform where the VSAM Redirector Server is installed.

Data Layout Used With HTMLHandler

The data layout for use with the *HTMLHandler* sample is shown in Figure 22:

```
Offset Length Type     Meaning
--------------------------------
0      20     STRING   first name
20     20     STRING   last name
40     4      UNSIGNED flight number
44     4      UNSIGNED seats
48     1      BINARY   smoking/non-smoking (non=0)
```

Figure 22. Data Layout Used With the HTMLHandler

The data consists of a mapped VSAM data. Therefore, the request handler can use the *VSEVsamField* offsets within the source cluster. An external *VSEVsamMap* definition is not used.

Note: Normally, you would have to provide a data mapping for your VSAM record layout, as described in Chapter 15, “Mapping VSE/VSAM Data to a Relational Structure”, on page 129.

Implementation of HTMLHandler

The request handler *HTMLHandler* implements a set of callback routines. These callback routines are called by the VSAM Redirector Server each time access to the target file or database system is performed.

```
package com.ibm.vse.htmlhandler;
import java.io.*;
import com.ibm.vse.redirector.*;
import com.ibm.vse.server.*;

public class HtmlHandler implements VSAMRequestHandler
{
   CodepageTranslator trnsl = null;
   VSAMFileInfo finfo = null;
   BufferedWriter htmloutput = null;
   int record_counter = 0;
   String htmlfilename = "output.html";
```

Figure 23. Implementation of the HTMLHandler

Initialize Method: The method shown in Figure 24 is called to initialize the *HTMLHandler* request handler. The *parameter translator* points to the codepage translator to be used for ASCII to EBCDIC translation.

```
public void initialize(CodepageTranslator translator)
{
   this.trnsl = translator;
}
```

Figure 24. Initialize Method of the HTMLHandler

Cleanup Method: The method shown in Figure 25 is called to cleanup the *HTMLHandler* request handler.

```
public void cleanup()
{
   trnsl = null;
}
```

Figure 25. Cleanup Method of the HTMLHandler

Open Method: The method shown in Figure 26 is called to open the file. The *fileInfo* parameter contains information about the file to open, and *options* parameter contains the options string.

```
public void open(VSAMFileInfo fileInfo,String options) throws VSAMRequestException
{
   this.finfo = fileInfo;
   try {
      // create the HTML file
      htmloutput = new BufferedWriter( new FileWriter( this.htmlfilename ) );
      // write the first part of HTML file
      htmloutput.write("<html><head><title>Redirector sample</title></head>
                       </body>");
      htmloutput.write("<h2>Output from redirected VSAM FLIGHT.ORDERING.ORDERS
                        cluster:</h2>");
      htmloutput.write("<table><tr><th>Flight Number</th><th>First name</th>
                       <th>Last name</th><th>Seats</th><th>Smoker?</th></tr>");
   } catch(Exception ex) { System.out.println("Error creating output file!"+ex); }

   System.out.println("Now receiving records:");
}
```

Figure 26. Open Method of the HTMLHandler

Close Method: The method shown in Figure 27 is called to close the file.

```
public void close() throws VSAMRequestException
{
   System.out.println("\nReady, "+this.record_counter+" records received.");
   try {
      // write the end of the HTML file
      htmloutput.write("</table></body></html>");
      // close the HTML file
      htmloutput.close();
   }
   catch (Exception ex) { }
   System.out.println("HTML file created: '" + this.htmlfilename + "'");
}
```

Figure 27. Close Method of the HTMLHandler

Request Method: The method shown in Figure 28 on page 68 is called to execute a VSAM request. The *requestInfo* parameter contains information about the request to execute. This example only implements the INSERT request. However, you would use this method to perform other actions on the data. You could even send an e-mail containing this data!.

```
public void request(VSAMRequestInfo requestInfo)
throws VSAMRequestException
{
   String flightnumber = null;
   int max_seats = 0;
   int seats_booked = 0;
   String sout = null;
   if ( requestInfo.isINSERT() )
   {
      firstname = requestInfo.getString(0, 20);
      lastname  = requestInfo.getString(20, 20);
      flightnumber = requestInfo.getNumber(40, 4);
      seats = requestInfo.getNumber(44, 4);
      smoktmp = requestInfo.getBinary(48, 1);
      if(smoktmp.length > 0)
           if( smoktmp[0] == 0 )
              smoker = false;
           else
              smoker = true;
       // now output data to html file
      sout = "<tr><td>"+flightnumber+"</td><td>"+firstname+"</td>
              <td>"+lastname+"</td><td>"+seats+"</td><td>"+smoker+"</td></tr>";
      try {
        htmloutput.write(sout);
        htmloutput.newLine();
      }
      catch (Exception ex)
      {
        System.out.println(
          "Exception while writing another line into the HTML file:"+ex);
      }
   }
   System.out.print("."); // print out a progress indicator
   this.record_counter++;
}
```

Figure 28. Request Method of the HTMLHandler

Finished Method: The method shown in Figure 29 is called to inform that a request that has been finished.

```
public void finished(byte success) throws VSAMRequestException
  {
     return;
  }
}
```

Figure 29. Finished Method of the HTMLHandler

Screenshots of HTMLHandler Console and Output

Figure 30 on page 69 shows the console listing produced from running the *HTMLHandler* request handler.

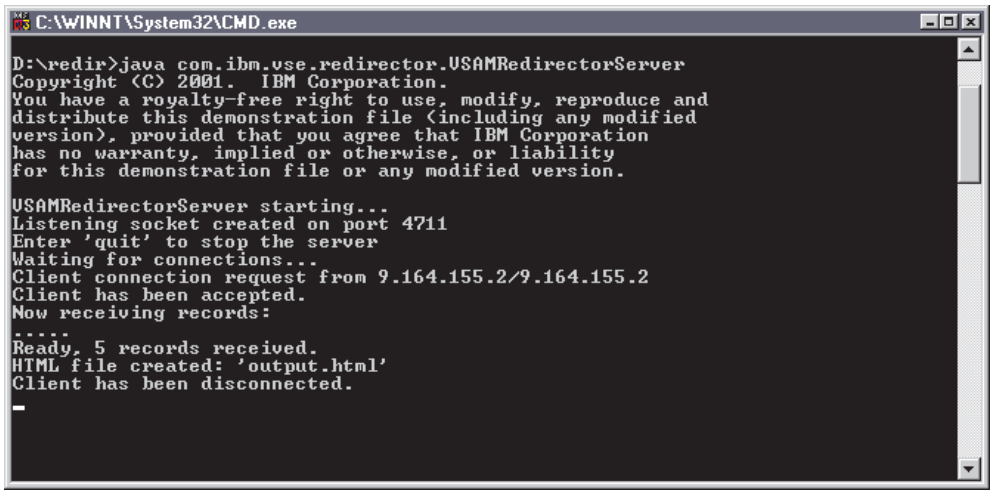


Figure 30. Console Listing from Running the HTMLHandler Request Handler

Figure 31 shows the output produced by running the *HTMLHandler* request handler.

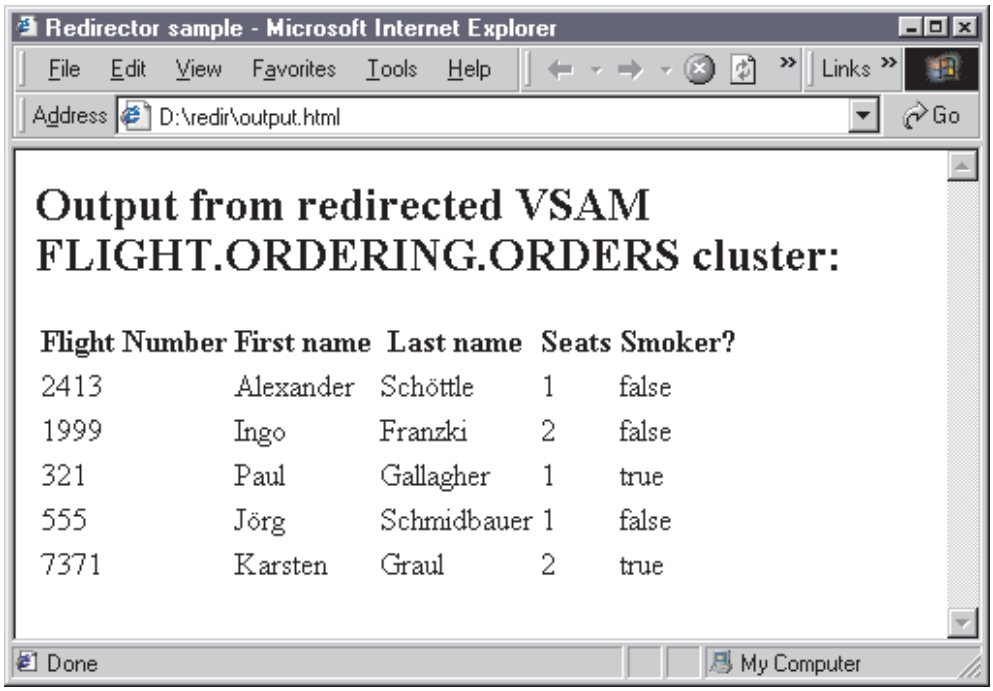


Figure 31. Output from Running the HTMLHandler Request Handler

Chapter 9. Customizing the DB2-Based Connector

This chapter describes the *customization* activities required before the DB2-based connector can be used, and contains these main sections:

- "Host Installation Activities That Must Be Already Completed"
- "Step 1: Customize CICS TS" on page 72
- "Step 2: Customize TCP/IP" on page 72
- "Step 3: Customize DB2 and Define Sample Database" on page 72
- "Step 4: Set Up for DRDA Support" on page 80
- "Step 5: Set Up Stored Procedure Server and Define to DB2" on page 81
- "Step 6: Set Up for Stored Procedures" on page 82
- "Step 7: Customize the DB2-Based Connector for VSAM Data Access" on page 83
- "Step 8: Customize the DB2-Based Connector for DL/I Data Access" on page 83
- "Step 9: Start DB2, and Start Stored Procedure Server" on page 84
- "Step 10: Install DB2 Connect and Establish Client-Host Connection" on page 85

DB2 Version!
The jobs and examples provided in this chapter use DB2 Server for VSE Version 7.2 . If you are using a later release of DB2, the jobs and samples might have a slightly different layout.

Host Installation Activities That Must Be Already Completed

This section provides a summary of the installation activities that must already be completed on the VSE/ESA host, before you begin to customize the DB2-based connector (for a more detailed description of these activities, refer to the chapter "VSE/ESA e-business Connectors" in the *VSE/ESA Planning*):

- The DB2 Server for VSE must be restored from the VSE/ESA Extended Base Tape to sub-library PRD2.DB2720. The startup job for the DB2 Server for VSE is defined for dynamic partition of class S. It is either installed:
  - During the initial installation of VSE/ESA 2.7
  - Following an FSU (Fast Service Upgrade).
- The AIBTDLI interface must be installed for accessing DL/I data via DB2 Stored Procedures. To use the AIBTDLI interface:
  - DL/I VSE 1.11 or later must be installed.
  - APAR PQ39683 for DL/I must be applied.
  - Your CICS/DLI system must have all databases (DBDs) that you wish to use defined in the CICS FCT, together with the AIBTDLI interface.
  - Your CICS/DLI system must have:
    - all PSBs defined in the DL/I online nucleus DLZNUCxx
    - an active MPS system.
  - The DL/I task termination exit *DLZBSEOT* (described in "Task Termination and Abend Handling" on page 329) must be resident in the SVA.

© Copyright IBM Corp. 2000, 2003

- One or more CICS TS systems must be customized for use with the DB2 Server for VSE.

Step 1: Customize CICS TS

If you have more than one CICS TS running, you must first decide which (one or more) of your CICS systems is to have access to the *DB2 Server for VSE*. To customize a selected CICS TS you must:

- Have journaling *active*. Use the corresponding DFHSITxx skeleton to set JCT=**SP** (for DFHJCT**SP**) or another suffix.
- Compile the JCT using the modified skeleton DFHJCTSP.
- Define the journal files. For CICSICCF use the provided skeleton SKJOURN. For PRODCICS use the provided skeleton SKJOUR2. Both are available in VSE/ICCF library 59.

Step 2: Customize TCP/IP

Ensure that your TCP/IP startup job has the following two values set:

```
SET WINDOW              = 8192
SET MAX_SEGMENT         = 700
```

For editing, use the TCP/IP dialog on your client workstation, or modify the appropriate startup member in your VSE library.

For details, refer to the *TCP/IP for VSE/ESA IBM Program Setup and Supplementary Information* manual.

Step 3: Customize DB2 and Define Sample Database

Step 3 involves running **SKDB2VAR** (*in partition BG*), which is the main skeleton you use for customizing the DB2-based connector. The jobs included in this skeleton are now described.

Notes:

1. Before starting this step, you must have activated the license key for using DB2. If not, refer to the *VSE/ESA Planning* manual for details of how to activate this key using the skeleton SKUSERBG (contained in ICCF Library 59).
2. Skeleton SKDB2VAR assumes you are using an IBM 3380 disk device for storage allocations. If you use a different disk device type, you must change these allocations accordingly (especially in the case of FBA devices).

These are the jobs that skeleton SKDB2VAR runs:

- "Step 3.1: Define User Catalog" on page 73
- "Step 3.2: Catalog New ARISIVAR.Z" on page 73
- "Step 3.3: Job Manager for Preparation / Installation Steps" on page 74
- "Step 3.4: Activate DRDA Server Support" on page 75
- "Step 3.5: Startup Job for Stored Procedure Server" on page 75
- "Step 3.6: Prepare DB2 Sample Database" on page 75
- "Step 3.7: Install DB2 Sample Database" on page 77

Step 3.1: Define User Catalog

The job **DB2DEFCT** defines a User Catalog and space on a separate volume.

For the DB2UCAT catalog, the allocated space consists of 150 cylinders.

A standard label is inserted for the new catalog, with the name **DB2UCAT**. You must enter your own values for these variable:

- -V001- -
The Id of your volume. This VOLID is also used in job DB2CTVAR.

- -V002- -
The number of tracks allocated for the catalog space. The recommended allocation is 150 cylinders.

```
$$ JOB JNM=DB2DEFCT,CLASS=0,DISP=D,NTFY=YES
$$ LST CLASS=Q,DISP=H
// JOB DB2DEFCT DEFINE USER CATALOG DB2UCAT
* THIS JOB WILL TERMINATE IN CASE THE DB2UCAT IS ALREADY DEFINED.
// EXEC IDCAMS,SIZE=AUTO
  LISTCAT ALL CATALOG(DB2.USER.CATALOG)
  IF LASTCC EQ 8 THEN DO
      SET LASTCC = 0
      SET MAXCC = 0
   DEFINE USERCATALOG(NAME(DB2.USER.CATALOG) -
      VOL(--V001--)                          -
      NOTRECOVERABLE                         -
      TRK(15))
   DEFINE SPACE(VOLUMES(--V001--)            -
      CYL(--V002--))                         -
      CATALOG(DB2.USER.CATALOG)
    END
   ELSE DO
      SET LASTCC = 0
      SET MAXCC = 0
   END
/*
// OPTION STDLABEL=DELETE
DB2UCAT
/*
// OPTION STDLABEL=ADD
// DLBL DB2UCAT,'DB2.USER.CATALOG',0,VSAM
/*
// EXEC IESVCLUP,SIZE=AUTO             ADD LABEL TO STDLABUP PROC
D                                             DB2UCAT
D DB2.SQLGLOB.MASTER                          SQLGLOB
A DB2.USER.CATALOG                            DB2UCAT
A DB2.SQLGLOB.MASTER                          SQLGLOB DB2UCAT OLD KEEP
/*
/&
$$ EOJ
```

Step 3.2: Catalog New ARISIVAR.Z

In this step, Job DB2CTVAR first renames the original DB2-supplied ARISIVAR.Z to ARISIVAR.ORIG, and catalogs the global variable member ARISIVAR.Z. Then the new ARISIVAR.Z is used to test the installation of the DB2-based connector and the sample database.

Note that the processing of ARISIVAR.Z is controlled by the **DB2 Job Manager**.

You start the Job Manager by releasing the job **DB2JMGR** in the VSE/POWER reader queue (placed there by skeleton SKDB2VAR). You must release the Job Manager once for each step: Preparation, Installation (or Migration).

Customizing the DB2-Based Connector

As described in detail in the *Program Directory* for the *DB2 Server for VSE*, ARISIVAR.Z processes many parameters, globals and variables, to define DB2 characteristics and resources.

Major definitions include, for example:

- The DB2 sample database **SQLDS** which is defined on volume **- -V001- -** (variable in skeleton SKDB2VAR).
- The **DB2 Server for VSE Help** component, the installation of which is controlled by the following variable:

  ```
  ARIS72JZ HELP       YES
  ```

  You are recommended to install the DB2 Server for VSE Help (you must replace **- -V003- -** with the address of your tape drive). When this variable is processed, you are requested to mount the corresponding tape (containing the DB2 Help files). This is the fourth tape (the extra tape) of the Base distribution tapes you received.
- The creation of DB2 (work) files such as **BINDFILE, BINDWFILE**, and **SQLGLOB**.
- The setting of **CICS TS** parameters as required for a DB2 environment.

Here are the contents of DB2CTVAR:

```
$$ JOB JNM=DB2CTVAR,CLASS=0,DISP=D,NTFY=YES
$$ LST CLASS=Q,DISP=H
// JOB DB2CTVAR CATALOG GLOBAL VARIABLE MEMBER FOR DB2
* ORIGINAL ARISIVAR.Z RENAMED TO ARISIVAR.ORIG
// EXEC  LIBR,PARM='MSHP'
    ACCESS  SUBLIB = PRD2.DB2720
    RENAME           ARISIVAR.Z:=.ORIG
CATALOG ARISIVAR.Z        EOD=&&          REPLACE=YES
⋮


 (for further details, refer to the sample SKDB2VAR in Library 59
```

Step 3.3: Job Manager for Preparation / Installation Steps

```
$$ JOB JNM=DB2JMGR,CLASS=R,DISP=L,NTFY=YES
$$ LST CLASS=Q,DISP=H
// JOB DB2JMGR DB2 JOB MANAGER
// LIBDEF *,SEARCH=(PRD2.DB2720)
// EXEC REXX=ARISIMGR
/*
/&
$$ EOJ
```

Job DB2JMGR is used in “Step 3.6: Prepare DB2 Sample Database” on page 75 and “Step 3.7: Install DB2 Sample Database” on page 77.

Step 3.4: Activate DRDA Server Support

This job will be later used in Step 4.

```
$$ JOB JNM=DB2DRDA,CLASS=R,DISP=L,NTFY=YES
$$ LST CLASS=Q,DISP=H
// JOB DB2DRDA ACTIVATE DRDA SERVER SUPPORT
* ********************************************************
*  LINK EDIT RDS WITH DRDA SERVER SUPPORT
* ********************************************************
// LIBDEF *,SEARCH=PRD2.DB2720
// LIBDEF PHASE,CATALOG=PRD2.DB2720
// OPTION CATAL
 INCLUDE ARISLKRA
// EXEC  PGM=LNKEDT,PARM='MSHP,AMODE=31,RMODE=ANY'
/*
/&
$$ EOJ
```

Step 3.5: Startup Job for Stored Procedure Server

In this step, skeleton SKDB2VAR loads the job to start the Stored Procedure Server, into the VSE/POWER reader queue.

```
$$ JOB JNM=PSERVER,CLASS=0,DISP=L
$$ LST CLASS=W,DISP=H
// JOB PSERVER
// LIBDEF PROC,SEARCH=(PRD2.DB2720)
// DLBL SQLGLOB,'DB2.SQLGLOB.MASTER',,VSAM,CAT=DB2UCAT,DISP=(OLD,KEEP)
// EXEC PROC=ARIS72PL        *-- DB2 PROD. LIBRARY ID PROC
// EXEC PROC=ARIS72DB        *-- DB2 DATABASE ID PROC
// EXEC  ARIDBS,SIZE=AUTO
CONNECT SQLDBA IDENTIFIED BY SQLDBAPW;
CREATE PSERVER SPSERV01 AUTOSTART YES;
/*
/&
$$ EOJ
```

Step 3.6: Prepare DB2 Sample Database

This job uses the preparation-related global definitions contained within ARISIVAR.Z (catalogued in Step 3.2). To run the preparation step, DB2 Job Manager (DB2JMGR) must be released. Refer to the console listing below for details of how to run this step.

Note: You must enter a **0** (partition **BG**) to run the *preparation* job – in the sample below, the DB2 Job Manager runs in partition F4, and DB2 Server for VSE Version 7.2 is used.

```
r rdr,db2jmgr
AR 0015 1C39I COMMAND PASSED TO VSE/POWER
F1 0001 1R88I  OK
F4 0001 1Q47I   F4 DB2JMGR 00249 FROM (HEHA) , TIME= 9:28:39
F4 0004 // JOB DB2JMGR DB2 JOB MANAGER
        DATE 07/07/2001, CLOCK 09/28/39
F4 0004 ************************************************************
F4 0004       PREPARE FOR INSTALLATION/MIGRATION PROCESS
F4 0004 ************************************************************
F4 0004 ENTER INSTALLATION LIBRARY NAME (PRD2.DB2720   default)
F4-0004
4
F4 0004 YOU HAVE SELECTED  PRD2.DB2720
F4 0004 PRESS ENTER TO CONTINUE OR ENTER ANY OTHER KEY TO
F4 0004 MODIFY YOUR SELECTION:
F4-0004
4
F4 0004 WHICH CLASS WILL YOU USE TO RUN THE PROCESS ?  (4  default)
```

```
F4-0004
4 0
F4 0004 YOU HAVE SELECTED  0
F4 0004 PRESS ENTER TO CONTINUE OR  ENTER ANY OTHER KEY TO
F4 0004 MODIFY YOUR SELECTION:
F4-0004
4
F4 0004 PLEASE SELECT ONE OF THE FOLLOWING :
F4 0004 FOR PREPARATION....  ENTER (P)
F4 0004 FOR INSTALLATION...  ENTER (I)
F4 0004 FOR MIGRATION......  ENTER (M)
F4-0004
F4-0004
4 p
F4 0004 IF PREPARATION FOR:
F4 0004 INSTALLATION... PLEASE ENTER (I)
F4 0004 MIGRATION...... PLEASE ENTER (M)
F4-0004
4 i
F4 0004 DO YOU WANT TO EXECUTE ALL JOBS? {Y|N-default}
F4-0004
4 y
F4 0004
F4 0004 ************ JOB ARIS72JD ********************
F4 0004 *   DEFINE DB2720 PROGRAMS AND TRANSACTIONS
F4 0004 *********************************************
F4 0004 JOB ARIS72JD IS OPTIONAL.
F4 0004 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F4-0004
4 y
BG 0001 1Q47I   BG ARIS72JD 00281 FROM (HEHA) , TIME=11:12:37
BG 0000 * ** JOB JNM=ARIS72JD,CLASS=0,DISP=D
BG 0000 * ** LST CLASS=V,DISP=D,DEST=(,XXXXXXX)
:
F4 0004 JOB ARIS729D IS OPTIONAL.
F4 0004 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F4-0004
4 y
BG 0001 1Q47I   BG ARIS729D 00285 FROM (HEHA) , TIME=11:13:10
BG 0000 // JOB ARIS729D
BG 0000 * ************************************************************
BG 0000 * ARIS729D: DEFINE VSAM CLUSTER FOR THE BINDWKF FILE
BG 0000 * ************************************************************
BG 0000 EOJ ARIS729D  MAX.RETURN CODE=0000
BG 0000 EOJ NO NAME
BG 0001 1Q34I   BG WAITING FOR WORK
F4 0004 *********************************************
F4 0004 * Job  ARIS729D  executed successfully
F4 0004 *********************************************
F4 0004
F4 0004 ************ JOB ARISIQBD ********************
F4 0004 *   ISQL BIND FILE CONVERSION
F4 0004 *********************************************
F4 0004 JOB ARISIQBD IS OPTIONAL.
F4 0004 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F4-0004
4 y
BG 0001 1Q47I   BG ARISIQBD 00286 FROM (HEHA) , TIME=11:13:20
BG 0000 // JOB ARISIQBD -- ISQL BIND FILE CONVERSION
BG-0000 // PAUSE
0
BG 0000 EOJ ARISIQBD  MAX.RETURN CODE=0000
BG 0000 EOJ NO NAME
BG 0001 1Q34I   BG WAITING FOR WORK
F4 0004 *********************************************
F4 0004 * Job  ARISIQBD  executed successfully
F4 0004 *********************************************
```

```
F4 0004
F4 0004 ************ JOB ARIS72CD ********************
F4 0004 *   DB2 SERVER STARTER DATABASE VSAM DEFINITIONS
F4 0004 ******************************************
BG 0001 1Q47I   BG ARIS72CD 00294 FROM (HEHA) , TIME=11:28:08
BG 0000 // JOB ARIS72CD              DB2 FOR VSE STARTER DB VSAM DEFINITIONS
BG 0000 EOJ ARIS72CD  MAX.RETURN CODE=0
F4 0004
F4 0004 ************ JOB ARISSTDL ********************
F4 0004 *   ADD NEW LABELS TO STANDARD LABELS
F4 0004 ******************************************
F4 0004 JOB ARISSTDL IS OPTIONAL.
F4 0004 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F4-0004
4 y
BG 0001 1Q47I   BG ARISSTDL 00297 FROM (HEHA) , TIME=11:30:06
BG 0000 // JOB ARISSTDL
BG 0000 EOJ ARISSTDL
BG 0000 EOJ NO NAME
BG 0001 1Q34I   BG WAITING FOR WORK
F4 0004 READ CONSOLE FOR DETAILS.
F4 0004 EOJ DB2JMGR   MAX.RETURN CODE=0000
```

Step 3.7: Install DB2 Sample Database

This job uses the installation-related global definitions contained within ARISIVAR.Z (catalogued in Step 3.2). To run the installation step, DB2 Job Manager (DB2JMGR) must be released. Refer to the console listing below for details of how to run this step.

Note: You must run this installation step in a *static* partition (the example below uses static partition **F4**) – in the sample below, the DB2 Job Manager runs in partition F7.

```
r rdr,db2jmgr
AR 0015 1C39I COMMAND PASSED TO VSE/POWER
F1 0001 1R88I  OK
F7 0007 // JOB DB2JMGR DB2 JOB MANAGER
F7 0007 ************************************************************
F7 0007       PREPARE FOR INSTALLATION/MIGRATION PROCESS
F7 0007 ************************************************************
F7 0007 ENTER INSTALLATION LIBRARY NAME (PRD2.DB2720   default)
F7-0007
7
F7 0007 YOU HAVE SELECTED  PRD2.DB2720
F7 0007 PRESS ENTER TO CONTINUE OR ENTER ANY OTHER KEY TO
F7 0007 MODIFY YOUR SELECTION:
F7-0007
7
F7 0007 WHICH CLASS WILL YOU USE TO RUN THE PROCESS ?  (4  default)
F7-0007
7 4
F7 0007 YOU HAVE SELECTED  4
F7 0007 PRESS ENTER TO CONTINUE OR  ENTER ANY OTHER KEY TO
F7 0007 MODIFY YOUR SELECTION:
F7-0007
7
F7 0007 PLEASE SELECT ONE OF THE FOLLOWING :
F7 0007 FOR PREPARATION....  ENTER (P)
F7 0007 FOR INSTALLATION...  ENTER (I)
F7 0007 FOR MIGRATION......  ENTER (M)
F7-0007
7 i
F7 0007 DO YOU WANT TO EXECUTE ALL JOBS? {Y|N-default}
F7-0007
7 y
```

```
F7 0007
F7 0007 ************ JOB ARISBDID ********************
F7 0007 * SETUP THE DBNAME DIRECTORY
F7 0007 *****************************************
F7 0007 JOB ARISBDID IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 y
F4 0001 1Q47I   F4 ARISBDID 00300 FROM (HEHA) , TIME=11:36:54
F4 0004 // JOB ARISBDID -- DBNAME DIRECTORY SERVICE GENERATION
F4 0004 * ************************************************************************
F4 0004 *                                                                        *
F4 0004 *  THIS JCL EXECUTES STEPS TO GENERATE THE DBNAME DIRECTORY SERVICE *
F4 0004 *  ROUTINE (ARICDIRD.PHASE).                                             *
F4 0004 *                                                                        *
F4 0004 *  STEP 1 EXECUTES THE PROCEDURE ARICCDID FOR MIGRATION                  *
F4 0004 *  OR ARICBDID FOR INSTALLATION TO READ THE DBNAME                       *
F4 0004 *  DIRECTORY SOURCE MEMBER ARISDIRD.A FROM THE PRODUCTION LIBRARY,  *
F4 0004 *  AND GENERATES THE ASSEMBLER VERSION OF ARISDIRD ON SYSPCH.       *
F4 0004 *                                                                        *
F4 0004 * ************************************************************************
F4 0004 * ************************************************************************
F4 0004 * ARISBDID  STEP 1 -- BUILD ASSEMBLER VERSION OF ARISDIRD           *
F4 0004 * ************************************************************************
F4 0004 * SQL/DS DBNAME DIRECTORY BUILT SUCCESSFULLY
F4 0004 EOJ ARISBDID  MAX.RETURN CODE=0000
F7 0007 *****************************************
F7 0007 * Job  ARISBDID  executed successfully
F7 0007 *****************************************
F7 0007
F7 0007 ************ JOB ARIS72BD ********************
F7 0007 * LINK EDIT DB2 SERVER ONLINE SUPPORT COMPONENTS
F7 0007 *****************************************
F7 0007 JOB ARIS72BD IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 y
F4 0001 1Q47I   F4 ARIS72BD 00301 FROM (HEHA) , TIME=11:37:07
F4 0004 // JOB ARIS72BD     LINK EDIT DB2 FOR VSE ONLINE SUPPORT COMPONENTS
F4 0004 * ************************************************************
F4 0004 * ARIS090D: LINK EDIT SQL/DS ONLINE RESOURCE ADAPTER CONTROL
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARIS140D: LINK EDIT ISQL
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARIS150D: LINK EDIT ISQL ITRM TERMINAL TRANSACTION
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARIS160D: LINK EDIT ISQL ITRM TERMINAL EXTENSION PROGRAM
F4 0004 * ************************************************************
F4 0004 EOJ ARIS72BD  MAX.RETURN CODE=0000
F4 0004 EOJ NO NAME
F4 0001 1Q34I   F4 WAITING FOR WORK
F7 0007 *****************************************
F7 0007 * Job  ARIS72BD  executed successfully
F7 0007 *****************************************
F7 0007
F7 0007 ************ JOB ARIS72DD ********************
F7 0007 * DATABASE DBGEN AND SET UP
F7 0007 *****************************************
F4 0001 1Q47I   F4 ARIS72DD 00302 FROM (HEHA) , TIME=11:37:20
F4 0004 // JOB ARIS72DD               DATABASE DBGEN AND SET UP
F4 0004 * ************************************************************
F4 0004 * ARIS72SL: DB2 SERVICE/PRODUCTION LIBRARY DEFINITION
F4 0004 * ************************************************************
F4 0004 * ************************************************************
```

```
F4 0004 * ARIS72DB: DB2 STARTER DATABASE IDENTIFICATION
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARIS030D: GENERATE THE STARTER DATABASE
F4 0004 * ************************************************************
F4 0004 ARI0025I The program ARISQLDS is loaded at 400078.
F4 0004 ARI0025I The program ARICMOD is loaded at 564D80.
F4 0004 ARI0025I The program ARIXSXR is loaded at 581C00.
F4-0004 ARI0919D Database generation invoked.
                 The database will be formatted and the original
                 database destroyed.

                 Enter either:
                   DBGEN   to continue, or
                   CANCEL  to cancel.
4 dbgen
F4 0004 System identification at DB generation = DB2 VSE & VM 7.2
F4 0004
⋮
F7 0007 *********************************************
F7 0007 * Job  ARIS72DD  executed successfully
F7 0007 *********************************************
F7 0007
F7 0007 ************ JOB ARIS72ED ********************
F7 0007 * INSTALL DATABASE COMPONENTS (ISQL, FIPS FLAGGER)
F7 0007 *********************************************
F7 0007 JOB ARIS72ED IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 y
F4 0001 1Q47I   F4 ARIS72ED 00303 FROM (HEHA) , TIME=11:45:41
F4 0004 // JOB ARIS72ED     INSTALL DATABASE COMPONENTS
F4 0004 * ************************************************************
F4 0004 * ARIS72SL: DB2 SERVICE/PRODUCTION LIBRARY DEFINITION
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARIS72DB: DB2 STARTER DATABASE IDENTIFICATION
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARIS080D: GRANT SCHEDULE AUTHORITY TO DBDCCICS
F4 0004 * ************************************************************
⋮
F4 0001 1Q34I   F4 WAITING FOR WORK
F7 0007 *********************************************
F7 0007 * Job  ARIS72WD  executed successfully
F7 0007 *********************************************
F7 0007 ************ JOB ARIS72HZ ********************
F7 0007 * ENLARGE HELP TEXT DBSPACE
F7 0007 *********************************************
F7 0007 JOB ARIS72HZ IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 n
F7 0007
F7 0007 ************ JOB ARIS72JZ ********************
F7 0007 * INSTALL LANGUAGE
F7 0007 *********************************************
F7 0007 JOB ARIS72JZ IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 n
F7 0007
F7 0007 ************ JOB ARIS72FD ********************
F7 0007 * GRANT SCHEDULE AUTHORITY
F7 0007 *********************************************
F7 0007 JOB ARIS72FD IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
```

```
F7-0007
7 y
F4 0001 1Q47I   F4 ARIS72FD 00305 FROM (HEHA) , TIME=11:47:09
F4 0004 // JOB ARIS72FD     GRANT SCHEDULE FOR DFHSIT APPLID
F4 0004 * ************************************************************
F4 0004 * ARIS72PL: DB2 PRODUCTION LIBRARY DEFINITION
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARIS72DB: DB2 STARTER DATABASE IDENTIFICATION
F4 0004 * ************************************************************
F4 0004 * ************************************************************
F4 0004 * ARISDBSD: EXECUTE THE DBS UTILITY IN SQL/DS SINGLE USER MODE
F4 0004 * ************************************************************
:
F7 0007
F7 0007 ************ JOB ARIS6ASD ********************
F7 0007 * SQL ASSEMBLER SAMPLE PROGRAM
F7 0007 *********************************************
F7 0007 JOB ARIS6ASD IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 y
F7 0001 1Q47I   F7 ARIS6ASD 00318 FROM (HEHA) , TIME=12:33:17
F7 0007 // JOB ARIS6ASD SQL ASSEMBLER SAMPLE PROGRAM
F7 0007 * STEP 1 - PREP
F7 0007 1T20I  SYS079 HAS BEEN ASSIGNED TO X'FED' (PERM)
F7 0007 1T20I  SYSPCH HAS BEEN ASSIGNED TO X'FED' (PERM)
F7 0007 * STEP 2 - ASSEMBLE
F7 0007 1T20I  SYSIPT HAS BEEN ASSIGNED TO X'FEC' (PERM)
F7 0007 * STEP 3 - LINK EDIT
F7 0007 * STEP 4 - EXECUTE THE SAMPLE PROGRAM
F7 0007 EOJ ARIS6ASD  MAX.RETURN CODE=0000
F7 0007 ************ JOB ARIS6CD ********************
F7 0007 * SQL C/370 SAMPLE PROGRAM
F7 0007 *********************************************
F7 0007 JOB ARIS6CD IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 n
F7 0007
F7 0007 ************ JOB ARIS6FTD ********************
F7 0007 * SQL FORTRAN SAMPLE PROGRAM
F7 0007 *********************************************
F7 0007 ************ JOB ARIS6PLD ********************
F7 0007 * SQL PL/I SAMPLE PROGRAM
F7 0007 *********************************************
F7 0007 JOB ARIS6PLD IS OPTIONAL.
F7 0007 WOULD YOU LIKE TO RUN IT? {Y|N-Default}
F7-0007
7 n
F7 0007 INSTALLATION STEP HAS ENDED SUCCESSFULLY....
F7 0007 PLEASE READ THE PROGRAM DIRECTORY AND PERFORM NECESSARY
F7 0007 STEPS MANUALLY.
F7 0007 EOJ DB2JMGR   MAX.RETURN CODE=0000
```

Step 4: Set Up for DRDA Support

This step, together with Steps 5 and 6, prepares your VSE/ESA environment for use with the DB2-based connector.

To create the definitions required for the DRDA setup, release job **DB2DRDA** placed by skeleton SKDB2VAR into the VSE/POWER reader queue.

```
* $$ JOB JNM=DB2DRDA,CLASS=R,DISP=L,NTFY=YES
* $$ LST CLASS=Q,DISP=H
// JOB DB2DRDA ACTIVATE DRDA SERVER SUPPORT
```

```
* ********************************************************
*  LINK EDIT RDS WITH DRDA SERVER SUPPORT
* ********************************************************
// LIBDEF *,SEARCH=PRD2.DB2720
// LIBDEF PHASE,CATALOG=PRD2.DB2720
// OPTION CATAL
 INCLUDE ARISLKRA
// EXEC  PGM=LNKEDT,PARM='MSHP,AMODE=31,RMODE=ANY'
/*
/&
* $$ EOJ
```

Step 5: Set Up Stored Procedure Server and Define to DB2

Step 5 is divided into:

- "Step 5.1: Set Up the Stored Procedure Server"
- "Step 5.2: Define Stored Procedure Server to DB2" on page 82

Step 5.1: Set Up the Stored Procedure Server

The *Stored Procedure Server* runs per default in a dynamic partition of class R with a partition size of 8MB. You can, however, configure the job to run in another partition that has a larger partition size (8MB is the minimum recommended).

Use skeleton **SKDB2SPS** (located in VSE/ICCF library 59) to catalog the startup job **SPSERV01** for the *Stored Procedure Server* into PRD2.DB2720. This job is shown below.

```
$$ JOB JNM=DB2SPSCA,CLASS=0,DISP=D,NTFY=YES
$$ LST CLASS=Q,DISP=H
// JOB DB2SPSCA CATALOG STORED PROCEDURE SERVER JOB
* ********************************************************************  C
* *                                                                  *  C
* *  THIS JOB WILL CATALOG THE JOB THAT WILL RUN IN THE              *  C
* *  STORED PROCEDURE SERVER PARTITION.                              *  C
* *  SERVER PARTITION SHOULD BE A DYNAMIC PARTITION OF AT LEAST      *  C
* *  8MB IN SIZE, DEFAULT IS CLASS R.                                *  C
* *  MODIFY JCL AS FOR YOUR NEEDS. CLASS IN THE POWER PUN CARD       *  C
* *  IS THE PARTITION WHERE THE SERVER RUNS.                         *  C
* *  THE STORED PROCEDURE LIBRARY PRD2.DB2STP SHOULD BE IN THE       *  C
* *  SEARCH CHAIN.                                                   *  C
* ********************************************************************  C
                                                                        C
         AFTER YOU HAVE MODIFIED THE SKELETON ENTER '§DTRSEXIT'         C
         FROM THE EDITOR'S COMMAND LINE.                                C
         THIS MACRO WILL DELETE ALL DESCRIPTIVE TEXT FROM THIS FILE,    C
         BY DELETING ALL LINES WHICH ARE MARKED WITH THE CHARACTER C    C
         IN COLUMN 71.                                                  C
                                                                        C
// EXEC  LIBR,PARM='MSHP'
   ACCESS  SUBLIB = PRD2.DB2720
CATALOG SPSERV01.A        EOD=&&          REPLACE=YES
. $$ PUN JNM=SPSERV01,DISP=I,CLASS=R
// JOB SPSERV01    START DB2 STORED PROCEDURE SERVER 01
// OPTION NODUMP,NOSYSDUMP
* // EXEC PROC=ARIS72SL
// ASSGN SYS098,SYSPCH
// LIBDEF *,SEARCH=(PRD2.DB2STP,PRD2.DB2720,PRD2.SCEEBASE,PRD1.BASE)
   ON $RC > 0 GOTO END
// EXEC  PGM=ARISPRC,SIZE=1M
/.END
/*
/&
```

```
&&
/*
/&
$$ EOJ
```

A *Stored Procedure Server* is always dedicated to a particular *DB2 Server for VSE* which starts an associated *Stored Procedure Server* during system startup.

Step 5.2: Define Stored Procedure Server to DB2

You can define the Stored Procedure Server SPSERV01 using:

- This DB2 command:

```
CREATE PSERVER SPSERV01 AUTOSTART YES
```

- This batch job, which you can run at this point in the installation:

```
* $$ JOB JNM=PSERVER,CLASS=0,DISP=D
* $$ LST CLASS=W,DISP=H,DEST=(,xxxxx)
// JOB PSERVER
// LIBDEF PROC,SEARCH=(PRD2.DB2720)
// DLBL SQLGLOB,'DB2.SQLGLOB.MASTER',,VSAM,CAT=DB2UCAT,DISP=(OLD,KEEP)
// EXEC PROC=ARIS72PL        *-- DB2 PROD. LIBRARY ID PROC
// EXEC PROC=ARIS72DB        *-- DB2 DATABASE ID PROC
// EXEC  ARIDBS,SIZE=AUTO
CONNECT SQLDBA IDENTIFIED BY SQLDBAPW;
CREATE PSERVER SPSERV01 AUTOSTART YES;
/*
/&
* $$ EOJ
```

Step 6: Set Up for Stored Procedures

Stored procedures need to be catalogued into the system library PRD2.DB2STP.

You must compile your stored procedures with the *reentrant* parameter. For an example of how to do so in a COBOL program, see the RENT option used in skeleton SKDLICMP (which is located in VSE/ICCF library 59).

To create definitions for Stored Procedures, use the skeletons listed under “Step 7: Customize the DB2-Based Connector for VSAM Data Access” on page 83 and “Step 8: Customize the DB2-Based Connector for DL/I Data Access” on page 83.

Stored Procedures are always dedicated to a particular *Stored Procedure Server*.

Step 7: Customize the DB2-Based Connector for VSAM Data Access

If you want to access VSAM data via the DB2-based connector, follow the steps below to create and define the Stored Procedures. The skeletons are provided in VSE/ICCF library 59.

- Step 7.1: Use skeleton SKCRESTP to create Stored Procedures for VSAM data access and define them to DB2.
- Step 7.2: Use skeleton SKCPSTP to compile and link-edit Stored Procedures written in C for VSAM data access.
- Step 7.3: Use skeleton SKVSSAMP to define a VSE/VSAM cluster and load sample data into it.

Step 8: Customize the DB2-Based Connector for DL/I Data Access

If you want to access DL/I data via the DB2-based connector, you must first install the VSE/ESA optional program DL/I VSE 1.11. You must also ensure that APAR PQ39683 has been applied to get the connector support. Then you can follow the steps below. The skeletons are located in VSE/ICCF library 59.

- Step 8.1: Use skeleton SKDLISMP to define and load a DL/I sample database.
- Step 8.2: Use skeleton SKDLISTP to create DB2 Stored Procedures used for accessing the DL/I sample database.
- Step 8.3: Use skeleton SKDLICMP to compile and linkedit COBOL DB2 Stored Procedures used for accessing the DL/I sample database.
- Step 8.4: Customize CICS TS. To access DL/I data via the DB2-based connector, you must customize a CICS TS - DL/I online system:
  1. Configure your CICS/DLI online system, as described in:
     - Part 6 of the *DL/I Resource Definition and Utilities* manual.
     - The section "CICS – DL/I Tables – Requirements" of the *DL/I Resource Definition and Utilities* manual.
     - Section "Migrating to DL/I VSE 1.11 and the CICS Transaction Server for VSE/ESA 1.1" of the *DL/I 1.11 Release Guide*.
  2. Define the sample database STDIDBP and other databases you wish to access, to CICS (using either the CICS FCT or transaction CEDA).
  3. Provide labels for the sample database STDIDBP (// DLBL STDIDBC ...) and other databases you wish to access.
  4. Create a new DL/I online nucleus (DLZACT generation), by including all DL/I online programs and PSBs that you wish to use. The CICS/DLI mirror program DLZBPC00 must be authorized for PSB STBICLG, used for accessing the DL/I sample database. The CICS/DLI mirror program DLZBPC00 must also be authorized for any other PSBs you wish to use for accessing other DL/I databases.
  5. Account for an increased number of concurrent DLZBPC00 mirror tasks in the CICS/DLI online system: you must accordingly adjust the MAXTASK and CMAXTSK parameters in the DLZACT generation.
  6. Load the DL/I exit routine DLZBSEOT into the SVA.
  7. Start an MPS system.

  DLZMPX00 is SVA-eligible, and is used for accessing DL/I data via the *AIBTDLI interface* (see "Overview of the AIBTDLI Interface" on page 321 for an explanation of DLZMPX00 and the AIBTDLI interface). The AIBTDLI interface uses DLZMPX00 from the SVA (if it resides there), or loads DLZMPX00 into partition space and uses it from there.

Step 9: Start DB2, and Start Stored Procedure Server

Run skeleton **SKDB2STR** to place the startup job **DB2START** into the VSE/POWER reader queue. This skeleton is shown below.

```
* $$ JOB JNM=DB2START,CLASS=S,DISP=L,NTFY=YES
* $$ LST CLASS=Q,DISP=H
// JOB DB2START DB2 SERVER STARTUP JOB
// LIBDEF PROC,SEARCH=(PRD2.DB2720,PRD2.DB2STP)
// LIBDEF PHASE,SEARCH=(PRD1.BASE,PRD2.SCEEBASE,PRD2.DB2720,           X
               PRD2.DB2STP)
// EXEC  PROC=ARIS72SL    *-- DB2 PRODUCTION LIBRARY ID PROC
// EXEC  PROC=ARIS72DB    *-- DB2 DATABASE ID PROC
// ASSGN SYS098,SYSPCH    *-- DB2 ENABLE POWER FOR STORED PROC HANDLER
// EXEC  ARISQLDS,SIZE=AUTO,PARM='TCPPORT=446,NCUSERS=05,DBNAME=SQLDS, X
               DSPLYDEV=B,RMTUSERS=10'
/*
/&
* $$ EOJ
```

You start the *DB2 Server for VSE* by releasing DB2START. The *DB2 Server for VSE* subsequently starts the *Stored Procedure Server* via startup job SPSERV01 retrieved from PRD2.DB2720. You may include DB2START in SKJCL1 for automatic startup processing (also refer to “Step 1: Customize CICS TS” on page 72).

DB2START requires that TCP/IP is running. To ensure that TCP/IP is running, you can insert the following job step in DB2START, before the EXEC ARISQLDS statement:

```
 // EXEC REXX=IESWAITR,PARM='TCPIP00'
 /*
```

where *TCPIP00* is the name of your TCP/IP startup job.

For further startups, you might consider opening the Sample database by inserting the CIRB transaction into your CICS TS startup job, as shown below:

```
// EXEC DFHSIP,SIZE=DFHSIP,PARM='SIT=C3,START=COLD,SEC=NO,STATRCD=OFF,S*
               VA=NO,NEWSIT=YES,DSALIM=8M,EDSALIM=30M,SI',DSPACE=2M,OS3*
               90
/*
CIRB PASSWORD,8,XXX03,1,GER,PRODDBI
/*
```

Step 10: Install DB2 Connect and Establish Client-Host Connection

To finally establish a connection from the client to the VSE/ESA host, you must install DB2 Connect Version 6 Release 1 or later, on your middle-tier (if not already installed). You must then *configure* DB2 Connect to enable it to access DB2 data stored on the VSE/ESA host. To do so, you can use either use the *Client Configuration Assistant* (CCA), or the DB2 command-line interface, *as described below*.

The sample database **sqlds** is stored on the VSE/ESA host, and is used together with the DB2-Based Connector. To define the **sqlds** database to DB2 Connect on the middle-tier so that it has the alias **db2vsewm**, you must:

1. Define the communication protocol between DB2 Connect on the middle-tier, and the database (**sqlds**) residing on DB2 Server for VSE. To do so, you define a *node*, as follows:

   ```
   db2 catalog protocol node nodename remote ip-addr server port-nr
   ```

   For the DB2-Based Connector samples (the DL/I applet and VSAM applet), you would enter a command such as:

   ```
   db2 catalog tcpip node tcpvse remote 9.111.122.33 server 446
   ```

2. Define the entry for the Database Connection Services (DCS). To do so, use this command:

   ```
   db2 catalog dcs database dcs-name as vse-dbname
   ```

   For the DB2-Based Connector samples (the DL/I applet and VSAM applet) you would enter a command such as:

   ```
   db2 catalog dcs database dcsdb as sqlds
   ```

3. Define the *alias* used by the DB2-Based Connector samples. To do so, use this command:

   ```
   db2 catalog dcs-name  as vse-alias-dbname at node nodename authentication dcs
   ```

   For the DB2-Based Connector samples (the DL/I applet and VSAM applet) you would enter a command such as:

   ```
   db2 catalog dcsdb as db2vsewm at node tcpvse authentication dcs
   ```

After you have completed the definition of the database **sqlds** to DB2 Connect, you must now execute the *bind* step:

```
db2 bind path@ddcsvse.lst blocking all sqlerror continue messages msg-file
grant public
```

For example, for Windows NT/2000/XP you would enter:

```
db2 bind db2\bnd\@ddcsvse.lst blocking all sqlerror continue messages log.msg
grant public
```

For detailed information on how to install and customize DB2 Connect, refer to the:

- *DB2 Connect User's Guide*, SC09-2838.

- DB2 home page at: `http://www.ibm.com/db2` which has links to most manuals covering DB2. By selecting **Library** and then **DB2 Publications** you can browse most manuals (in PDF, Postscript, and/or HTML format) concerning both DB2 Server for VSE and DB2 Connect.

Chapter 10. Configuring the VSAM-Via-CICS Service

Until VSE/ESA 2.6, if a VSAM cluster was opened for update by CICS, the VSE Connector Server and DB2 Stored Procedures could only access the same VSAM cluster in read-only mode, *unless* the VSAM shareoption 4 was used. Using VSAM shareoption 4 resulted in performance degradation.

From VSE/ESA 2.6 onwards and using the *VSAM-via-CICS service*, the VSE Connector Server and DB2 Stored Procedures can access VSAM data *without* the restriction that you must use VSAM shareoption 4.

VSAM clusters must be opened *only once* by CICS. The sharing problem no longer exists, and use of VSAM shareoption 4 is no longer required.

This chapter describes how you can take advantage of this performance improvement. It contains these main sections:

- "Configuring the IBM-Supplied CICS System"
- "Configuring a Further CICS System for VSAM-Via-CICS" on page 88
- "How the VSAM-Via-CICS Service Works" on page 89
- "CICS Transactions for Use with VSAM-Via-CICS" on page 89

Configuring the IBM-Supplied CICS System

These CICS programs are included for accessing VSAM via CICS:
- IESCVSRV (server task)
- IESCVMIR (mirror task)
- IESCVSTA (start transaction)
- IESCVSTI (internal start transaction)
- IESCVSTP (stop transaction)

These file types are supported by the VSAM-via-CICS service:
- ESDS
- KSDS
- RRDS
- KSDS-PATH
- ESDS-PATH

To use the VSAM-via-CICS service, you must therefore:

1. Define the VSAM clusters that you wish to access to CICS. To do so, use the `CEDA DEFINE` transaction. Ensure that the VSAM clusters are defined (using `CEDA DEFINE`) so that:
   - All VSAM clusters are enabled.
   - For read-only access, the VSAM clusters are `readable` and `browsable`.
   - For write access, the VSAM clusters are `addable`, `updateable`, and `deleteable`.
2. Ensure that the module IESCVSVA.PHASE is loaded in the SVA, before you start:
   - The VSAM-via-CICS service
   - CICS

© Copyright IBM Corp. 2000, 2003

3. Ensure the VSAM-via-CICS service is active. The IBM-supplied CICS system is configured so that this service will be started *automatically*. If you wish to start this service *yourself*, use transaction ICVA to do so. To stop the VSAM-via-CICS service, use transaction ICVP.
4. Ensure that the applications that are to access VSAM clusters via CICS (either VSE Connector Client applications, or applications that call DB2 Stored Procedures) use these naming conventions:
   - Catalog File ID must be named as:

     `#VSAM.#CICS.CICS applid`

     For example, #VSAM.#CICS.DBDCCICS
   - Cluster File ID must be the same as the one you used for CICS (consisting of 7-characters).

   Here is an example of how to use these naming conventions. Assume a VSAM cluster exists with the name `MY.TEST.CLUSTER`, and that this cluster resides in the VSAM catalog `MY.USER.CATALOG`. The file is defined as `MYTEST` in the CICS system that has applid DBDCCICS.

   To access file `MYTEST` from either a VSE Connector Client application or an application calling DB2 Stored Procedures, you would use these mapping names:

   ```
   Catalog File ID: #VSAM.#CICS.DBDCCICS
   Cluster File ID: MYTEST
   ```

The only change that you therefore must make to your existing programs (VSE Connector Client applications, or applications calling DB2 Stored Procedures) is to ensure that these programs use the naming conventions for VSAM mapping, as described above.

To optimize performance, you can:

- Access VSAM clusters in batch for *read* commands, using the original VSAM name.
- Access VSAM clusters using the VSAM-via-CICS service for *write* commands, using the VSAM-via-CICS service name.

Configuring a Further CICS System for VSAM-Via-CICS

From VSE/ESA 2.6 onwards, each shipped CICS system is configured *by default* so that the VSAM-via-CICS service is active. Therefore, you are not required to perform any customization activities to other CICS systems shipped from VSE/ESA 2.6 onwards.

However, if you wish to configure a CICS system *that was shipped before VSE/ESA 2.6* to make use of the VSAM-via-CICS service, you must:

1. Define the following programs (providing they are not already defined) in your CICS in the same way as they are defined in the IBM-supplied CICS:
   - IESCVSRV (server task)
   - IESCVMIR (mirror task)
   - IESCVSTA (start transaction)
   - IESCVSTI (internal start transaction)
   - IESCVSTP (stop transaction)
2. Define the following transactions (providing they are not already defined) in your CICS in the same way as they are defined in the IBM-supplied CICS:

- ICVS
- ICVM
- ICVA
- ICVP

(For details of these transactions, see "CICS Transactions for Use with VSAM-Via-CICS").

3. If you wish the VSAM-via-CICS service to be *automatically* started, you must:
   a. Add this statement to the CICS PLT table DFHPLTPI (as the *last* statement in the table):

      ```
      DFHPLT TYPE=ENTRY, PROGRAM=IESCVSTI
      ```

   b. Add this statement to the CICS PLT table DFHPLTSD (as the *first* statement in the table):

      ```
      DFHPLT TYPE=ENTRY, PROGRAM=IESCVSTP
      ```

4. Ensure that the IESCVSVA.PHASE is loaded into the SVA (the batch-side support for VSAM access via CICS resides in $IESCVBA.PHASE), using load list $SVACONN.

How the VSAM-Via-CICS Service Works

The VSAM-via-CICS service works in this way:

1. A VSE Connector Client application or an application calling a DB2 Stored Procedure issues a VSAM-access request.
2. The request is sent to the batch partition where the VSE Connector Server or DB2 Stored Procedure is running.
3. The request is forwarded to CICS via XPCC (cross-partition communication).
4. The VSAM-via-CICS service running within the CICS Transaction Server executes the request (for example, reads a record) and passes the data back to the batch partition where the VSE Connector Server or DB2 Stored Procedure is running.
5. The VSE Connector Server or DB2 Stored Procedure returns the data to the application.

CICS Transactions for Use with VSAM-Via-CICS

The following CICS transactions have been provided for use with VSAM-via-CICS service (and are pre-defined in the VSE/ESA 2.6 system):

ICVS The server transaction. Uses program IESCVSRV.

ICVM The mirror transaction. Uses program IESCVMIR.

ICVA You can use this transaction to start the VSAM-via-CICS service. Uses program IESCVSTA.

ICVP You can use this transaction to stop the VSAM-via-CICS service. Uses program IESCVSTP.

VSAM-Via-CICS

Chapter 11. Configuring Your VSE/ESA Host for SSL

This chapter describes the steps you take to configure your VSE/ESA host for Secure Sockets Layer (SSL) support. From VSE/ESA 2.7 onwards, VSE/ESA also provides hardware crypto support, which requires a PCI Cryptographic Accelerator (PCICA) card or equivalent.

You can configure your host for SSL using either IBM-supplied keys and certificates (which you can use, for example, for testing purposes) or using your own keys and certificates. The IBM-supplied keys and certificates use a weak encryption key, but you can start using SSL almost immediately after you have followed the required two implementation steps.

Using your own keys and certificates, the implementation-procedure uses a strong-encryption key, is more complicated, and involves using a Certificate Authority such as the Thawte Corporation. You would use your own keys and certificates, for example, when implementing SSL for internet applications.

This chapter consists of these main sections:

- "Configuring for SSL Using IBM-Supplied Keys/Certificates" on page 92
- "Configuring for SSL Using Your Own Keys/Certificates" on page 95
- "SSL Examples Provided With the Online Documentation" on page 104

You must have configured your VSE/ESA host for SSL before you implement:

- Server authentication in the Java-based connector (see Chapter 12, "Configuring the Java-Based Connector for Server Authentication", on page 105).
- Client authentication in the Java-based connector (see Chapter 13, "Configuring the Java-Based Connector for Client Authentication", on page 115).
- Server authentication and client authentication in CICS Web Support (refer to "Configuring CICS to use SSL" in the *CICS Transaction Server for VSE/ESA, Enhancements Guide*, GC34-5763).
- Server authentication in VSE/POWER networking (refer to the *VSE/POWER Networking*, SC33-6735).
- Server authentication and client authentication in any other installed TCP/IP application that runs on VSE/ESA.

© Copyright IBM Corp. 2000, 2003

Configuring for SSL Using IBM-Supplied Keys/Certificates

Step 1: Activate TCP/IP for VSE/ESA

When you activate *TCP/IP for VSE/ESA* you have access to all TCP/IP functions, including SSL support. Please note that for SSL support, you require the TCP/IP for VSE/ESA Application Pak.

For details of how to activate TCP/IP for VSE/ESA, refer to the *TCP/IP for VSE/ESA IBM Program Setup and Supplementary Information*, SC33-6601.

The instructions for using SSL with VSE/ESA are provided in the manual *TCP/IP for VSE 1.4, SSL for VSE User's Guide*, which you can obtain from either:

- The VSE/ESA Homepage (described on "Where to Find More Information" on page xix).
- Disk 3 (PDFs) of the *VSE Collection* Online Library, SK2T-0060.

Step 2: Catalog Keyring Set Into the VSE Keyring Library

This section describes Job SKSSLKEY, which you use to catalog the IBM-supplied sample *keyring set* into the VSE Keyring Library. Each keyring set consists of:

- A private key, with member type .PRVK
- A server certificate (which includes the public key), with member type .CERT
- A root certificate, with member type .ROOT

During the installation of VSE/ESA, the VSE Keyring Library (CRYPTO.KEYRING) is *automatically* defined on IJSYSCT (the VSAM Master Catalog), and is used for storing keyring sets that are used by:

- CICS Web Support (of the CICS Transaction Server for VSE/ESA).
- The VSE/ESA Java-based connector.
- VSE/POWER PNET SSL.

In the example shown in Figure 32 on page 93, Job SKSSLKEY contains a sample keyring set (consisting of SAMPLE.PRVK, SAMPLE.CERT, and SAMPLE.ROOT) which you can use together with CICS Web Support and/or the VSE/ESA Java-based connector *for testing purposes*. But for production purposes, you should create:

- your own keyring set to be used by the CICS Transaction Server for VSE/ESA.
- your own keyring set to be used by the VSE/ESA Java-based connector. This keyring set must be different to the keyring set used by the CICS Transaction Server for VSE/ESA.

```
* $$ JOB JNM=SETUPSSL,DISP=D,CLASS=0
// JOB SETUPSSL  DEFINE SSL SAMPLE ENVIRONMENT
* ********************************************************************
*
* STEP 1: CREATE RSA PRIVATE KEY 'SAMPLE.PRVK'
*
* ********************************************************************
// OPTION SYSPARM='00'             SYSID OF MAIN TCP/IP PARTITION
// LIBDEF PHASE,SEARCH=(PRD1.BASE)
// EXEC CIALPRVK,SIZE=CIALPRVK,PARM='CRYPTO.KEYRING.SAMPLE'
-----BEGIN RSA Private Key-----
hXNnvtgWEHuF4rhLWODrmJhG7yNyDYhXjTN1sALJEn2wCYsuaqhnmco5WbJOKdPe
g+oFi0o1MrsPQABoDtes/tNfMtTVzS6Vz/5EmpdrO0MlpNdK/QLdzyS5SgSSA0ZN
gWhVe3eY4+2FQb3x8D5pnjhGuMc3NzxZynBa2j+dz5ae8+nAH8qfQRsPfcXU715Y
EbsMmPGZIMuiFU4B3RsKFk27fkeZ5DpCwMht3M3FP8sII20h/R8BgT5UQUPbUxNS
DXVQ94nBWZI3jU5maBmgbYTxt5hk411gJe6TmfI6OdKp7XdZLpDKyS6O+u9vM4m4
bBtBnIb81DPwKyhY5949D7eTAo0o2RSzKfP1Je7PgN5DJl98DkwpW3tjFTl1kmOo
BMycTt/WF576j1R+KD36AB2rRcu9GJ4JKBWSYP3zROFueS5ON8mkN9mKjdKsgQse
cpl0glQAW48LHT6ZfPLifmYX+U3ad2mfiVVm3RyNNxxFtgE0JLZj/6zzyp3S538L
gBIh1PptdTyAF+rAXCop3NShvIq9lxsF0/sGo5RxraQ8GjVjeljhOztP2z9id6vB
GXt1vHTh5XmJT6PPTmR/wTJeF/8rM6PibNLuAjmG4mETj0sSns9lxniXb3qyME/v
btyuSgTNpgWE0Gaq29eauUhEiOD473xc6hkHFkUtm2RqjWOq+PtyNRWrb7xCup0S
9Feklc+GnX6bWbms3IySIak+2WYKFl/EyfAz7FYMVbjKDA30ngZSWpRK1upAIbXv
c24WW317FpIY5Cm8MBKFNfsWf3W4PD1/e+lyvuMGEi20Sk0fGdAXbxsPBhT2VV4t
dh4vQTjgla0nFQjb9gPz0YA39LIaf4pBaS/Uh63BEL4mRaY9BowDbwB5AwmDwwqt
CqZBXCR+vpOoQwXDGXy0Juo8rIRkW+TJDojVBC0DnDZos0b8IpX3ZcAEz5lUVJPF
GovDWJDr4rvGXMS0M2WbTX0tbZFNfqK5k64p5CPwAW8s05nYLmRRpjAolrv/NGYz
mRdqyddhIhl5YivWDxSMh+2hSqXOAd8S/NxeNJ09cKrdkSVSi2KidL1wFgYSxq6M
MX9l6tYUB2vgWOece4tMgWuSie1UE9qitQMit902hwpQdZqIStHXpA//RV3AhiBo
e/z3edoki/AofaCvMeh5lUZ20iI4upoN9CLeLFTbPQQBgRHnCT9TyltG9pACstsN
i+Qsr3iLX4rUSOnpp1zB2wm31aRGqvbKmkBBvOmCkEUQorqfBvC58T1Yolx3vumk
vJcsjTeEJEWB2yPT6tSiDb8Kz+3cdv1VJTkL2ouk4mJezyFkk8kL+IDqJW9vgV8q
FPZZLkdG/h2+W4wux3OccQ==
-----END RSA Private Key-----
/*
* ********************************************************************
*
* STEP 2: CREATE VSE SERVER CERTIFICATE 'SAMPLE.CERT'
*
* ********************************************************************
// OPTION SYSPARM='00'             SysId of main TCP/IP partition
// LIBDEF PHASE,SEARCH=(PRD1.BASE)
// EXEC CIALCERT,SIZE=CIALCERT,PARM='CRYPTO.KEYRING.SAMPLE'
-----BEGIN CERTIFICATE-----
MIICJTCCAc8CBHiMye4wDQYJKoZIhvcNAQEFBQAwgZYxIDAeBgkqhkiG9w0BCQEW
EXZzZWVzYUBkZS5pYm0uY29tMQswCQYDVQQGEwJERTETMBEGA1UEBxMKQm9lYmxp
bmdlbjEUMBIGA1UEChMLSUJNIEdlcm1hbnkxGDAWBgNVBAsTD1ZTRSBEZXZlbG9w
bWVudDEgMB4GA1UEAxMXU2FtcGxlIFJPT1QgQ2VydGlmaWNhdGUwHhcNMDIxMTEy
MTQwMDEwWhcNMDcxMTExMTQwMDEwWjCBoDEkMCIGCSqGSIb3DQEJARYVaW5mb0B5
b3VyLmNvbXBhbnkuY29tMQswCQYDVQQGEwJERTEWMBQGA1UEBxMNWW91ciBsb2Nh
dGlvbjEVMBMGA1UEChMMWW91ciBjb21wYW55MRQwEgYDVQQLEwtEZXZlbG9wbWVu
dDEmMCQGA1UEAxMdU2FtcGxlIFZTRSBTZXJ2ZXIgQ2VydGlmaWNhdGUwXDANBgkq
hkiG9w0BAQEFAANLADBIAkEAvFjIzkhM8Ur5jPVRfNOvwE3f4XKNDJKN2t6ah04A
ijZ5HdMsPzL1xzmL6YQc3fSQry46zvR89PA8tTm71T3aEQIDAQABMA0GCSqGSIb3
DQEBBQUAA0EAGusgSg0xHL8jijzkCWHiI6k+3BC0uzp6NXUDFkXVPzuDp9uUfDi1
dE270tVeiwBbPIMr8IX4S7G8uaDJ8L0RJQ==
-----END CERTIFICATE-----
```

Figure 32. Job SKSSLKEY to Catalog a Sample Keyring Set into the VSE Keyring Library (Part 1 of 2)

```
/*
* ****************************************************************
*
* STEP 3: CREATE ROOT CERTIFICATE 'SAMPLE.ROOT'
*
* ****************************************************************
// OPTION SYSPARM='00'             SysId of main TCP/IP partition
// LIBDEF PHASE,SEARCH=(PRD1.BASE)
// EXEC CIALROOT,SIZE=CIALROOT,PARM='CRYPTO.KEYRING.SAMPLE'
-----BEGIN CERTIFICATE-----
MIICGzCCAcUCBHiLtz0wDQYJKoZIhvcNAQEFBQAwgZYxIDAeBgkqhkiG9w0BCQEW
EXZzZWVzYUBkZS5pYm0uY29tMQswCQYDVQQGEwJERTETMBEGA1UEBxMKQm9lYmxp
bmdlbjEUMBIGA1UEChMLSUJNIEdlcm1hbnkxGDAWBgNVBAsTD1ZTRSBEZXZlbG9w
bWVudDEgMB4GA1UEAxMXU2FtcGxlIFJPT1QgQ2VydGlmaWNhdGUwHhcNMDIxMTEy
MTM1OTAwWhcNMDcxMTExMTM1OTAwWjCBljEgMB4GCSqGSIb3DQEJARYRdnNlZXNh
QGRlLmlibS5jb20xCzAJBgNVBAYTAkRFMRMwEQYDVQQHEwpCb2VibGluZ2VuMRQw
EgYDVQQKEwtJQk0gR2VybWFueTEYMBYGA1UECxMPVlNFIERldmVsb3BtZW50MSAw
HgYDVQQDExdTYW1wbGUgUk9PVCBDZXJ0aWZpY2F0ZTBcMA0GCSqGSIb3DQEBAQUA
A0sAMEgCQQCrLnONFQiXWHsUoPntYZLDBiIEZst07iiXsHXtypXIyHmw3TDmTV0j
/BuwF39m37NpoU16/0YHud9ERMXPF4NjAgMBAAEwDQYJKoZIhvcNAQEFBQADQQAQ
cKdj1cTozx1hAGTlsRm7YpI5PNRBByD2SBkmIChfVf/5aQnpt9igsU6ipenEfo7X
3RZwGWqKvx+QihfytlSX
-----END CERTIFICATE-----
/*
/&
* $$ EOJ
```

Figure 32. Job SKSSLKEY to Catalog a Sample Keyring Set into the VSE Keyring Library (Part 2 of 2)

Configuring for SSL Using Your Own Keys/Certificates

Check Home Page for Latest Information!

Before starting to configure your VSE/ESA host for production SSL, you are advised to check the *VSE/ESA e-business Connectors and Utilities* home page for any new information relating to SSL. The URL is:

```
http://www.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/conmain.htm
```

Step 1: Activate TCP/IP for VSE/ESA

This procedure is the same as when you configure for SSL using your own keys and certificates. Therefore complete the instructions provided in "Step 1: Activate TCP/IP for VSE/ESA" on page 92.

Step 2: Install/Configure Utility CIALCLNT On a Web Client

The CIALCLNT utility provides a Graphical User Interface that enables you to:

1. Create a private key.
2. Send the private key to the VSE/ESA host (as described in "Step 3.2: Start Utility CIALCLNT on the Web Client" on page 96). The private key is stored on the VSE/ESA host in the .PRVK member (see "Step 2: Catalog Keyring Set Into the VSE Keyring Library" on page 92 for details).

For details of how to obtain the CIALCLNT utility, refer to the *TCP/IP for VSE 1.4, SSL for VSE User's Guide*.

The CIALCLNT utility also uses the CIALSRVR utility which runs on the VSE/ESA host, to set up a connection to VSE/ESA (as described in "Step 3: Generate a Key Pair, Request a Server Certificate" below).

To install CIALCLNT on a Web client:

1. Run the SETUP.EXE for CIALCLNT.
2. Update the file VSESYS.TXT (automatically installed into the directory of your choice) so that it contains the IP address of your VSE/ESA host. For example:

   ```
   "VSEDRS", "009", "164", "155", "002", "6045"
   ```

   The port number **6045** will be used by the CIALSRVR utility to receive the key on the VSE/ESA host, as described in "Step 3: Generate a Key Pair, Request a Server Certificate".

Step 3: Generate a Key Pair, Request a Server Certificate

This step requires actions on both the:

- Web client where you installed the CIALCLNT utility
- VSE/ESA host

Step 3.1: Start Utility CIALSRVR on the VSE/ESA Host

Submit job CIALSRVR.JCL on the target VSE/ESA host system (where the VSE Keyring Library is located) *before* you perform "Step 3.2: Start Utility CIALCLNT on the Web Client" on page 96.

```
* $$ JOB JNM=CIALSRVR,CLASS=A,DISP=D
* $$ LST CLASS=A
// JOB CIALSRVR
// OPTION SYSPARM='00'             SysId of main TCP/IP partition
// LIBDEF PHASE,SEARCH=PRD1.BASE
// EXEC CIALSRVR,SIZE=CIALSRVR
SETPORT 6045
/*
/&
* $$ EOJ
```

Figure 33. Job to Start Utility CIALSRVR on the VSE/ESA Host

After starting utility CIALSRVR, this information will be displayed on the console:

```
BG 0001 1Q47I   BG CIALSRVR 44667 FROM (JSCH) , TIME=13:00:54
BG 0000 // JOB CIALSRVR
        DATE 11/30/2000, CLOCK 13/00/54
BG 0000 CIALSRVR=00501000
BG 0000 SETPORT 6045
```

Step 3.2: Start Utility CIALCLNT on the Web Client

1. Start the CIALCLNT utility on your Web client. The *TCP/IP for VSE Key Generator* window is displayed:

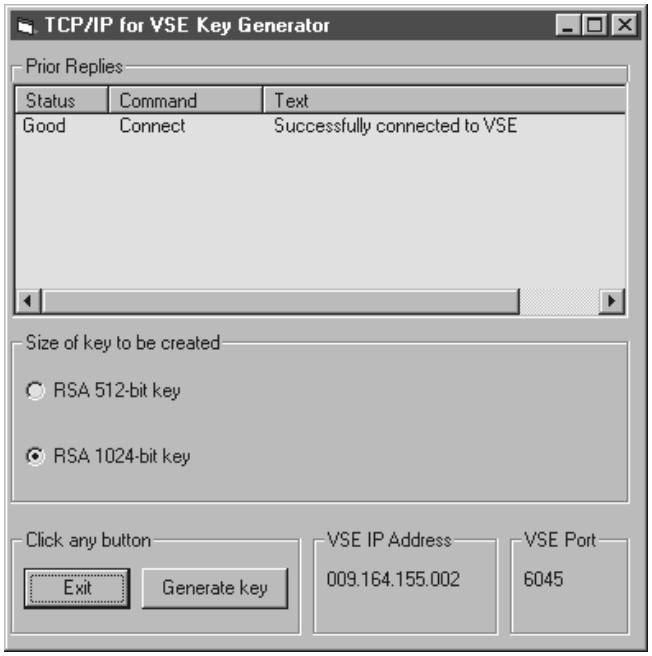


Figure 34. "TCP/IP for VSE Key Generator" window

2. Click **Generate Key** to begin the process of generating an RSA key pair consisting of:
   - a private key
   - a public key
3. The *CIAL_Client* window is displayed. Click **OK** to confirm.
4. The *Generate RSA private key* window is displayed (shown in Figure 35 on page 97) which contains the private key. Now send the private key to the CIALSRVR utility running on the VSE/ESA host, by clicking **Send it to VSE**.

The private and public keys are then sent (in binary format) to the CIALSRVR utility running on the VSE/ESA host.

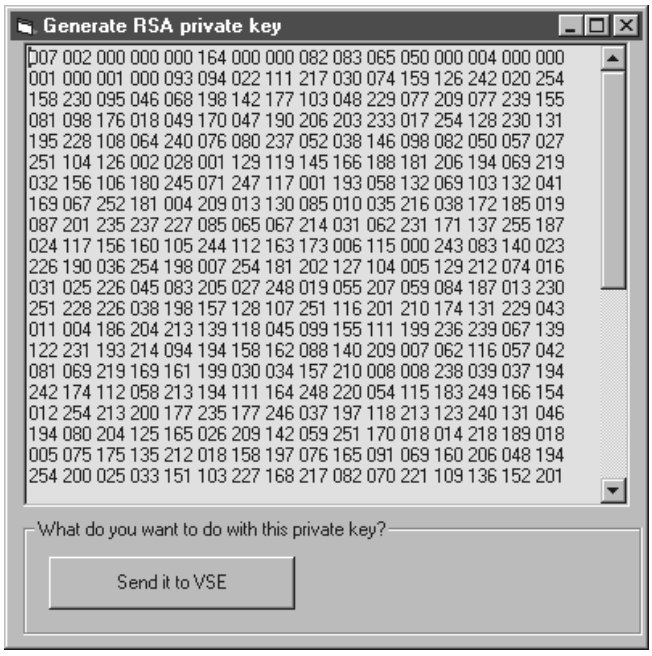


Figure 35. "Generate RSA Private Key" window

5. The CIALSRVR utility running on the VSE/ESA host then catalogs the data that was sent from CIAL_Client into the .PRVK member in the VSE Keyring Library.

Step 3.3: Submit Job CIALCREQ (Submit a Server Certificate Request)

Modify the sample job CIALCREQ.JCL, and submit it to the target VSE system.

You can submit the sample CIALCREQ.JCL either from a Web client or the VSE/ESA host.

Note: If you submit the job from a Web client, ensure you have uppercase translation set to OFF. For example, if you submit the job from a 3270 Emulator, you enter:

```
send cialcreq.job a: (file=rdr crlf nouc
```

Figure 36 on page 98 shows the sample job CIALCREQ.JCL, in which sub-library PRD2.CONFIG contains the $SOCKOPT phase (shown in Figure 32 on page 93).

```
* $$ JOB JNM=CIALCREQ,CLASS=A,DISP=D
* $$ LST CLASS=A
// JOB CIALCREQ
// OPTION SYSPARM='00'             SysId of main TCP/IP partition
// LIBDEF PHASE,SEARCH=(PRD2.CONFIG,PRD1.BASE)
// EXEC CIALCREQ,SIZE=CIALCREQ
Webmaster: vseesa@de.ibm.com
Phone:
Server: SSL for VSE
Common-name: www.ibm.com
Organization Unit: Development
Organization: International Business Machines
Locality: Boeblingen
State: BW
Country: Germany
/*
/&
* $$ EOJ
```

Figure 36. Job CIALCREQ to Request a Server Certificate

The output from this job is sent to the VSE/POWER list queue. An example is shown in Figure 37 on page 99.

Step 4: Obtain a Signed Server Certificate and Copy to Job CIALCERT

During this step, the public key that was created during "Step 3: Generate a Key Pair, Request a Server Certificate" on page 95 is first entered onto a certificate, to create a *server certificate request*. This server certificate request is then sent to a Certificate Authority (CA) to be "signed" and therefore to create a *server certificate*. When a server certificate is signed:

- The CA calculates a *digital signature* over the certificate (including the public key). It uses the CA's own private key to do so.
- Thereafter, clients in an SSL conversation can use the CA's public key which is stored on a *root certificate*, to decrypt the server certificate and use its public key.

Step 4.1: Submit Request for Server Certificate to be Created/Signed

The output from job CIALCREQ.JCL of the previous step, is shown below. It contains mixed characters, and includes the public key that was generated during "Step 3.2: Start Utility CIALCLNT on the Web Client" on page 96.

```
// JOB CIALCREQ
// OPTION SYSPARM='00'            SysId of main TCP/IP partition
// LIBDEF PHASE,SEARCH=(PRD2.CONFIG,PRD1.BASE)
// EXEC CIALCREQ,SIZE=CIALCREQ
CIALCREQ 01.04.00 20001116 12.37
Webmaster: vseesa@de.ibm.com
Phone:
Server: SSL for VSE
Common-name: www.ibm.com
Organization Unit: Development
Organization: International Business Machines
Locality: Boeblingen
State: BW
Country: Germany
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIByzCCATQCAQAwgYoxEDAOBgNVBAYTB0dlcm1hbnkxCzAJBgNVBAgTAkJXMRMw
EQYDVQQHEwpCb2VibGluZ2VuMSgwJgYDVQQKEx9JbnRlcm5hdGlvbmFsIEJ1c2lu
ZXNzIE1hY2hpbmVzMRQwEgYDVQQLEwtEZXZlbG9wbWVudDEUMBIGA1UEAxMLd3d3
LmlibS5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAK+jXsPgzjOjdrYs
RL2izE6bnfMQfjci2w8GwcTx9hQCnllSQoG64GWzUVRk+NMK8GcchcFK/Wq7KtjN
77TwS6k8lE6jKWR6Cy0iQny/HixoV6C8G6/dp0tEy4w3iqkMKk/MiC5r5fJvspin
nkFdyidhWXcdQ81MFvay82VLiUudAgMBAAGgADANBgkqhkiG9w0BAQQFAAOBgQAm
lRNm9J75VnKAG63enqB6tVyNtplYTALruwGfww1cItHn4n1/ylxUWgr3JMEtY+0X
7Dmyt7xxJAUsddtxTpnUGIcUeYCAf7ryZonZ1PhFTlkZsVMOOnMacXMcGRMFYmGX
+49/hdM/IaOYj8AzA8JBXHpXb1TL+8mZpWcwas8/wA==
-----END NEW CERTIFICATE REQUEST-----
1S55I  LAST RETURN CODE WAS 0000
EOJ CIALCREQ  MAX.RETURN CODE=0000
```

Figure 37. Example Output Listing From Job CIALCREQ

The output from job CIALCREQ contains a certificate request. To obtain a signed server certificate (digital certificate), you can now submit this certificate request to any Certificate Authority. This example shows you how to obtain a free *test* signed server certificate from the Thawte Corporation.

1. Start your Web browser and proceed to the Thawte Corporation site where you can obtain a signed SSL Server certificate at:

   ```
   https://www.thawte.com/cgi/server/test.exe
   ```

   After you have registered with Thawte, the site then provides clear instructions on how to obtain a signed certificate.

2. Copy and Paste via the Clipboard the server certificate request details, shown in bold-text within Figure 37, into the appropriate area of the *Test Thawte Certificates – Netscape* window. An example is shown in Figure 38 on page 100:

Figure 38. Thawte "Certificate Signing Request" Window

Complete the other details, such as:

- Validity: you can choose a number of days up to one year.
- Type of test certificate: select **Test X509v3 SSL Cert**.
- Format for Chained CA Certs: select **Use the "standard" format** (the BASE64 encoding of an X.509v3 certificate).

3. Click **Generate Test Certificate**. The output from this request is a signed server certificate such as that shown in Figure 39.

```
-----BEGIN CERTIFICATE-----
MIICsDCCAhmgAwIBAgIDBYAlMA0GCSqGSIb3DQEBBAUAMIGHMQswCQYDVQQGEwJa
QTEiMCAGA1UECBMZRk9SIFRFU1RJTkcgUFVSUE9TRVMgT05MWTEdMBsGA1UEChMU
VGhhd3RlIENlcnRpZmljYXRpb24xFzAVBgNVBAsTDlRFU1QgVEVTVCBURVNUMRww
GgYDVQQDExNUaGF3dGUgVGVzdCBDQSBSb290MB4XDTAwMTEzMDE0NDk1MloXDTAx
MTEzMDE0NDk1MlowgYoxEDAOBgNVBAYTB0dlcm1hbnkxCzAJBgNVBAgTAkJXMRMw
EQYDVQQHEwpCb2VibGluZ2VuMSgwJgYDVQQKEx9JbnRlcm5hdGlvbmFsIEJ1c2lu
ZXNzIE1hY2hpbmVzMRQwEgYDVQQLEwtEZXZlbG9wbWVudDEUMBIGA1UEAxMLd3d3
LmlibS5jb20wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAK+jXsPgzjOjdrYs
RL2izE6bnfMQfjci2w8GwcTx9hQCnllSQoG64GWzUVRk+NMK8GcchcFK/Wq7KtjN
77TwS6k8lE6jKWR6Cy0iQny/HixoV6C8G6/dp0tEy4w3iqkMKk/MiC5r5fJvspin
nkFdyidhWXcdQ81MFvay82VLiUudAgMBAAGgJTAjMBMGA1UdJQQMMAoGCCsGAQUF
BwMBMAwGA1UdEwEB/wQCMAAwDQYJKoZIhvcNAQEEBQADgYEAXLBSZq1l1X36JU1A
1HX+xg218B7F8rZGmiM9GqvVfC5bsAkbJYZC7BcEevrTsTFStAe2XfExkXj2jdFn
GEgejK7ByI1RYvcAuJrTMJYtKCQV0HAtzOibKg6mJOihr60uj23c+LOxtyWfss+3
9xHLMQztYJ32YelZKFTSNbC4rkA=
-----END CERTIFICATE-----
```

Figure 39. A Thawte Signed Server Certificate

Step 4.2: Copy/Paste Signed Server Certificate Into Job CIALCERT

Now Copy and Paste via the Clipboard the signed server certificate (shown in Figure 39 on page 100) into a CIALCERT job, as shown in Figure 40.

```
* $$ JOB JNM=CIALCERT,CLASS=0,DISP=D
* $$ LST CLASS=A
// JOB CIALCERT
// OPTION SYSPARM='00'           SysId of main TCP/IP partition
// LIBDEF PHASE,SEARCH=(PRD1.BASE)
// EXEC CIALCERT,SIZE=CIALCERT,PARM='CRYPTO.KEYRING.PROD1024'
-----BEGIN CERTIFICATE-----
MIICtDCCAh0AAAAAwkQwDQYJKoZIhvcNAQEFBQAwgaAxIDAeBgkqhkiG9w0BCQEW
EXZzZWVzYUBkZS5pYm0uY29tMQswCQYDVQQGEwJERTEbMBkGA1UECBMSQmFkZW4t
V3VlcnR0ZW1iZXJnMRMwEQYDVQQHEwpCb2VibGluZ2VuMQwwCgYDVQQKEwNJQk0x
FDASBgNVBAsTC0lCTSBHZXJtYW55MRkwFwYDVQQDExBURVNUMDcgUk9PVCBDZXJ0
MB4XDTAyMTExMjA5MTUwNFoXDTAzMTExMjA5MTUwNFowgaAxHjAcBgkqhkiG9w0B
CQEWD2JsYWhAZGUuaWJtLmNvbTELMAkGA1UEBhMCREUxGzAZBgNVBAgTEkJhZGVu
LVd1ZXJ0dGVtYmVyZzETMBEGA1UEBxMKQm9lYmxpbmdlbjEMMAoGA1UEChMDSUJN
MRQwEgYDVQQLEwtEZXZlbG9wbWVudDEbMBkGA1UEAxMSVEVTVDA3IFNlcnZlciBD
ZXJ0MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCNH1UAPPobaRKShjuxZ7ol
TpQh4LNgzBTBkpt2FhfPtz3dSvxbMSsOaOfmDXGrgEiCdzWoln7GxrwNGBRfoYih
6JzKFSLGJ2HkCKujAJ1VqgSrsS30c+0PUcGUdB78NgiYYo0Glo8CurrtPemZlExX
wvVqJOXO2mpbR4fJkXi+YwIDAQABMA0GCSqGSIb3DQEBBQUAA4GBAFO6inZU9nGq
8MqiEARXTbsEpUTZSkM2Ez2URIMenHMUnWp/grDcajrG8IrPuG5U0XW2o4nilhBk
laZaET+Xfm1YSJuN87lnJ0sLPhJ8GrHe9JvK648Jokn2nxU0wINtwtZOKfjDlNum
jgZfiZUZAE39HRCgOM3YvnjHAACXpsY8
-----END CERTIFICATE-----
/*
/&
* $$ EOJ
```

Figure 40. Job CIALCERT to Catalog the Server Certificate

Now you should submit this job.

Step 5: Obtain a Root Certificate and Copy to Job CIALROOT

A *root certificate* is supplied by a Certificate Authority, and is the "starting point" for using SSL. You must obtain a root certificate for applications that both:

- run on your VSE/ESA host
- are to use SSL security

On Web clients, the root certificate enables clients to verify the *server certificates* that the clients receive. "Verify" means checking that the server certificate has been issued by an *approved* Certificate Authority, and not simply by *any* person or organization. The server certificate includes a public key, organization's name, address, and so on.

On the server (the VSE/ESA host), the root certificate enables the server to verify any *client certificates* that the server receives. This takes place when *client authentication* has been implemented (see Chapter 13, "Configuring the Java-Based Connector for Client Authentication", on page 115 for details).

Usually, each Web client stores a number of root certificates. Each root certificate has been issued from a different Certificate Authority (CA). Therefore, if the Web client receives a server certificate that has been signed by a CA, there is a good chance that the Web client can use a root certificate issued by the *same* CA, to decrypt the server certificate (and its public key). But even if the Web client does

not have a root certificate issued by the required CA, this is not a problem since the CAs are arranged in a kind of “tree structure”. The Web client can search the CAs until a required root certificate has been found.

After you have obtained a root certificate, by copying and renaming it, you can use the *same* root certificate, for example, for:

- The VSE Connector Server
- CICS1 (a CICS Transaction Server running in a partition)
- CICS2 (a second CICS Transaction Server running in a different partition)
- CICS3 (a third CICS Transaction Server running in a different partition)

The procedure below describes how you might obtain a free root certificate from the Thawte Corporation for a *trial period*.

1. Start your Web browser, and proceed to this Thawte Web page:

   ```
   https://www.thawte.com/cgi/server/test.exe
   ```

2. When you arrive at the Web page, the introductory paragraph reads:

   ```
   Welcome to our Test CA. Please note that the certs issued here are for
   testing and evaluation only ...
   ```

   Click the link from which you obtain a root certificate in text format.

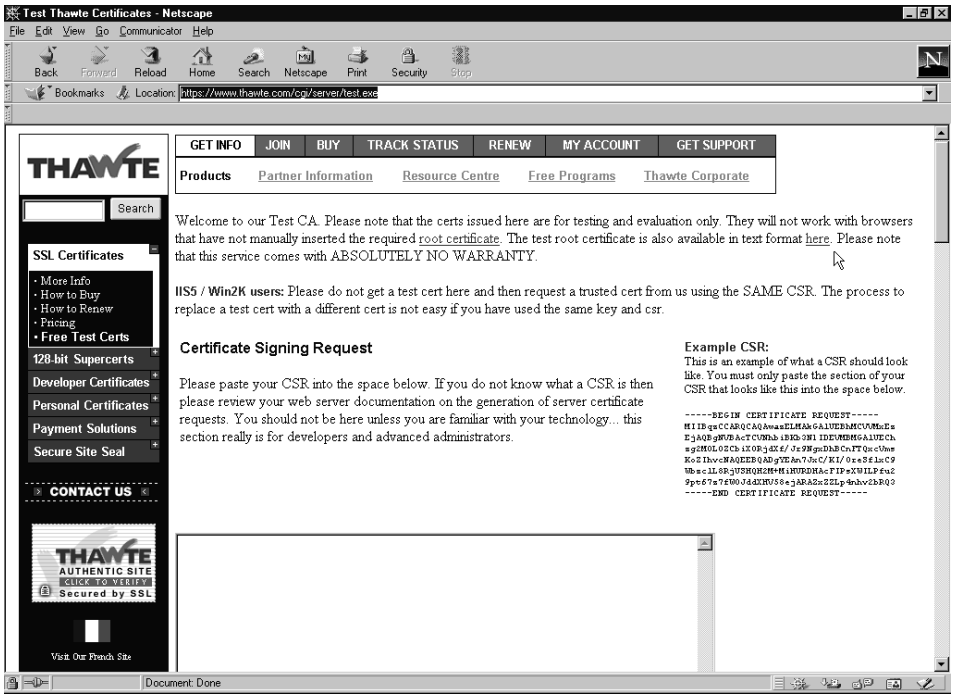


Figure 41. Thawte Certificate Authority Web Site

3. Copy and Paste (via the Clipboard) the root certificate’s contents into Job CIALROOT, in Figure 42 on page 103.

```
* $$ JOB JNM=CIALROOT,CLASS=0,DISP=D
* $$ LST CLASS=A
// JOB CIALROOT
// OPTION SYSPARM='00'           SysId of main TCP/IP partition
// LIBDEF PHASE,SEARCH=PRD1.BASE
// EXEC CIALROOT,SIZE=CIALROOT,PARM='CRYPTO.KEYRING.PROD1024'
-----BEGIN CERTIFICATE-----
MIICLjCCAAAAAAdem7MwDQYJKoZIhvcNAQEFBQAwXjEWMBQGA1UEAxMNSlNDSDEw
MjQgUk9PVDEUMBIGA1UECxMLRGV2ZWxvcG1lbnQxDDAKBgNVBAoTA0lCTTETMBEG
A1UEBxMKQm9lYmxpbmdlbjELMAkGA1UEBhMCREUwHhcNMDIxMDE4MTE1ODQzWhcN
MDMxMTE3MjMwMDAwWjBeMRYwFAYDVQQDEw1KU0NIMTAyNCBST09UMRQwEgYDVQQL
EwtEZXZlbG9wbWVudDEMMAoGA1UEChMDSUJNMRMwEQYDVQQHEwpCb2VibGluZ2Vu
MQswCQYDVQQGEwJERTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA5hLlm9O5
piEONbipLmKoLZcnCae7m04yO2EKf7BxggWN8jEro2dN5U/rGuvjD40rAbbtCm/J
Nems3GKJffQjrs/GoJwYuq7ExIgv02kPBsryTSnMAfy7EeIoZcSZ7sIJRZCuvLRk
HRkEp8LaSYFgbU2wp+3tjpI8FvvBTELZZ5MCAwEAATANBgkqhkiG9w0BAQUFAAOB
gQAmXSS1jGsXZOgQMxiH5x7/CVhIto6Mx9cVS0Tg6eRU2QjrqxPchogmVctUWN7L
BzCWLNjPtw2KecDhB2RrBB467W+WoNRFbn+sTz3RgjwGNAC5DdjqtiV2JzvLZ8DR
4Qqz6fY/HwOWKzjuI3nSCarpXSF0ArTVHbULbvCWC8D8fg==
-----END CERTIFICATE-----
/*
/&
* $$ EOJ
```

Figure 42. Job CIALROOT to Catalog the Root Certificate

4. Submit this job to your VSE/ESA host.

Step 6: Verify Your Certificates on the Host

To verify that your root certificate and server certificate are valid and correct, submit the job CIALSIGV.JCL. The resulting output is shown in Figure 43:

```
// JOB CIALSIGV                                          DATE 11/30/2
// OPTION LOG
// OPTION SYSPARM='00'             SysId of main TCP/IP partition
*
* * The following is a sample job that will validate the
* * signature in the VSE SSL server certificate is signed by
* * the installed root certificate.
* * It can be used for initial testing on your system.
* *
*
// LIBDEF PHASE,SEARCH=(PRD2.CONFIG,PRD1.BASE)
// EXEC CIALSIGV,SIZE=CIALSIGV,PARM='CRYPTO.KEYRING.PROD1024'
The digital signature contained in the CERTFIL is verified with the
ROOTFIL public key.
Root certificate contains a 1024-bit public key.
Certificate signature successfully verified.
1S55I  LAST RETURN CODE WAS 0000
EOJ CIALSIGV  MAX.RETURN CODE=0000                    DATE 11/30/2
```

Figure 43. Job CIALSIGV to Verify Certificates

Step 7: Secure Your VSE Keyring Library Entries

You must ensure that your own library members such as private keys, server certificates, and root certificates are *securely* stored in the VSE Keyring Library (CRYPTO.KEYRING). To do so, use the:

- VSE/ESA *Access Control Function*, which is part of the Basic Security Manager (BSM).
- Access control table **DTSECTAB**. The default VSE Keyring Library (CRYPTO.KEYRING) is secured with all parts.

For details, refer to the section ″Protecting Resources with Access Control Table DTSECTAB″ of the VSE/ESA Administration, SC33-6705.

In addition, you must:

- Ensure that your system is started with `SEC=YES` in the IPL SYS parameter.
- Ensure that your private key and server certificate have full read/write access protection.
- Include an ID statement in the startup job of each application that is to use certificates for SSL. The application might be a CICS application or the VSE Connector Server, for example.
- Submit the startup job for an application:
  - when security is active (that is, `SEC=YES` in the IPL SYS parameter).
  - using an *authorized* user (for example, the Administrator).
- Ensure that your VSE Keyring Library (the IBM-default is CRYPTO.KEYRING) cannot be accessed via FTP. For example, you should not specify the name of your VSE Keyring Library in any `DEFINE FILE` statement.

SSL Examples Provided With the Online Documentation

You might also refer to the SSL examples provided with the VSE Connector Client's online documentation:

- **SSLApiExample** shows how to code a Java application to connect to the VSE Connector Server via SSL.
- **SSLConsoleExample** shows how to connect via SSL, to submit a console command, and then obtain the resulting console messages.

Both of these examples are ready-to-run, and use the IBM-provided Client Keyring File (**Keyring.pfx**). In addition, you must also have submitted job SKSSLKEY to catalog the corresponding entries in the VSE Keyring Library (CRYPTO.KEYRING). For details, see "Step 2: Catalog Keyring Set Into the VSE Keyring Library" on page 92.

You can find the complete Java source code for the above SSL examples in the `Samples` sub-directory of the directory where you installed the VSE Connector Client. For details of how to use the online documentation, see "Using the Online Documentation Options" on page 28.

Chapter 12. Configuring the Java-Based Connector for Server Authentication

This chapter describes how you configure the Java-based connector (that is, the VSE Connector Server and VSE Connector Client) for *server authentication*.

Before starting this section, you should have first completed the actions described in Chapter 11, "Configuring Your VSE/ESA Host for SSL", on page 91.

Note: If you require *client authentication* in addition to server authentication for your Java-based connector, after completing the steps in this chapter you must then complete the steps described in Chapter 13, "Configuring the Java-Based Connector for Client Authentication", on page 115.

This chapter contains these main sections:
- "Configuring the VSE Connector Server for Server Authentication"
- "Configuring the VSE Connector Client for Server Authentication" on page 108

Configuring the VSE Connector Server for Server Authentication

These are the changes you must make to *each* VSE Connector Server installation, in order to implement server authentication.

You can run the VSE Connector Server either in SSL-mode, or in non-SSL-mode.

Note: You cannot have both SSL-mode and non-SSL-mode connections using the *same* VSE Connector Server. However you can, of course, run multiple VSE Connector Servers in different partitions on the VSE/ESA hostVSE/ESA host, and with or without SSL.

This section contains these main sub-sections:
- "Step 1: Configure and Catalog the VSE Connector Server's SSL Profile"
- "Step 2: Activate SSL Profile in Main Configuration File" on page 106

Step 1: Configure and Catalog the VSE Connector Server's SSL Profile

Use job skeleton SKVCSSSL (in ICCF library 59) to configure and catalog the VSE Connector Server's SSL profile on the VSE/ESA host. Here is an example of skeleton SKVCSSSL:

© Copyright IBM Corp. 2000, 2003

```
; ****************************************************************
;      SSL CONFIGURATION MEMBER FOR VSE CONNECTOR SERVER
; ****************************************************************

; ****************************************************************
; SSLVERSION   SPECIFIES THE MINIMUM VERSION THAT IS TO BE USED
;              POSSIBLE VALUES ARE:   SSL30 AND TLS31
; KEYRING      SPECIFIES THE SUBLIBRARY WHERE THE KEY FILES ARE
;              STORED.
; CERTNAME     NAME OF THE CERTIFICATE THAT IS USED BY THE SERVER
; SESSIONTIMEOUT NUMBER OF SECONDS THAT THE SERVER WILL USE TO
;              ALLOW A CLIENT TO RECONNECT WITHOUT PERFORMING A
;              FULL HANDSHAKE. (86440 SEC = 24 HOURS)
; AUTHENTICATION TYPE OF AUTHENTICATION. POSSIBLE VALUES ARE:
;              SERVER - SERVER AUTHENTICATION ONLY
;              CLIENT - SERVER AND CLIENT AUTHENTICATION
;              LOGON  - SERVER AND CLIENT AUTENTICATION WITH LOGON.
;                       THE CLIENT CERTIFICATE IS USED FOR THE LOGON.
; ****************************************************************
SSLVERSION     = SSL30
KEYRING        = CRYPTO.KEYRING
CERTNAME       = SAMPLE
SESSIONTIMEOUT = 86440
AUTHENTICATION = SERVER

; ****************************************************************
; CIPHERSUITES SPECIFIES A LIST OF CIPHER SUITES THAT ARE ALLOWED
; ****************************************************************
CIPHERSUITES =   ; COMMA SEPARATED LIST OF NUMERIC VALUES
              01,  ; RSA512_NULL_MD5
              02,  ; RSA512_NULL_SHA
              08,  ; RSA512_DES40CBC_SHA
              09,  ; RSA1024_DESCBC_SHA
              0A,  ; RSA1024_3DESCBC_SHA
              62   ; RSA1024_EXPORT_DESCBC_SHA
```

Figure 44. Skeleton SKVCSSSL (Configure SSL for the VSE Connector Server)

Notes:

1. You specify cipher suites as a list of hex numbers.
2. For a complete list of supported cipher suites refer to Table 3 on page 112.
3. The cipher suite with hex number 62 (shown above) is currently not supported by the VSE Connector Client.

Step 2: Activate SSL Profile in Main Configuration File

Use job skeleton SKVCSCFG (in ICCF library 59) to enable SSL and catalog the SSL profile in the VSE Connector Server's main configuration file. Here is an example of skeleton SKVCSCFG:

```
:
   SERVERPORT   = 2893
   MAXCLIENTS   = 256
   SSLENABLE    = YES

:
   LIBRCFGFILE   = DD:PRIMARY.TEST(IESLIBDF.Z)
   USERSCFGFILE  = DD:PRIMARY.TEST(IESUSERS.Z)
   PLUGINCFGFILE = DD:PRIMARY.TEST(IESPLGIN.Z)
   SSLCFGFILE    = DD:PRIMARY.TEST(IESSSLCF.Z)

:
```

Figure 45. Skeleton SKVCSCFG (Activate SSL Profile for the VSE Connector Server)

Configuring the VSE Connector Client for Server Authentication

This section describes the changes for SSL server-authentication support that you must make to:

- Each VSE Connector Client installed on Web clients of the 2-tier environment shown in Figure 2 on page 12.
- The VSE Connector Client installed on the middle-tier of the 3-tier environment shown in Figure 3 on page 13.

This section contains these main sub-sections:

- "Step 1: Set SSL Flag in Class VSEConnectionSpec"
- "Step 2: Configure SSL Profile" on page 109
- "Step 3: Copy a Server Certificate Into Client Keyring File" on page 110
- "Description of the IBM-Supplied Client Keyring File" on page 111
- "Currently-Supported SSL Cipher Suites" on page 112

Step 1: Set SSL Flag in Class VSEConnectionSpec

From VSE/ESA 2.6 onwards, class *VSEConnectionSpec* of the VSE Connector Client's VSE Java Beans supports an SSL flag, which can be set for your user applications. When this flag is set, you must also specify additional SSL-related parameters. To do so, you can either:

- Define a Java properties object.
- Create a Java properties file to contain the required SSL parameters.

In the example shown in Figure 46 on page 109, the SSL parameters are set using a Java *properties object*.

```
.
.
.
try {
   spec = new VSEConnectionSpec(
                InetAddress.getByName(ipAddr),
                2893,userID,password);
}
catch (UnknownHostException e) { ... }

/* Specify secure SSL connection */
spec.setSSL(true);

/* Specify SSL properties */
sslProps = new Properties();
sslProps.put("SSLVERSION", "SSL");
sslProps.put("CIPHERSUITES",
     "SSL_RSA_WITH_NULL_MD5," +
     "SSL_RSA_WITH_NULL_SHA," +
     "SSL_RSA_EXPORT_WITH_DES40_CBC_SHA," +
     "SSL_RSA_WITH_DES_CBC_SHA," +
     "SSL_RSA_WITH_3DES_EDE_CBC_SHA");
sslProps.put("KEYRINGFILE", "c:\\vsecon\\KeyRing.pfx");
sslProps.put("KEYRINGPWD", "ssltest");
spec.setSSLProperties(sslProps);

/* Create VSE system instance with this connection */
system = new VSESystem(spec);

/* Connect to host */
system.connect();

.
.
.
```

Figure 46. Set VSE Connector Client's SSL Parameters Using a Properties Object

Step 2: Configure SSL Profile

Configure the SSL profile for each VSE Connector Client. To do so, you must create a *Java properties file* containing these pairs of keys / values (an example of which is shown in Figure 47 on page 110:

Key	Value
SSLVERSION	SSL or TLS
CIPHERSUITES	Specifies a list of symmetric encryption/decryption algorithms which the VSE Connector Client can use to negotiate the cipher used in the related connection. For an SSL-connection to be established, both the VSE Connector Client and the VSE Connector Server must support the same cipher with the same encryption strength (40-bit, 56-bit, 128-bit, and so on). For a list of the currently-supported cipher suites, see Table 3 on page 112.
KEYRINGFILE	Pathname of a *client keyring file* which is stored on either each Web client of a 2-tier environment or on the middle-tier of a 3-tier environment, and is protected by a password. See "Description of the IBM-Supplied Client Keyring File" on page 111 for a description of the keyring file.

Note: The name that you define here must also be used when you create your client keyring file for the first time. See "Step 2: Generate and Store a Client Certificate" on page 118 for details.

KEYRINGPWD
: Password to protect the client keyring file. For the client keyring file automatically created during installation, "ssltest" is the pre-set password.

LOGONWITHCERT
: YES or NO. An *implicit logon* is possible providing:
1. The Client Keyring File contains a client certificate which is *also* stored as a .CCERT member in the VSE Keyring Library on the VSE/ESA host.
2. The .CCERT member in the VSE Keyring Library has been mapped to a VSE/ESA User ID.

An implicit logon means that the VSE Connector Client does not have to provide logon information (that is, a User ID and password) explicitly. If LOGONWITHCERT is set to YES, you must also specify AUTHENTICATION=LOGON in the VSE Connector Server's configuration file (see Figure 44 on page 106 for details).

You can use the hash character '**#**' as a comment delimiter.

Here is an example of a Java properties file:

```
SSLVERSION=SSL    # SSL or TLS
CIPHERSUITES=SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
KEYRINGFILE=c:\\vsecon\\keyring.pfx
LOGONWITHCERT=NO
```

Figure 47. Example of Java Properties File for the VSE Connector Client

Notes:

1. You are not required to define *all* SSL parameters in your Java properties file.
2. If you do not define an SSL parameter in your Java properties file, you must ensure that your user application requests any missing information from the user, before connecting to the server.
3. In Figure 47, parameter KEYRINGPWD has not been specified.

Step 3: Copy a Server Certificate Into Client Keyring File

A *server certificate* includes a public key, and is required so that the VSE Connector Client can perform *server authentication*.

Decide if you wish to use the IBM-supplied sample server certificate with each VSE Connector Client, or if instead you wish to use your *own* server certificate. Depending upon your decision, you must copy either:

- the IBM-supplied sample server certificate (`SAMPLE.CERT`),
- your own server certificate (see "Step 4: Obtain a Signed Server Certificate and Copy to Job CIALCERT" on page 98 for details),

from the VSE Keyring Library on the VSE/ESA host, into the client keyring file (**KeyRing.pfx**) located on either:

- Each Web client of a 2-tier environment.
- The middle-tier of a 3-tier environment.

See “Description of the IBM-Supplied Client Keyring File” for details of the client keyring file.

For 2-tier environments, if a Web client does not contain a copy of the server certificate and communicates with the VSE/ESA host, the user will be prompted to decide if a copy of the server certificate should be stored on the Web client. You can therefore either:
- Provide each Web client with a copy of the server certificate before starting.
- Let each Web client take a copy during the first processing that takes place.

Note: You can use a tool such as the IBM KeyMan (which you can download from **http://www.alphaworks.ibm.com**) to copy a server certificate into the client keyring file stored on a Web client.

Description of the IBM-Supplied Client Keyring File

During the installation of the VSE Connector Client, a client keyring file with the name **KeyRing.pfx** is automatically stored in the directory:

```
\vsecon\samples
```

on either each Web client (for a 2-tier environment) or on the middle-tier (for a 3-tier environment).

After being created during the installation, the client keyring file contains a sample *server certificate*, which includes a public key. This public key is one of a key pair – the matching private key (SAMPLE.PRVK) is installed on the VSE Connector Server.

Providing you have already run job SKSSLKEY (see Figure 32 on page 93), the sample server certificate enables you to begin using the Java-based connector with SSL immediately after installation, to communicate between the VSE Connector Client and the VSE Connector Server.

The client keyring file is initially protected by the password ‘ssltest’.

To:
- change the password,
- manage the certificates stored in the client keyring file,

you can use:

1. The *IBM KeyMan* tool, which you can download from **http://www.alphaworks.ibm.com/tech/keyman** .
2. The *ikeyman* tool, contained in the JSSE package of the IBM Java Development Kit for Windows 1.3 onwards.
3. The **keytool.exe**, contained in the Java Development Kit 1.3 onwards from IBM and Sun Microsystems.

Note: The description contained in this manual (for example Figure 48 on page 112 below) is based upon the use of the *IBM KeyMan* tool.

Figure 48 on page 112 shows how the contents of the IBM-supplied client keyring file might look.

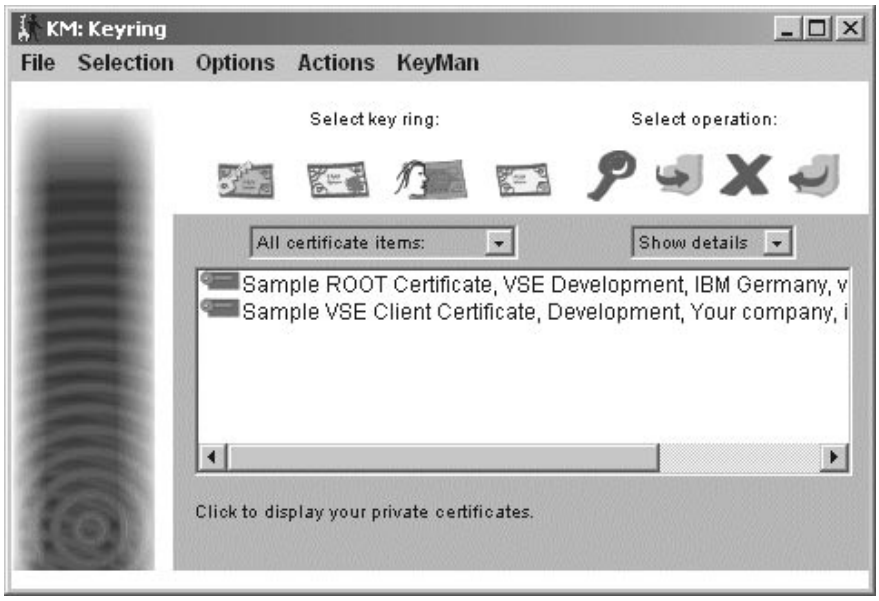


Figure 48. Example of a Client Keyring File

Currently-Supported SSL Cipher Suites

Table 3 shows the SSL cipher suites that are currently supported by the VSE Connector Client and VSE Connector Server:

Table 3. Currently Supported SSL Cipher Suites

Hex Code	Cipher Suite	Handshaking	Encryption
01	SSL_RSA_WITH_NULL_MD5	512-bit	No
02	SSL_RSA_WITH_NULL_SHA	512-bit	No
08	SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	512-bit	40-bit
09	SSL_RSA_WITH_DES_CBC_SHA	1024-bit	56-bit
0A	SSL_RSA_WITH_3DES_EDE_CBC_SHA	1024-bit	168-bit

Notes:

1. Table 3 represents the format you use when defining these cipher suites for the VSE Connector Client.
2. Refer to Figure 44 on page 106 for the format you use when defining these cipher suites for the VSE Connector Server.
3. Hex code 62 (VSE name RSA1024_EXPORT_DESCBC_SHA) is not supported by the VSE Connector Client. For a detailed description of the SSL V3 protocol and supported cipher suites, refer to the *SSL Protocol Version 3.0*.
4. The exportable ciphers 01 and 02 (also 62) require the SSL 3.0 handshaking. They cannot be used with TLS 1.0 handshaking. The other ciphers (08, 09, 0A) can be used with both SSL 3.0 and TLS 1.0, handshaking. You define the SSL Version using `SSLVERSION`, which is contained in the:
   - Java properties object of the VSE Connector Client (see Figure 46 on page 109).
   - Java properties file of the VSE Connector Client (see Figure 47 on page 110).

- SSL configuration file of the VSE Connector Server (see Figure 44 on page 106).

5. The NULL_MD5 (01) and NULL_SHA (02), and DES40_CBC_SHA (08) cipher suites require to have a 512-bit key on the VSE side. They cannot be used with a 1024-bit key.
6. The cipher suites that use 1024-bit handshaking, also require a 1024-bit key on the VSE/ESA host.
7. See "Step 2: Generate and Store a Client Certificate" on page 118 for other information about key lengths used for client authentication.

VSE Connector Client for Server Authentication

Chapter 13. Configuring the Java-Based Connector for Client Authentication

This chapter describes the steps you must take to configure the Java-based connector (that is, the VSE Connector Server and VSE Connector Client) for *client authentication*.

Digital certificates can be either root certificates, server certificates, or client certificates. If client authentication is required (in addition to server authentication), a *client certificate* is provided by clients to authenticate the client to the server.

To configure the Java-based connector for client authentication, you must configure:

- The VSE Connector Server installed on the VSE/ESA host.
- Each VSE Connector Client installed on Web clients of a 2-tier environment.
- The VSE Connector Client installed on the middle-tier of a 3-tier environment.
- The client-certificate/User-ID mapping list on the VSE/ESA host, which you can use to control access rights from VSE Connector Clients to VSE/ESA host resources.

Notes:

1. If you do not plan to implement client authentication, you can skip this chapter.
2. The description and pictures contained in this chapter are based upon the *IBM KeyMan* tool.
3. Instead of using the IBM Keyman tool, you can use for example the *ikeyman* tool (contained in the JSSE package of the IBM Java Development Kit for Windows 1.3 onwards), or the **keytool.exe** (part of the Java Development Kit 1.3 onwards, from IBM and Sun Microsystems).

This chapter consists of these main sections:

- "Configuring the VSE Connector Server for Client Authentication" on page 116
- "Configuring the VSE Connector Client for Client Authentication" on page 117

© Copyright IBM Corp. 2000, 2003

Configuring the VSE Connector Server for Client Authentication

To configure the VSE Connector Server for client authentication, you use SSL configuration member SKVCSSSL contained in ICCF Library 59. An example of the required settings is shown in Figure 49.

```
; ****************************************************************
;      SSL CONFIGURATION MEMBER FOR VSE CONNECTOR SERVER
; ****************************************************************

; ****************************************************************
; SSLVERSION   SPECIFIES THE MINIMUM VERSION THAT IS TO BE USED
;              POSSIBLE VALUES ARE:   SSL30 AND TLS31
; KEYRING      SPECIFIES THE SUBLIBRARY WHERE THE KEY FILES ARE
;              STORED.
; CERTNAME     NAME OF THE CERTIFICATE THAT IS USED BY THE SERVER
; SESSIONTIMEOUT NUMBER OF SECONDS THAT THE SERVER WILL USE TO
;              ALLOW A CLIENT TO RECONNECT WITHOUT PERFORMING A
;              FULL HANDSHAKE. (86440 SEC = 24 HOURS)
; AUTHENTICATION TYPE OF AUTHENTICATION. POSSIBLE VALUES ARE:
;              SERVER - SERVER AUTHENTICATION ONLY
;              CLIENT - SERVER AND CLIENT AUTENTICATION
;              LOGON  - SERVER AND CLIENT AUTENTICATION WITH LOGON
;                       THE CLIENT CERTIFICATE IS USED TO LOGON.
; ****************************************************************
SSLVERSION     = SSL30
KEYRING        = CRYPTO.KEYRING
CERTNAME       = SAMPLE
SESSIONTIMEOUT = 86440
AUTHENTICATION = CLIENT or LOGON

⋮
```

Figure 49. Job to Configure the VSE Connector Server for Client Authentication

You can set the parameter AUTHENTICATION to specify:

SERVER Server authentication only.

CLIENT Server authentication *and* client authentication.

LOGON Server authentication *and* client authentication that uses a client certificate to logon.

Note: If you specify that you require client authentication in Figure 49, then *each* client (where a VSE Connector Client is installed) must:

- have a valid client certificate stored in the *client keyring file* (see "Step 2: Generate and Store a Client Certificate" on page 118 for details).
- provide a client certificate when connecting to the VSE Connector Server (it is not possible to have client authentication for *some* clients only).

Configuring the VSE Connector Client for Client Authentication

This section describes the steps you must follow to configure for client authentication:

- each VSE Connector Client installed on Web clients of the 2-tier environment shown in Figure 2 on page 12,
- the VSE Connector Client installed on the middle-tier of the 3-tier environment shown in Figure 3 on page 13,

To configure each VSE Connector Client for client authentication, you should follow these steps:

- "Step 1: Generate a Key Pair"
- "Step 2: Generate and Store a Client Certificate" on page 118
- "Step 3: Import the CA's Root Certificate into the Client Keyring File" on page 120
- "Step 4: Save Your Client Keyring File" on page 121
- "Step 5: Define Access Rights for VSE Connector Client to Use Host Resources" on page 121

Step 1: Generate a Key Pair

During the initial "handshake", SSL uses a "key pair" consisting of:
- A private key
- A public key

This description of how to generate a key pair uses the IBM *KeyMan* tool to generate a key pair (private and public keys). The public key will be later stored on a *client certificate*. You can obtain KeyMan from:

```
http://www.alphaworks.ibm.com/
```

However you can, of course, use another tool of your choice to generate a key pair and store them in the *client keyring file* (described in "Description of the IBM-Supplied Client Keyring File" on page 111).

These are the steps you take to generate a key pair:

1. Start the KeyMan program. Open an existing client keyring file or create a new client keyring file (with PKCS#12 token) by clicking the appropriate icon in the first window.
2. In the *KM: Keyring* window, from the Menu pull-down select **Actions** and **Generate key**. The window shown in Figure 50 on page 118 is displayed.

Figure 50. "Generate Key" Window

3. Select the key length you require, and start the generation process. After a few seconds, a window *KM: New* is displayed, showing the new entry for a key pair.

   Notes:

   a. Although the window has the title *Generate New Key*, there are (always) *two* keys that are generated: a private key and a public key.

   b. You must ensure that the key length you choose matches the length of the key used on the VSE/ESA host:
      - Cipher suites that use 512-bit handshaking can only be used with a 512-bit key that is stored on the VSE/ESA host.
      - Cipher suites that use 1024-bit handshaking can only be used with a 1024-bit key that is stored on the VSE/ESA host.

      Also see "Currently-Supported SSL Cipher Suites" on page 112 for further details.

Step 2: Generate and Store a Client Certificate

In this step, a client certificate is generated and then stored in a client keyring file. The client certificate will include the public key that was generated in the previous step.

1. Click on the key pair that was generated in the previous step, and from the menu pull-down select **Actions — Request Certificate**. The *KM: Request a Certificate* window is displayed, as shown in Figure 51.

Figure 51. "KM: Request a Certificate" Window – Choose Request Method

Now select **Go to an online CA**. Click **Next dialog page** (the arrow pointing to the right), and you are then given a choice of Certificate Authorities (CAs) from which you wish to obtain a certificate.

2. Select the **Thawte Trial Server** and click **Next dialog page**.
3. You are now presented with a dialog into which you enter your personal details (Figure 52).

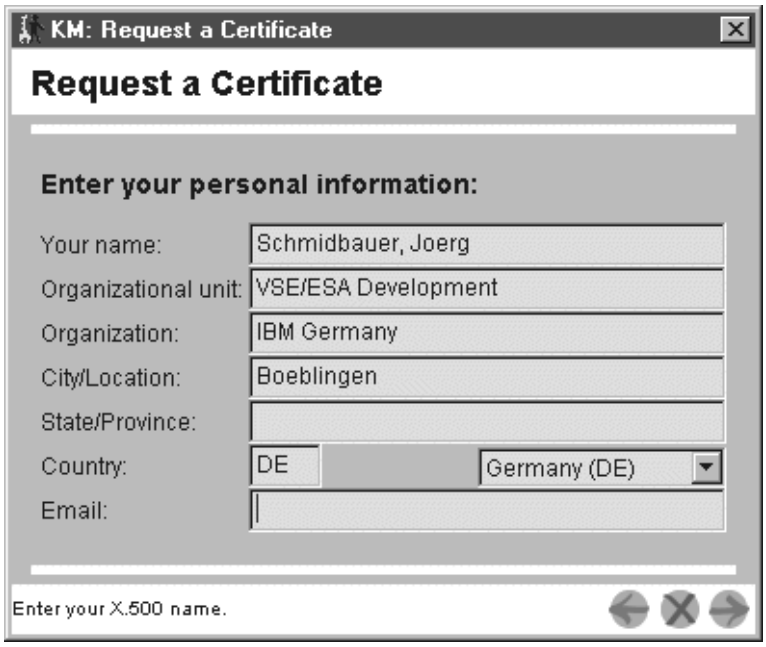


Figure 52. "KM: Request a Certificate" Window – Enter Personal Details

4. KeyMan now launches your default Web browser, and connects to **www.thawte.com**. Your certificate request has been automatically copied onto your Clipboard.
5. Now you must decide if you wish to:
   - purchase a client certificate from Thawte for production use.
   - obtain a ***personal certificate*** from Thawte for trial use, which is free of charge, and use this as a client certificate.

   After you have pasted your certificate request into an area on the Web page provided by Thawte, your client certificate is generated and displayed. Copy this certificate to the Clipboard, and then import it into the client keyring file (Figure 53 on page 120).

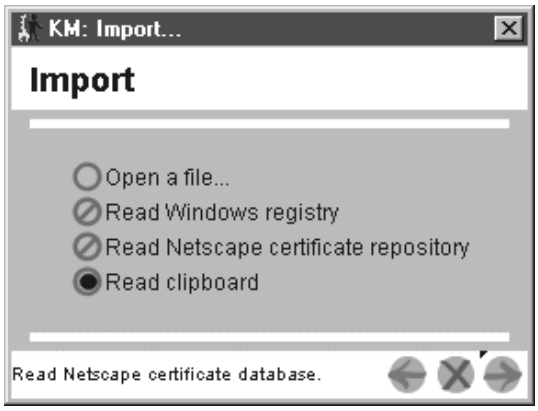


Figure 53. "KM: Import" Window – Import Certificate into Client Keyring File

Also from the Clipboard, paste the client certificate into an ASCII text file on your client workstation, then select **Save As** and give the file a name that you can later easily refer to. This file containing a client certificate will be required in "Step 5: Define Access Rights for VSE Connector Client to Use Host Resources" on page 121.

6. The client keyring file (Figure 54) now shows your client certificate. The public key that you generated has been merged into this client certificate.

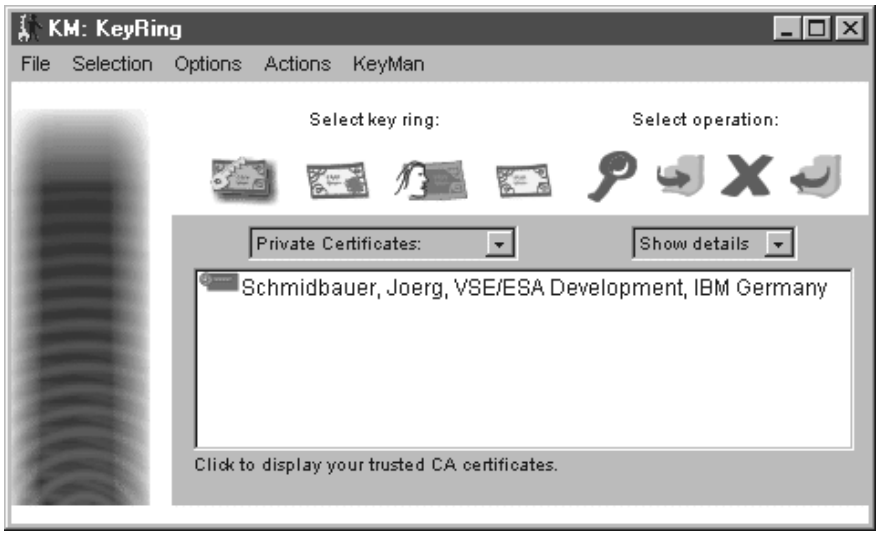


Figure 54. "KM: KeyRing" Window – Display Client Certificate

7. If you created a *new* client keyring file at "Step 2: Generate and Store a Client Certificate" on page 118, save it by selecting **File** and **Save** from the menu pull-down. Then give the new client keyring file a password and file name.

   Note: The name that you give to your client keyring file must also be defined when you configure each VSE Connector Client for SSL. See Figure 47 on page 110 for an example job.

Step 3: Import the CA's Root Certificate into the Client Keyring File

Now return to the Thawte selection panel, and click the option for obtaining the root certificate (that is related to your client certificate).

Ensure that this root certificate is copied to the Clipboard, and then import it into the client keyring file as shown in Figure 55.

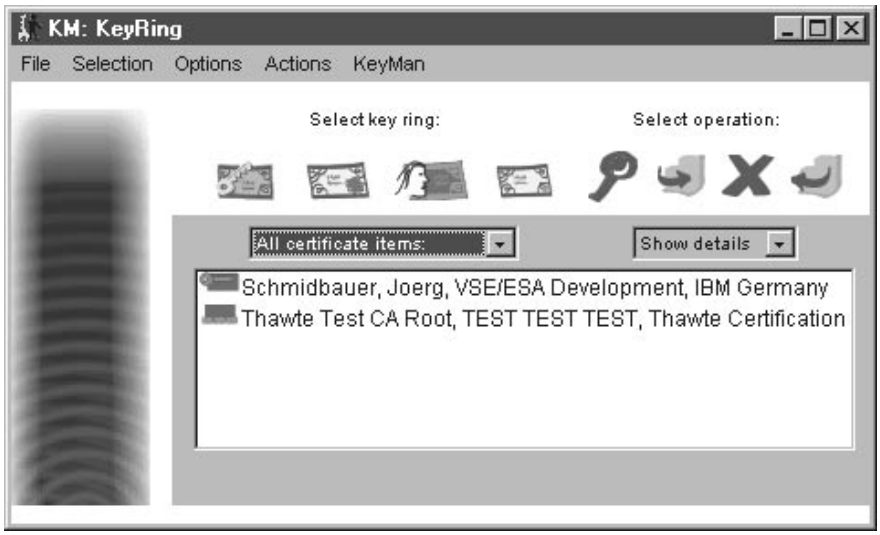


Figure 55. "KM: KeyRing" Window – Import Root Certificate

Step 4: Save Your Client Keyring File

After the root certificate has been imported into your client keyring file, you *must* save the client keyring file by selecting **File** and **Save** from the menu pull-down.

You are also prompted to enter a password to protect this client keyring file.

Step 5: Define Access Rights for VSE Connector Client to Use Host Resources

You can use the service functions provided with client authentication to control the access rights that a VSE Connector Client has to VSE/ESA host resources, using a VSE/ESA User ID that is associated with the client certificate that the client provides. A client-certificates/User-IDs mapping list is managed on the VSE/ESA host, where each client certificate in the mapping list is associated with a VSE/ESA User ID.

To control the access rights for a VSE Connector Client, you must:

1. Paste the client certificate that you saved in a new file (in "Step 2: Generate and Store a Client Certificate" on page 118) into a job on the VSE/ESA host similar to the one shown in Figure 56 on page 122. This job will catalog your client certificate into the VSE Keyring Library on the VSE/ESA host.

```
* $$ JOB JNM=SKCCERT,CLASS=0,DISP=D
// JOB SKCCERT,
// ID USER=<uid>,PWD=<password>
*
* * The following is a sample job that contains a x509v3
* * client certificate in Base64 format.
*
// EXEC LIBR,PARM='MSHP'
A S=CRYPTO.KEYRING
CATALOG SAMPLE.CCERT           DATA=YES REPLACE=YES EOD=/#
-----BEGIN CERTIFICATE-----
MIICLTCCAdcCBJzu1UQwDQYJKoZIhvcNAQEFBQAwgZYxIDAeBgkqhkiG9w0BCQEW
EXZzZWVzYUBkZS5pYm0uY29tMQswCQYDVQQGEwJERTETMBEGA1UEBxMKQm9lYmxp
bmdlbjEUMBIGA1UEChMLSUJNIEdlcm1hbnkxFDASBgNVBAsTC0RldmVsb3BtZW50
MSQwIgYDVQQDExtWU0UvRVNBIERldmVsb3BtZW50IENBIFJPT1QwHhcNMDIxMTE5
MTUzMzM1WhcNMDcxMTE4MTUzMzM1WjCBqDEmMCQGCSqGSIb3DQEJARYXY2xpZW50
QHlvdXIuY29tcGFueS5jb20xCzAJBgNVBAYTAkRFMRcwFQYDVQQHEw5Zb3VyIGxv
Y2FjdGlvbjEaMBgGA1UEChMRWW91ciBvcmdhbml6YXRpb24xFDASBgNVBAsTC0Rl
dmVsb3BtZW50MSYwJAYDVQQDEx1TYW1wbGUgVlNFIENsaWVudCBDZXJ0aWZpY2F0
ZTBcMA0GCSqGSIb3DQEBAQUAA0sAMEgCQQDk7yzQsyauG5yFTI1ouxJ5QAaVdHA5
8zsqq2kyV1M/95l3SACbAw6CHSEBRj1NyXBy8K0L1ZaGvlzYQxL7gs7PAgMBAAEw
DQYJKoZIhvcNAQEFBQADQQB1bcAvxQANxKbVdoWtnROoQBKI9WURLP6Qvvf9NqD1
N6xvdf1pWzjY8A22am+3vXemZx/8RdOtzJOdZJ/kHGNx
-----END CERTIFICATE-----
/#
/*
/&
* $$ EOJ
```

Figure 56. Sample Job to Catalog Client Certificate into VSE Keyring Library

2. Submit the job.
3. Use the *Client Certificates/User-IDs dialog* on the VSE/ESA host to assign a VSE/ESA User ID to this client certificate.
4. Define access rights for the VSE/ESA User ID, giving it access to the VSE/ESA host resources you think are appropriate.

For details of the *Client Certificates/User-IDs dialog* and client-authentication service functions, see Chapter 14, “Service Functions for Client Authentication”, on page 123.

Chapter 14. Service Functions for Client Authentication

If client authentication is required in addition to server authentication, a client certificate is provided by clients to authenticate the client to the server.

Using the service functions for client authentication described in this chapter, you can introduce access checking on client certificates via VSE/ESA User IDs that have been assigned to these client certificates.

The client certificates belong to either CICS clients or VSE Connector Clients. Therefore using client certificates, you can control the access rights from:
- CICS clients to VSE/ESA host resources.
- VSE Connector Clients to VSE/ESA host resources.

A client-certificate/User-ID mapping list can be built and maintained using either the batch function BSSDCERT, or using the Client-Certificates/User-IDs dialog.

This chapter consists of these main sections:

- "Prerequisites"
- "Using the Batch Service Function BSSDCERT"
- "Changing the Defaults (Optional)" on page 125
- "Using the Client-Certificates/User-IDs Dialog" on page 125

Prerequisites

Before you can use the service functions described here, you must have obtained and stored in the VSE Keyring Library on the VSE/ESA host (default CRYPTO.KEYRING) *at least one* client certificate. This is a client certificate in Base64 format, to which you wish to assign a VSE/ESA User ID.

For details of how to obtain and store client certificates for CICS Web Support, refer to the chapter "Configuring CICS to use SSL" in the *CICS Transaction Server for VSE/ESA, Enhancements Guide*.

Using the Batch Service Function BSSDCERT

This section describes how you can use the BSSDCERT service *in batch* to build and to maintain the mapping list of client-certificate/User-ID pairs.

```
EXEC BSSDCERT,PARM='options'

options:
       ADD,<CertMemName>,<Uid>,TRUST|NOTRUST
       CHG,<CertMemName>,<Uid>,TRUST|NOTRUST
       DEL,<CertMemName>
       LST[,<ListMemName>[,I]]
       ACT
       CML,<CmdListMemName>
       STA,<ALL>
```

where:

ADD Adds a new certificate to the client-certificate/User-ID mapping list.

© Copyright IBM Corp. 2000, 2003

Client-Authentication Service Functions

CHG Changes the details of an entry in the client-certificate/User-ID mapping list.

DEL Deletes an entry for a given certificate in the client-certificate/User-ID mapping list.

LST Extracts a readable list from the contents of the client-certificate/User-ID mapping list. The list contains the CertMemName, the assigned userid, the trusted indication, and from the certificate information about the subject. This can be the subject's common name (for example, the certificate owner), and the subject's organization. However, the information will be truncated to fit on the line. The created mapping list can be either displayed on the console, or written to a librarian member in plain text format or IPF format. IPF format is required so that the mapping list can be displayed in the Client-Certificates/User-IDs Dialog (Figure 57 on page 126).

ACT Builds an incore version of the client-certificate/User-ID mapping list. It also activates the list so that it can be used by CICS Web Support (CWS).

CML Specifies a librarian member name that contains a list of BSSDCERT function calls.

STA Shows the status of the client-certificate/User-ID mapping list. If this list is active, the storage size of this list and the number of records, will be displayed. If you specify parameter ALL, the current name settings of the related VSE/ESA library members are also displayed. For further details about name settings, see "Changing the Defaults (Optional)" on page 125.

CertMemName The name of the library member that contains the client certificate. The member suffix is CCERT. The client certificate is a Base64 encoded X.509 certificate as returned from a PKCS #10 certificate request. The data must include the string `'-----BEGIN CERTIFICATE-----'` immediately before the Base64 encoding, and the string `'-----END CERTIFICATE-----'` immediately following it.

Uid The VSE/ESA User ID defined using, for example, the Basic Security Manager (BSM). This User ID will be associated with a client certificate. The client (a CICS client or a VSE Connector Client) will then have the access rights defined for this User ID, during a connection to the VSE/ESA host.

TRUST The specified client certificate is trusted.

NOTRUST The specified client certificate is not trusted.

ListMemName The name of librarian member to which the output of the LST function should be written.

I Specifies that the output of the LST function should be written to in IPF format to the librarian member specified using ListMemName. (IPF format is required so that the mapping list can be displayed in the Client-Certificates/User-IDs Dialog).

CmdListMemName
The name of a librarian member that contains a list BSSDCERT function calls.

Changing the Defaults (Optional)

You might wish to change the library and member-names defaults that are used by BSSDCERT (for example, the name of the VSE Keyring Library). However normally, you should not need to change these defaults.

The SETPARM defaults are shown below. To change any of the defaults, you use the **SETPARM SYSTEM,BSSDC..='value'** statement.

```
Certificate/userid mapping file:
       BSSDCUI='l.s[.mn[.mt]]'
              default: CRYPTO.KEYRING.BSSDCUID.MAPPING

   Client certificates:
       BSSDCCL='l.s'
              default: CRYPTO.KEYRING

   List output in files:
       BSSDCLT='l.s'
              default: IJSYSRS.SYSLIB

   Command list input file:
       BSSDCCS='l.s'
              default: IJSYSRS.SYSLIB
```

For example, to change the *Certificate/Userid mapping file* default to MYCRYPTO.KEYRING, you would enter at the system console:

```
SETPARM SYSTEM,BSSDCUI='MYCRYPTO.KEYRING'
```

Please Note!

If you change the above defaults for *List output in files* or *Command list input file*, you will no longer be able to use the Client-Certificates/User-IDs Dialog.

For details of how to use the SETPARM command, refer to *VSE/ESA System Control Statements*.

Using the Client-Certificates/User-IDs Dialog

The Client-Certificates/User-IDs dialog is provided so that you can more easily manage your client-certificate/User-ID mapping list. It calls the BSSDCERT function, which was described in "Using the Batch Service Function BSSDCERT" on page 123.

Step 1: Starting the Dialog

To start the Client-Certificates/User-IDs dialog, you select *29 Maintain Certificate - User ID List* on the administrator's *Resource Definition Selection* panel (IESEDEF). The panel shown in Figure 57 on page 126 is then displayed, which shows the list of all client-certificate/User-ID pairs defined for your VSE/ESA system.

```
TAS$CERS                CLIENT CERTIFICATES - USER IDS
Enter the required data and press ENTER

OPTIONS: 1 = ADD          2 = CHANGE      5 = DELETE

 OPT          CERTIFICATE                                  USERID  TRUSTED
     COMMON NAME                              MEMBER NAME
 _   John Haeberle, IBM                       KRL00010     JOHNHB    X
 _   Paul Gallagher, IBM                      KRL00012     PAULGL    X
 _   Helmut Hellner, IBM                      KRL00015     HELHELL
 _   Alexander Schoettle, IBM                 KRL00017     ASCHOETT  X
 _   TCPIP4VSE, Connectivity Systems          KRL00018     473337    X
 _   TCPIP5VSE, Connectivity Systems          KRL00019     460341    X
 _   TCPIP6VSE, Connectivity Systems          KRL00022     155287    X
 _   Herbert Nass, IBM                        KRL00024     NASSHER
 _   Anita Stark, IBM                         KRL00026     NETTANI
 _   Elke Schaefer, IBM                       KRL00027     ELKESCHA  X

 LOCATE MEMBER NAME == >
PF1=HELP       2=REDISPLAY    3=END     5=PROCESS    6=ACTIVATE
               8=FORWARD
```

Figure 57. Listing All Client-Certificate/User-ID Pairs

Here is an explanation of the fields displayed in Figure 57:

OPT In this column you can enter either a 1 (to add a certificate to the mapping list), 2 (to change a certificate in the mapping list), or 5 (to delete a certificate from the mapping list). Explained in "Step 2: Selecting an Option".

CERTIFICATE COMMON NAME
Common name contained on the client certificate, and the organization of the certificate owner.

CERTIFICATE MEMBER NAME
Name of the member that contains the client certificate.

USERID VSE/ESA User ID that the administrator has assigned to the client certificate.

TRUSTED If an X is displayed next to a client certificate, then the administrator has decided that this client's User ID can be used for access checking. Otherwise, this client's User ID will not be used for access checking. You can use this field to temporarily deactivate the assignment of a client certificate to a User ID.

The use of the PF5 (Process) and PF6 (Activate) keys are described in "Step 3: Creating the Output Job" on page 127.

Step 2: Selecting an Option

From the panel shown in Figure 57, you can enter in the OPT column one of these options:

1 (ADD) If you enter a **1** (in Figure 57) to add a new client-certificate/User-ID pair, you get the panel shown in Figure 58 on page 127. The system displays default values if you select an empty line. If you selected an already-defined client-certificate/User-ID pair, the dialog uses this client-certificate/User-ID pair and its parameters (for `USER ID` and `TRUSTED`) as a model for the new client-certificate/User-ID pair.

```
TAS$CER             CLIENT CERTIFICATES - USER IDS: ADD

Enter the required data and press ENTER.



MEMBER NAME..... _______     Unique name of the library member
                             that contains the client certificate

USER ID............... UAX    VSE User Id associated to
                             the certificate.

TRUSTED............... 1      The certificate is trusted.
                             Enter 1 for yes and 2 for no.




 PF1=HELP       2=REDISPLAY  3=END
```

Figure 58. Adding a Client-Certificate/User-ID Pair

You must then enter in the field:

Member Name
The name of the member that contains the client certificate. By default, this member will be stored in the VSE Keyring Library.

User ID The VSE/ESA User ID of the client-certificate/User-ID pair to be added.

Trusted Either the value:
1 The client certificate should be trusted.
2 The client certificate should not be trusted.

After completing the fields above, you press Enter to save your changes and return to the Client-Certificates/User-IDs dialog (Figure 57 on page 126).

2 (CHANGE) After entering a **2** next to a client-certificate/User-ID pair, you can then overtype either the User ID or the Trusted parameter of the pair, Press Enter to save your changes.

5 (DELETE) After entering a **5** next to a client-certificate/User-ID pair, you can then press Enter to carry out the deletion.

Step 3: Creating the Output Job

After all your changes have been entered, you can either press PF5 (Process) or PF6 (Activate).

PF5 (Process only)
A job is created (shown in "Step 4: Submitting or Storing the Output Job" on page 128) in which the mapping list of Client-Certificates/User-IDs is to be updated with a list of your changes. These changes might include:

- new pairs whose details you have defined using Option 1.
- changed pairs whose details you have defined using Option 2.
- pairs to be deleted, which you have identified using Option 5.

You can activate these changes at a later time using the PF6 function.

PF6 (Process and Activate)

As for PF5, a job is created in which the mapping list of Client-Certificates/User-IDs is to be updated with a list of your changes. In addition however:

- an incore version of the new mapping list will be built and activated.
- the automatic activation of the new mapping list during all subsequent system-startups, is prepared.

Step 4: Submitting or Storing the Output Job

After completing "Step 3: Creating the Output Job" on page 127, the dialog creates a job with the default name CATCERT. On the *Job Disposition* panel, you can submit the job to batch, file it in your default primary library, or both.

Chapter 15. Mapping VSE/VSAM Data to a Relational Structure

This chapter describes how VSE/VSAM data is mapped to a relational structure, and then the four methods that you can use to do so. It contains these main sections:

- "Introduction to Mapping VSE/VSAM Data"
- "How VSAM Maps Are Structured" on page 130
- "How Maps Are Stored on the VSE/ESA host" on page 130
- "Defining a Map Using RECMAP" on page 132
- "Defining a Map Using the Sample Applet" on page 134
- "Defining a Map Using a Java Application" on page 134
- "Defining a Map Using the VSAM MapTool" on page 140

Introduction to Mapping VSE/VSAM Data

You must map VSAM records to a relational structure, if you wish to access VSAM data using:

- A DB2 Stored Procedure via the VSAM CLI (Call Level Interface), as described in "Using DB2 Stored Procedures to Access VSAM Data" on page 312.
- VSE Java Beans, as described in "Contents of the VSE Java Beans Class Library" on page 155.

In the past, VSAM data was mainly accessed using application programs that understood the internal structure of the VSAM data: the record layout was represented by data structures within the application programs. The disadvantages of using this method are:

- There is no way to share a given record layout with other applications.
- If the record layout changes, the application program must also be changed.
- The data structure is dependent on the programming language (for example COBOL, Assembler, or PL/I).
- Formatted data reports have to be created on the operating-system platform on which the application programs run.

However, the development of *e-business applications* as described in this book, requires:

- A sharing of data representation across operating-system platforms.
- Easy access to data representations from different applications.
- That data representation and data display are independent of operating-system platform and programming language.

VSE/ESA therefore provides you with two methods of mapping your VSAM data so it can be used within e-business applications:

- Creating a *data map* (referred to simply as a *map)* for a given VSAM record or record type. A map splits the VSAM record into columns and their data fields, that have a name, a length, a datatype, and an offset within the record.
- Creating a *data view* (referred to simply as a *view)* that contains a subset of the fields contained within a map. A view always points to a subset of the data fields of a given VSAM map. Therefore, if you change a VSAM map, the views that use this map will also be affected. For example, deleting a map will also

© Copyright IBM Corp. 2000, 2003

delete all views of this map. You can use views to give different user groups different views of the same data (for example, to hide some information from specific users).

How VSAM Maps Are Structured

The mapping definitions have the following *hierarchical* structure:

```
CATALOG       (MY.USER.CATALOG)
 CLUSTER      (MY.DATA.CLUSTER)
  MAP         (Map One)
   COLUMN     (Name,      Ofs=0,Len=10,Type=String,Description=xxxxx)
   COLUMN     (Street,    Ofs=10,Len=20,Type=String,Description=xxxxx)
   ...
   VIEW       (View One)
     COLUMN   (PersonInfo, Ref=Name,Description=xxxxx)
     COLUMN   (Address,    Ref=Street,Description=xxxxx)
     ...
   VIEW       (View Two)
   ...
  MAP         (Map Two)
  ...
```

Figure 59. Hierarchical Structure of VSAM Maps

A data field represents one specific column of a VSAM record. It is always part of a given map or view. It consists of:
- a name (the column name)
- a length
- a datatype
- an offset within the record
- a description

When an application accesses VSAM data using a map, only the *map's* fields are used.

You can also specify filters when accessing VSAM data using a map. For example, you might display only those records where a given column (data field) matches a given filter string. You can use filters in any Java program (Java applications, Java servlets, Java applets, or Enterprise Java Beans).

Note: When you define a map for a cluster, no check is made to see if a cluster exists. Therefore you must take steps to ensure that your mapping definitions are always in synchronization with your VSAM clusters.

How Maps Are Stored on the VSE/ESA host

On the host, the maps of all VSAM clusters are stored in one single VSAM file, `VSE.VSAM.RECORD.MAPPING.DEFS`. *This file is created automatically during the initial installation or FSU of VSE/ESA.*

The short name for this VSAM file is IESMAPD, and the cluster for this file is stored in `VSESP.USER.CATALOG`.

Note: If you have migrated from VSE/ESA 2.5, you do not have to redefine your `VSE.VSAM.RECORD.MAPPING.DEFS` file: your previous mappings can be used as they were originally defined. However from VSE/ESA 2.6 onwards, each field of your previous mappings will now contain a blank description field.

Figure 60 is an example of a job used to define the cluster for file VSE.VSAM.RECORD.MAPPING.DEFS . You can find this example job in the member VSAMDEFS.Z in Library IJSYSRS.SYSLIB.

Notes:

1. You should not normally need to run this job, since it runs automatically during VSE/ESA installation.
2. You should not delete or move this file.

```
* $$ JOB JNM=DEFINE,CLASS=0,DISP=D
// JOB  DEFINE FILE
// EXEC IDCAMS,SIZE=AUTO
         DEFINE CLUSTER (NAME (VSE.VSAM.RECORD.MAPPING.DEFS) -
         RECORDS(2000,1000) -
         SHAREOPTIONS (2) -
         RECORDSIZE (284    284   ) -
         VOLUMES (DOSRES SYSWK1 ) -
         NOREUSE -
         INDEXED -
         FREESPACE (15 7) -
         KEYS (222 0     ) -
         NOCOMPRESSED -
         TO (99366  )) -
         DATA (NAME (VSE.VSAM.RECORD.MAPPING.DEFS.@D@) -
         CONTROLINTERVALSIZE (4096  )) -
         INDEX (NAME (VSE.VSAM.RECORD.MAPPING.DEFS.@I@)) -
         CATALOG (VSESP.USER.CATALOG)
IF LASTCC NE 0 THEN CANCEL JOB
/*
/&
* $$ EOJ
```

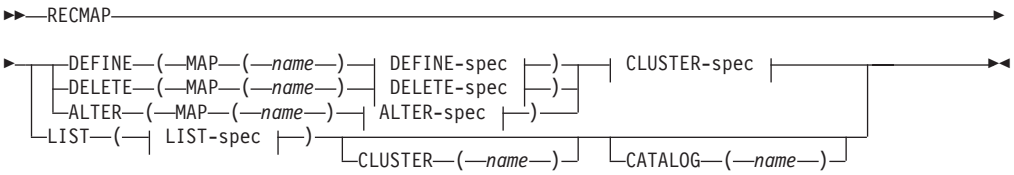
Figure 60. Job To Define the Cluster for VSAM.RECORD.MAPPING.DEFS

Defining a Map Using RECMAP

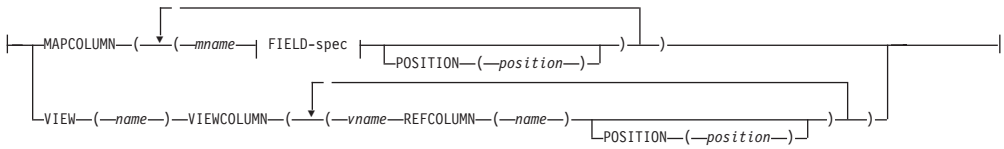
This method of defining a map for a VSAM cluster uses the IDCAMS **RECMAP** command. Using RECMAP, you can also delete, change (alter), or list the contents of a map or view. For a practical example of how RECMAP can be used, see "4. Define the VSAM Data Cluster" on page 219.

The syntax of this command is shown in Figure 61.

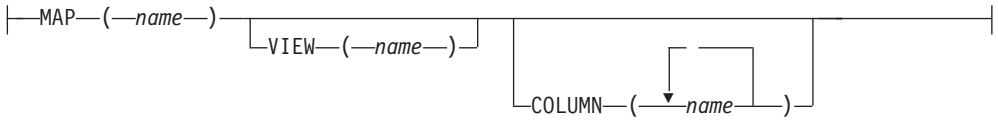
Figure 61. Syntax of RECMAP Command (IDCAMS)



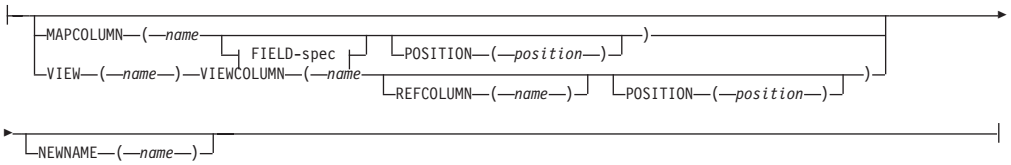
DEFINE-spec:



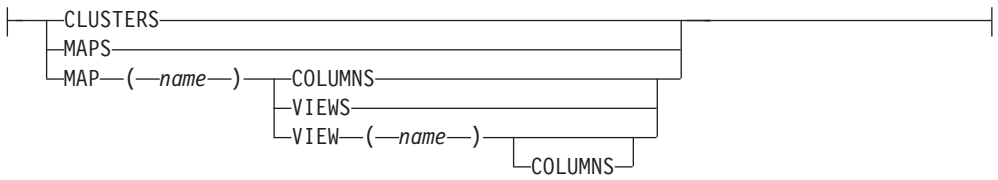
DELETE-spec:



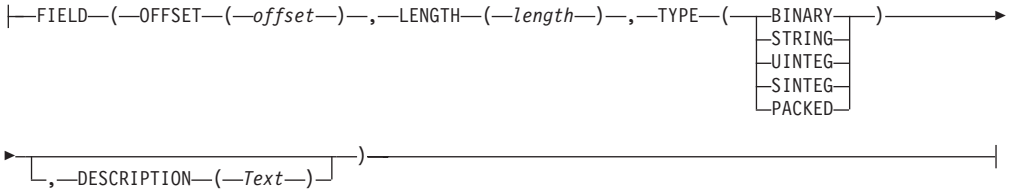
ALTER-spec:



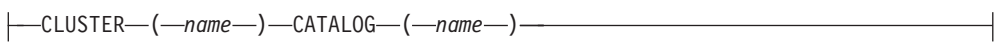
LIST-spec:



FIELD-spec:



CLUSTER-spec:



These are the parameters you can define for use with this command:

name A character string of maximum length 44 characters.

MAP(name) The name of a map you want to define and reference.

MAPCOLUMN
A parameter used when you want to define one or more columns for a map.

mname The name of a column you want to define for a map.

Field(OFFSET(o),LENGTH(l),TYPE(type))
The VSAM record's offset, length and data-type that you want to map. You can write these parameters in any order.

Position(position)
The position number of a column in a map or view.

VIEW(name) The name of a view that you want to define. This parameter requires that you specify **MAP(name)**.

VIEWCOLUMN
The name of a column contained in a view (which can be different from this column as specified in **MAPCOLUMN**).

REFCOLUMN(name)
The name of a pointer to the corresponding **MAPCOLUMN**.

COLUMN(name)
The name of a column that will be deleted from a map or view, when you specify **VIEW(name)**.

NEWNAME(name)
The new name for a column/view/map (which depends upon the previous parameter that was specified).

CLUSTERS You use this parameter to list all maps, views and columns belonging to all clusters.

MAPS You use this parameter to list all columns belonging to all maps.

COLUMNS You use this parameter to list all columns belonging to a specific map.

VIEWS You use this parameter to list all columns belonging to all views.

View(name) Columns
You use this parameter to list all columns belonging to a specific view.

DESCRIPTION(text)
Description of the field.

CLUSTER(name)
The name of a VSAM cluster (maximum 44 characters length) that you want to reference.

CATALOG(name)
The name of a VSAM catalog (maximum 44 characters length) that you want to reference.

Defining a Map Using the Sample Applet

This method of defining a map for a VSAM cluster uses *an applet* that runs on a *Web browser.*

A sample applet for used for defining maps is contained in **VsamMappingApplet.html** of the VSE/ESA e-business Connectors online documentation, and is described in "Running the Sample Data-Mapping Applet" on page 207.

Defining a Map Using a Java Application

This method of defining a map for a VSAM cluster uses a *Java application* that runs on *a workstation.*

The example described here is for a simple Java application that creates a map for a given VSAM cluster, and also defines two different views. The example is taken from the *VSE/ESA e-business Connectors* online documentation.

If you require the online version of this example, refer to *VsamDataMapping.html*. The source code is contained in Java source file *VsamDataMapping.java*, which is located in the **com.ibm.vse.samples** directory, that is positioned below the samples directory of your connector client installation (described in "Installing the VSE Connector Client" on page 25).

The following steps describe how you code a Java application that defines a map for a VSAM cluster.

1. *Create a Map for the VSAM Cluster:*

   To create a map, you must first instantiate a local object of the *VSEVsamMap* class. The creation of the map definition on the host also requires a call to the *create()* method.

   To use an existing map, you must first instantiate a local object of class *VSEVsamMap* and then use the *isExistent()* method to check if this map exists on the host.

   Note: Be aware that distributed objects are being used. Therefore, when local objects of class *VSEVsamMap* and *VSEVsamView* are created, this does not automatically imply that the related definitions are created on the host.

   Create a local map object and check whether it exists on the host. If it does not exist, create it on the host using the *create()* method. Put the code into try/catch

blocks, since the *create()* and *isExistent()* methods can produce exception errors. Here is an example of how to create a local VSAM map object:

```
...
public static void main(String argv[ ]) throws IOException
{
   VSESystem system;
   VSEVsamMap map;
   VSEVsamView employeeView=null, managerView=null;
   VSEVsamField dataField;
   ...
   // Create VSESystem and connect
   ...

   // Create map ...
   String catalogID = "VSESP.USER.CATALOG";
   String clusterID = "VSAM.DISPLAY.DEMO.CLUSTER";
   try
   {
     System.out.println("Creating map MYMAP ...");
     map = new VSEVsamMap(system, catalogID, clusterID, "MYMAP");
     if (!(map.isExistent()))
     {
        System.out.println("  Map does not exist on host. Create it ...");
        map.create();
     }
     else
        System.out.println("  Map already exists on host. Continue ...");
   }
   catch (Exception e)
   {
      System.out.println("  Exception when creating map: ");
      System.out.println(e);
      return;
   }
   ... (Continued)
```

Figure 62. Example of Creating a Local VSAM Map Object

2. *Create Data Fields for the Map*

   Data fields of a given view of a map are simply references to the fields of the map. But the same field can have different names in different views. Deleting a field of a map, for example, will also make this field unavailable for all views. Deleting the whole map will also delete all views of this map. However, your local view objects *are not notified* that their map has been deleted. When the next access is made to a field that belongs to a view, an exception will be generated.

   Now you must add fields to the map, which describe the record's internal structure. Note that these field names must be in *uppercase* (otherwise they will be translated to uppercase when sending the request to the host). Whereas creating a *VSEVsamField* object is a local operation only, the *addField()* method sends this request to the host. Here is an example of how to create data fields for a map:

```
System.out.println("   ");
System.out.println("Adding fields to the map...");
String[ ] fields = {"LAST NAME", "FIRST NAME", "DEPARTMENT", "AGE"};
int[ ] types = {VSEVsamMap.TYPE_STRING, VSEVsamMap.TYPE_STRING,
               VSEVsamMap.TYPE_UNSIGNED, VSEVsamMap.TYPE_UNSIGNED};
int[ ] lengths = {20, 10, 4, 2};
int[ ] offsets = {0, 20, 30, 34};

for (int i=0;i<fields.length;i++)
{
   try
   {
      dataField = new VSEVsamField(system, fields[i], types[i],
                                   lengths[i], offsets[i]);
      map.addField(dataField);
   }
   catch (Exception e)
   {
      System.out.println("Exception when adding " + fields[i] + "
                                      to the map: ");
   }
}
... (Continued)
```

Figure 63. Example of Creating Data Fields for a Map

3. *Display the Properties of the Map*

   Here is an example of how to display some of the properties of the map you have created:

```
...
/* Now display some properties of the map */
System.out.println("Map properties");
System.out.println("  Map name         : " + map.getName());
System.out.println("  Catalog name     : " + map.getCatalog());
System.out.println("  Cluster name     : " + map.getCluster());
System.out.println("  System name      : " + map.getVSESystem());
System.out.println("  Number of fields : " + new Integer(
                                  (map.getNoOfFields())).toString());
for (int i=0;i<map.getNoOfFields();i++)
{
   System.out.println("  Field " + new Integer(i).toString() + " : " +
                                                  map.getFieldName(i));
   System.out.println("   type   = " + new Integer(
                                    (map.getFieldType(i))).toString());
   System.out.println("   length = " + new Integer(
                                   (map.getFieldLength(i))).toString());
   System.out.println("   offset = " + new Integer(
                                   (map.getFieldOffset(i))).toString());
}
... (Continued)
```

Figure 64. Example of Displaying the Properties of a Map

4. *Create the Employee View for the Map*

   The example provided in the VSE/ESA e-business Connectors online documentation creates two different views within this map: a manager's view (not shown here), and an employee's view. Again, creating the view object is only local, but the *create()* method creates the view on the host.

```
...
System.out.println("   ");
System.out.println("Creating employee's view on the map ...");
try
{
   employeeView = new VSEVsamView(system, catalogID, clusterID,
                                  "MYMAP", "EMPLOYEEVIEW");
   if (!(employeeView.isExistent()))
   {
      System.out.println("Employee view does not exist. Create it ...");
      employeeView.create();
   }
   else
      System.out.println("Employee already exists. Continue ...");
}
catch (Exception e)
{
   System.out.println("Exception when creating employeeView: ");
   return;
}
...
...
```

Figure 65. Example of Creating a View for a Map

5. *Add Data Fields to the Employee View*

   In the example we now add fields to the employee view. This uses the same method as adding fields to a map. We also decide that employees are allowed to see only three fields of a VSAM record: the last name, the first name, and the department number. Here is the code for such a scenario:

```
String[ ] empFields = {"LAST NAME", "FIRST NAME", "DEPARTMENT"};
System.out.println("Adding fields to employeeView ...");
for (int i=0;i<emp.Fields.length;i++)
{
     try
     {
        employeeView.addField(empFields[i]);
     }
     catch (Exception e)
     {
        System.out.println("Exception when adding " + empFields[i] + "
                                          to employeeView: ");
     }
}
  ... (Continued)
```

Figure 66. Example of Adding Data Fields to a View

6. *Display Properties of the Employee View*

   Here is an example of how to display the properties of the employee view:

```
/* Now display some properties of the employeeView */
   System.out.println("employeeView properties");
   System.out.println("  View name        : " + employeeView.getName());
   System.out.println("  Catalog name     : " + employeeView.getCatalog());
   System.out.println("  Cluster name     : " + employeeView.getCluster());
   System.out.println("  System name      : " + employeeView.getVSESystem());
   System.out.println("  Number of fields : " + new Integer(
                                          (employeeView.getNoOfFields())).toString());
   for (int i=0;i<employeeView.getNoOfFields();i++)
   {
   System.out.println("  Field " + new Integer(i).toString() + " : " +
                                                    employeeView.getFieldName(i));
      System.out.println("   type   = " + new
                                Integer((employeeView.getFieldType(i))).toString());
      System.out.println("   length = " + new
                               Integer((employeeView.getFieldLength(i))).toString());
      System.out.println("   offset = " + new
                               Integer((employeeView.getFieldOffset(i))).toString());
   }
   ... (Continued)
```

Figure 67. Example of Displaying the Properties of a View

7. *Delete the Map*

   Here the map is deleted, together with its views and fields:

```
/* Now delete the map including all views and fields */
   try
   {
      System.out.println("Deleting map MYMAP ...");
      map.delete();
   }
   catch (Exception e)
   {
      System.out.println("Exception when deleting map:");
      System.out.println(e);
   }
   ...
}
```

Figure 68. Example of How to Delete a Map

Java Console Output Produced By the Sample

The following output was produced on the Java console when running this sample from the Web browser (using file *RunExamples.html*) on the IBM test system.

```
C:\vsecon\samples>java  com.ibm.vse.samples.VsamDataMapping
Please enter your VSE IP address:
9.164.155.95
Please enter your VSE user ID:
sysa
Please enter password:
*****  (password will display in clear)
Creating map MYMAP ...
  Map does not exist on host. Create it ...
```

```
Adding fields to the map...
Map properties
  Map name         : MYMAP
  Catalog name     : VSESP.USER.CATALOG
  Cluster name     : VSAM.DISPLAY.DEMO.CLUSTER
  System name      : 9.164.155.95
  Number of fields : 4
  Field 0 : LAST NAME
   type   = 2
   length = 20
   offset = 0
  Field 1 : FIRST NAME
   type   = 2
   length = 10
   offset = 20
  Field 2 : DEPARTMENT
   type   = 3
   length = 4
   offset = 30
  Field 3 : AGE
   type   = 3
   length = 2
   offset = 34

Creating employee's view on the map ...
Employee view does not exist. Create it ...
Adding fields to employeeView ...
employeeView properties
  View name        : EMPLOYEEVIEW
  Catalog name     : VSESP.USER.CATALOG
  Cluster name     : VSAM.DISPLAY.DEMO.CLUSTER
  System name      : 9.164.155.95
  Number of fields : 3
  Field 0 : LAST NAME
   type   = 2
   length = 20
   offset = 0
  Field 1 : FIRST NAME
   type   = 2
   length = 10
   offset = 20
  Field 2 : DEPARTMENT
   type   = 3
   length = 4
   offset = 30

Creating manager's view on the map ...
Manager view does not exist. Create it ...
Adding fields to managerView ...
Exception when adding MONTHLY SALARY to managerView:
managerView properties
  View name        : MANAGERVIEW
  Catalog name     : VSESP.USER.CATALOG
  Cluster name     : VSAM.DISPLAY.DEMO.CLUSTER
  System name      : 9.164.155.95
  Number of fields : 4
  Field 0 : LAST NAME
   type   = 2
   length = 20
   offset = 0
  Field 1 : FIRST NAME
   type   = 2
   length = 10
   offset = 20
  Field 2 : DEPARTMENT
   type   = 3
   length = 4
```

```
   offset = 30
  Field 3 : AGE
   type   = 3
   length = 2
   offset = 34
Deleting map MYMAP ...
Finished.

C:\vsecon\samples>pause
Press any key to continue . . .
```

After you have defined one or more maps for a VSAM cluster (which describe the internal structure of VSAM records), you can display the mapped VSAM data using:

- A DB2 Stored Procedure via the VSAM CLI Interface (described in "Using DB2 Stored Procedures to Access VSAM Data" on page 312)
- The VSE Java Beans (described in "Using VSE Java Beans for Accessing VSAM Data" on page 164).

Defining a Map Using the VSAM MapTool

This method of defining a map for a VSAM cluster uses the *VSAM MapTool*, which creates a map by parsing a COBOL copybook.

In addition, you can use the VSAM MapTool to:

- Import (receive) a specified map from a specified VSE/ESA system.
- Export a map to a VSE/ESA system (that is, send it to VSE).
- Import a map from an XML file.
- Export a map to an XML file.
- Create a Java source file from a specified map. The Java program can obtain all records from the related VSAM file, by using this map.

Figure 69 shows window that the VSAM MapTool displays:

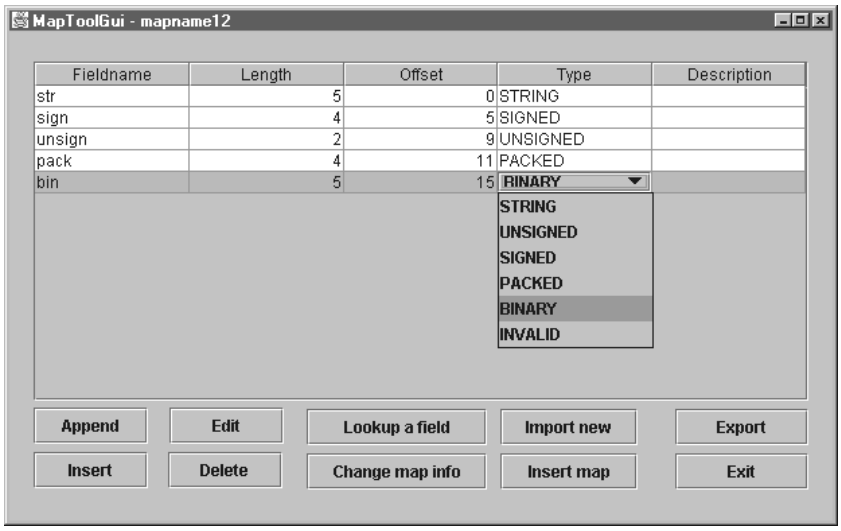


Figure 69. Example of a VSAM MapTool Window

To install the VSAM MapTool you must:

1. Download the file **maptool.zip** from this internet address:

```
http://www.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/maptool.htm
```

2. Run the installation procedure described on the Web site whose address is given above.

The installation package includes many examples of how to use the VSAM MapTool.

Part 3. Programming

Part 3 contains these chapters:

- Chapter 16, “Migrating Your Programs”, on page 145
- Chapter 17, “Using VSE Java Beans to Implement Java Programs”, on page 153
- Chapter 18, “Using JDBC to Access VSAM Data”, on page 187
- Chapter 19, “Using Java Applets to Access Data”, on page 201
- Chapter 20, “Using Java Servlets to Access Data”, on page 243
- Chapter 21, “Using Java Server Pages to Access Data”, on page 263
- Chapter 22, “Using EJBs to Represent Data”, on page 267
- Chapter 23, “Extending the Java-Based Connector”, on page 287
- Chapter 24, “Using the DB2-Based Connector to Access Data”, on page 309
- Chapter 25, “Using SOAP for Inter-Program Communication”, on page 331
- Chapter 26, “Using the VSE Script Connector for Non-Java Access”, on page 349

© Copyright IBM Corp. 2000, 2003

Chapter 16. Migrating Your Programs

This chapter describes the actions you must take to migrate your programs so that they can take advantage of the latest support. It contains these main sections:

- "Migrating from CCF to CCI"
- "Migrating to Secure Connections Using SSL" on page 147
- "Migrating to VSAM-Access Via JDBC" on page 149
- "Migrating Your Applets to JDK 1.3 or Later" on page 151
- "Using the New Methods in VSE Java Beans" on page 152
- "Migrating Servlets and EJBs" on page 152

Migrating from CCF to CCI

From VSE/ESA 2.6 onwards, IBM's Common Connector Framework (CCF) that was previously used with VSE/ESA 2.5, was replaced by the J2EE *Common Client Interface* (CCI). This change is only relevant for 3-tier applications, in which (for example) servlets and EJBs are used.

These are the changes you must make to your code, in order to use CCI instead of CCF:

```
/* Import CCF classes */
import com.ibm.connector.*;
import com.ibm.connector.internal.*;

.
.
.
/* Create VSESystem and connection */
try {
  spec = new VSEConnectionSpec(InetAddress.getByName(ipAddr),
                               2893,userid,password);
}
catch (Exception e)
{

.
.
.
  return;
}

spec.setReapTime(5);
spec.setMaxConnections(5);
spec.setMinConnections(0);
spec.setUnusedTimeout(10000);
spec.setLogonMode(true);

/* Create VSE system instance with this connection */
system = new VSESystem(spec);

if (system.getConnectionManager() != connmgr)
   system.setConnectionManager(connmgr);

.
.
.
```

Figure 70. Code Containing CCF Statements (VSE/ESA 2.5)

© Copyright IBM Corp. 2000, 2003

```
/* Import CCI classes */
import javax.resource.*;


.
.
.
/* Create VSESystem and connection */
try {
  spec = new VSEConnectionSpec(InetAddress.getByName(ipAddr),
                               2893,userid,password);
}
catch (Exception e)
{

.
.
.
  return;
}

spec.setMaxConnections(5);
spec.setLogonMode(true);

/* Create VSE system instance with this connection  */
system = new VSESystem(spec);

/* Note: The following statements are only required */
/*       if you use a Web Application Server        */
/*       (for example IBM WebSphere)                */

Context ctx = new InitialContext();
spec.setJNDIContext(ctx);
spec.setJNDIName("eis/VSEConnector");

.
.
.
```

Note: String `eis/VSEConnector` (above) is an example of a JNDI Binding Path, which is a property of a WebSphere Connection Factory. For further details, refer to the VSE Connector Client online documentation (see "Using the Online Documentation Options" on page 28).

Figure 71. Equivalent Code Containing CCI Statements (VSE/ESA 2.6)

The CCI-related classes are stored in file **cci.jar**. Your local classpath *must* contain this file.

Migrating to Secure Connections Using SSL

From VSE/ESA 2.6 onwards, Secure Sockets Layer (SSL) security is supported for connecting between VSE Connector Clients and the VSE Connector Server running on the VSE/ESA host. For details, see "Configuring the VSE Connector Server for Server Authentication" on page 105 and "Configuring the VSE Connector Client for Server Authentication" on page 108.

The SSL-related Java classes are contained in file **ibmjsse.jar**, which must be contained in your local classpath.

These are the changes you must make to your code, in order to include SSL connections:

```
:
 try {
    spec = new VSEConnectionSpec(InetAddress.getByName(ipAddr),
                                    2893,userID,password);
 }
 catch (UnknownHostException e)
 {
    System.out.println("Unknown host : " + e);
    return;
 }


:

 /* Create VSE system instance with this connection */
 system = new VSESystem(spec);

 /* Connect to host */
 system.connect();

:
```

Figure 72. Code Without SSL Support (VSE/ESA 2.5)

```
...
try {
   spec = new VSEConnectionSpec(
                InetAddress.getByName(ipAddr),
                2893,userID,password);
}
catch (UnknownHostException e) { ... }

/* Specify secure SSL connection */
spec.setSSL(true);

/* Specify SSL properties */
sslProps = new Properties();
sslProps.put("SSLVERSION", "SSL");
sslProps.put("CIPHERSUITES",
     "SSL_RSA_WITH_NULL_MD5," +
     "SSL_RSA_WITH_NULL_SHA," +
     "SSL_RSA_EXPORT_WITH_DES40_CBC_SHA," +
     "SSL_RSA_WITH_DES_CBC_SHA," +
     "SSL_RSA_WITH_3DES_EDE_CBC_SHA");
sslProps.put("KEYRINGFILE", "c:\\vsecon\\KeyRing.pfx");
sslProps.put("KEYRINGPWD", "ssltest");
spec.setSSLProperties(sslProps);

/* Create VSE system instance with this connection */
system = new VSESystem(spec);

/* Connect to host */
system.connect();

...
```

Figure 73. Equivalent Code Containing SSL Support (VSE/ESA 2.6)

Migrating to VSAM-Access Via JDBC

From VSE/ESA 2.6 onwards, JDBC support was introduced for the Java-based connector. This included a JDBC driver with a limited set of supported SQL constructs. For details, see Chapter 18, “Using JDBC to Access VSAM Data”, on page 187.

These are the changes you must make to your code, in order to include JDBC for VSAM access:

```
:
 /* Create VSE system instance */
 system = new VSESystem(spec);
 vsam = system.getVSEVsam();

 /* Create a listener that gets notified when objects */
 /* are retrieved from the host */
 rl = new RecordListener();
 vsam.addVSEResourceListener(rl);

 /* Get VSAM records from host using the given map */
 map = new VSEVsamMap(system, catName, fileName, mapName);
 cluster = new VSEVsamCluster(system, catName, fileName);
 cluster.addVSEResourceListener(rl);
 cluster.selectRecords(map);
 cluster.removeVSEResourceListener(rl);

 /* Get vector containing all records from listener */
 vRecords = rl.getRecords();

 /* Display records ... */
 int numMapFields = map.getNoOfFields();
 for (int k=0;k<vRecords.size();k++)
 {
    record = (VSEVsamRecord)(vRecords.elementAt(k));
    for (int i=0;i<numMapFields;i++)
    {
       try {
          if (map.isFieldPartOfPrimaryKey(i))
             System.out.println(map.getFieldName(i) +
                " (Key) : " + record.getField(i).toString());
          else
             System.out.println(map.getFieldName(i) +
                " : " + record.getField(i).toString());
       }
       catch (Exception e)
       {

:

       }
    }
 }

:
```

Figure 74. Code Using VSE Java Beans to Access VSAM Data (VSE/ESA 2.5)

```
 /* Setup JDBC driver */

...
try {
   /* Create an instance of the JDBC driver */
   jdbcDriver = (java.sql.Driver) Class.forName(
        "com.ibm.vse.jdbc.VsamJdbcDriver").newInstance();

   // Build the URL to use to connect
   String url = "jdbc:vsam:"+vseHostName;

   // Assign properties for the driver
   java.util.Properties prop = new java.util.Properties();
   prop.put("port", vsePort);
   prop.put("user", userID);
   prop.put("password", password);

   // Connect to the driver
   jdbcCon = DriverManager.getConnection(url, prop);

   // Set the catalog.
   jdbcCon.setCatalog(vsamCatalog);
}
catch (Throwable t)
{

...
}

/* Display a list of database rows */
try
{

   // get a statement
   java.sql.Statement stmt = jdbcCon.createStatement();

   // execute the query
   java.sql.ResultSet rs = stmt.executeQuery("SELECT * FROM "+
          vsamCatalog+"\\"+flightsCluster+"\\"+flightsMapName);

   // Walk through the results
   while (rs.next())
   {
      // get the values
      flightNumber = rs.getInt("FLIGHT_NUMBER");
      start        = rs.getString("START");
      destination  = rs.getString("DESTINATION");
      departure    = rs.getString("DEPARTURE");
      arrival      = rs.getString("ARRIVAL");
      price        = rs.getInt("PRICE");
      airline      = rs.getString("AIRLINE");
   }
   rs.close();
   stmt.close();
}
catch (SQLException t)
{

...
}
```

Figure 75. Equivalent Code Using JDBC to Access VSAM Data (VSE/ESA 2.6)

Migrating Your Applets to JDK 1.3 or Later

The behavior of applets was changed with JDK 1.3. For example, by default the *appletviewer* no longer allowed applets to connect to other platforms.

Because of the new security concept of the Java 2 platform, a policy must be used so that the *appletviewer* can allow default security restrictions. To do so, you use the "policy tool" that is supplied with the JDK or JRE.

Below are the steps you must carry out in order to set up a policy file. For further details, refer to the online documentation provided with the VSE Connector Client (see "Using the Online Documentation Options" on page 28 for details).

1. Write your applet and ensure that it compiles correctly. Create a **.jar** file containing the applet class, and any other classes that are required.
2. Create an HTML file from which the applet is called.
3. Test the HTML file using the applet viewer, as follows:
   a. Run `policytool` from the command prompt (DOS Console).
   b. Follow the instructions for setting up your policy file, as provided by the tool. For details of the policy tool, refer to the documentation provided with the Sun Microsystems Inc. Java Tutorial.
   c. Save the policy file, and then exit from the policy tool.
   d. Run the appletviewer for the created policy file, by typing:

      ```
      appletviewer -J-Djava.security.policy=<policyFile> <URL>
      ```

   After you have tested the policy file using the appletviewer, you can test it using your Web browser.

   Because differences exist between various Web browsers and Java APIs, you must modify the standard HTML file that you created above, so that the file is compatible with all Web browsers and Java APIs. To do so, use the Java Plug-in HTML Converter, which you can download from Sun Microsystems at **`http://java.sun.com/products/plugin`**.
4. Go to the directory where you downloaded and unzipped the HTML Converter. In this directory there should be a file called **htmlconv1_3_0_01.jar**. You execute this file by typing:

   ```
   java -jar htmlconv1_3_0_01.jar
   ```

   After being started, you can specify the name of the file or directory that you want to convert.

Using the New Methods in VSE Java Beans

From VSE/ESA 2.6 onwards, a number of new methods were introduced in the IBM-supplied VSE Java Beans classes. Here are the VSE Java Beans classes, and most important new methods:

VSEVsamCluster
getFirstRecord(), getLastRecord()

VSEVsamRecord
getPrevious(), getNext()

VSEPowerQueue
isInCreationQueue()

VSEVsamField
A new field "Description" has a textual description for this data field. It is compatible with VSE/ESA 2.5 map and view fields (that is, old fields simply contain an empty string as their description).

Note: If have migrated from VSE/ESA 2.5, you must *not* redefine your `VSE.VSAM.RECORD.MAPPING.DEFS` file: your previous mappings can be used as they were originally defined. However from VSE/ESA 2.6 onwards, each field of your previous mappings will now contain a blank description field.

Migrating Servlets and EJBs

For details of how to migrate your servlets and EJBs to the latest supported version of WebSphere, refer to the VSE Connector Client online documentation (see "Using the Online Documentation Options" on page 28).

Chapter 17. Using VSE Java Beans to Implement Java Programs

This chapter describes how you use the VSE Connector Client's *VSE Java Beans* within 2-tier and 3-tier environments. It contains these main sections:

- "Where VSE Java Beans Are Installed and Used"
- "How JavaBeans and EJBs Compare to VSE Java Beans" on page 154
- "Contents of the VSE Java Beans Class Library" on page 155
- "Example of a Javadoc for a VSE Java Bean" on page 158
- "Using the Callback Mechanism of VSE Java Beans" on page 159
- "Using VSE Java Beans to Connect to a VSE/ESA host" on page 162
- "Using VSE Java Beans for Accessing VSAM Data" on page 164
- "Using VSE Java Beans for Accessing DL/I Data" on page 167
- "Using VSE Java Beans for Accessing VSE/POWER Data" on page 172
- "Using VSE Java Beans for Submitting Jobs" on page 173
- "Using VSE Java Beans for Accessing Librarian Data" on page 175
- "Using VSE Java Beans for Accessing VSE/ICCF Data" on page 177
- "Using VSE Java Beans for Accessing the Operator Console" on page 180
- "Using the VSE Navigator Application" on page 182

Where VSE Java Beans Are Installed and Used

All VSE Java Beans classes are contained in one Java archive **VSEConnector.jar**, which are included in the VSE Connector Client.

You will usually install the VSE Java Beans in this way:

1. Install the VSE Connector Client on the workstations where you develop your Java applications.
2. When these Java applications are ready to be implemented in your production systems, copy file **VSEConnector.jar** to either the Web clients of 2-tier environments, or middle-tier of 3-tier environments.

The use of VSE Java Beans is described for:

- applets in 2-tier environments, in Figure 100 on page 202.
- applets in 3-tier environments, in Figure 101 on page 204.
- servlets in 3-tier environments, in Figure 115 on page 243.
- JSPs in 3-tier environments, in Figure 130 on page 264.
- EJBs in 3-tier environments, in Figure 135 on page 273.

© Copyright IBM Corp. 2000, 2003

How JavaBeans and EJBs Compare to VSE Java Beans

You should distinguish between *JavaBeans*, *Enterprise Java Beans (EJBs)*, and *VSE Java Beans*:

- *JavaBeans* are usually *visual* components, such as push buttons, sliders, and list boxes. *VisualAge for Java* allows you to assemble dialogs using these components without any programming effort. Refer to **http://www.javasoft.com/beans/docs** for more information.
- Enterprise Java Beans (EJBs) are *distributed* Java Beans. The term *distributed* is used because:
  - One part of an EJB runs inside the JVM of a Web application server (such as IBM's WebSphere Application Server).
  - One part runs (typically) inside the JVM of a Web browser.

  An EJB represents either one data row in a database (an *entity bean*), or a connection to a remote database (a *session bean*). Usually, entity beans and session beans are used together, to allow data to be represented and accessed in a standardized way, and in heterogeneous environments that contain both relational and non-relational data. For further information about EJBs, refer to Chapter 22, "Using EJBs to Represent Data", on page 267.
- *The VSE Java Beans* supplied with the VSE/ESA e-business Connectors are not visual Java Beans: they conform to the specifications for Java Beans, but are not visual components. Instead, they *represent* VSE/ESA-based objects such as:
  - File systems (VSE/Librarian, VSE/POWER, VSE/ICCF, VSE/VSAM).
  - System components (such as the operator console).
  - Data objects (such as VSE libraries, POWER queue entries, and VSAM catalogs).

Contents of the VSE Java Beans Class Library

The following table shows the classes of the VSE Java Beans (package *com.ibm.vse.connector*). The Java interfaces are shown in italics.

Table 4. Contents of the VSE Java Beans Class Library

Class	Description
VSECertificateEvent	This class is a event that is specific to an SSL Certificate. It contains the VSEConnectionSpec as source of this event and information about the certificate
VSECertificateListener	This interface must be implemented by any VSECertificateListener. You can register a VSECertificateListener using the method addVSECertificateListener of the VSEConnectionSpec bean. This Listener is used to notify about certificates of a SSL connection.
VSEConnectionManager	Maintains connections to remote VSE systems.
VSEConnectionSpec	This class represents the specification of a connection from your workstation to a VSE/ESA host. It implements interface ConnectionSpec, which is part of the IBM Common Connector Framework (CCF). CCF maintains a pool of connections that can be reused by different Java programs. This is important especially when writing WebSphere based Java programs, such as servlets, or Enterprise Java Beans. In this case, short living programs, such as servlets, can reuse existing connections to the VSE/ESA host.
VSEConnectorTrace	This interface must be implemented by any Trace-Class. During startup of the VSE Connector Beans the system tries to load a class called Trace.class in the default package. This class is loaded using Class.forName(). Normally there is no Trace.class available. In this case no trace messages are written. Users can implement its own Trace.class by implementing the interface VSEConnectorTrace. The VSE Connector Beans will call the method writeTrace for each line that is to be written. The Trace class will typically write the trace text to System.out or to a file.
VSEConsole	This class represents the VSE operator console. It allows you to issue console commands, obtain the message output, and get message explanation for a given message number.
VSEConsoleExplanation	This class represents a textual explanation of a given console message. It provides methods to get the message lines and the number of lines.
VSEConsoleMessage	This class represents a console message. It provides methods to get properties, such as message number, color, attributes, and so on.
VSEDli	This class represents the DL/I subsystem on VSE/ESA. It provides methods to get a list of PSBs.
VSEDliPsb	This class represents a DL/I PSB with its corresponding PSB name. It provides methods to schedule or terminate a PSB, to take a checkpoint or to do a rollback. It also provides a list of PCBs.
VSEDliPcb	This class represents a DL/I PCB witch can be used to execute DL/I requests like GN, GNP, GU, GHU, GHN, GHNP, DLET, ISRT, REPL.
VSEIccf	This class represents the VSE/ICCF component of a VSESystem. It provides *read only access* to the VSE/ICCF libraries and members. It provides methods to get a list of VSE/ICCF libraries, search for members, and get properties.
VSEIccfLibrary	This class represents an VSE/ICCF library. It provides methods to get a list of it's members, search for members, and so on.
VSEIccfMember	This class represents an ICCF member. It provides methods to copy and download a member, and get properties.
VSELibrarian	This class represents the Librarian of a VSE System. It is required in order to get access to the VSE library system. It provides methods to list libraries and search for members. The list of libraries is specified in the VSE Connector Server configuration file IESLIBDF.

Table 4. Contents of the VSE Java Beans Class Library (continued)

Class	Description
VSELibrary	This class represents a VSE library. It provides methods to access its sub-libraries, search for members, get library properties, and so on. See also VSESubLibrary.
VSELibraryExtent	This class represents extent information of a VSELibrary.
VSELibraryMember	This class represents a VSE library member. It provides methods to copy, delete, download, upload a member instance, get and set properties, and so on.
VSEMessage	This class allows a message from another user to be received.
VSEPlugin	This abstract class is the base class of all user written plugin beans. It allows you to extend the VSE Java Beans class library by your own beans, by implementing the basic functionality of plugins. For a description of how to write a plugin for the VSE Java Beans, refer to "Implementing a Client Plugin" on page 303.
VSEPower	This class represents the VSE/POWER component of a given VSESystem. It provides methods to access the POWER queues, submit jobs, get the related job output, search for queue entries, get and set properties, and so on.
VSEPowerEntry	This class represents one VSE/POWER queue entry. It provides methods to copy, delete, release, download a queue entry, get and set properties, upload a local file into a queue entry, and so on.
VSEPowerQueue	This class represents an instance of a VSE/POWER queue. This can be the reader, list, punch, or transmit queue. It provides methods to get a list of the queue entries, search for queue entries, get queue properties, and so on.
VSEResource	This abstract class is the base class of all VSEResource beans. It implements the basic functionality that must be present for each resource.
VSEResourceEvent	This class represents an event specific to a VSEResource. It contains the source of this event and optionally event related data. It is used, for example, when implementing a VSEResourceListener.
VSEResourceListener	This interface is used to implement callback routines that allow to synchronize sending actions to the host and receiving data objects. A resource listener is called from the low level functions of the VSE Java Beans to notify about received data objects. See online documentation (VSEConnectors.html) for many samples of resource listener implementations.
VSESubLibrary	This class represents a sub-library of a given VSELibrary. It provides methods to create and delete an instance of this class, get a list of members in this sub-library, get and set properties, search for members, and so on. See also VSELibrary.
VSESystem	This class represents a VSE/ESA host. An instance of this class is needed to connect to the host and get access to the VSE file systems and functionality.
VSEUser	This class represents a user on a given VSE system. It provides methods to check if this user is currently active, get properties and so on.
VSEVsam	This class represents the VSAM component of a given VSESystem. It provides access to the VSAM catalogs and clusters.
VSEVsamCatalog	This class represents a VSAM catalog. It provides methods to get a list of clusters in this catalog, get and set properties.
VSEVsamCluster	This class represents a VSAM file. This can be any cluster, except VRDS. It provides methods to get a list of data maps (see VSEVsamMap), get and set properties, select data records from the cluster, and so on.
VSEVsamField	This class represents a data field of a given VSAM record. A data field represents one specific column of a VSAM record. It is always part of a given VSAM map or view. It consists of a name (the column name), a length, a datatype, and an offset within the record.

Table 4. Contents of the VSE Java Beans Class Library (continued)

Class	Description
VSEVsamFilter	This class represents a filter when getting VSAM data from a VSAM file. It consists of a VSEVsamField, a filter string that may contain wildcards, and boolean operations.
VSEVsamMap	This class represents a data map for a given VSAM record. A map splits the VSAM record into columns, respectively data fields, that have a name, a length, a datatype, and an offset within the record. In addition to data fields, a map can contain data views that are subsets of the map's fields.
VSEVsamRecord	This class represents a data record of a given VSAM file. It provides methods to access the data fields of the record.
VSEVsamView	This class represents a data view on a given VSAM record together with a given data map. A data view always points to a subset of the data fields of a given VSAM map. As a consequence, actions against VSAM maps always influence the views of the related map. For example, deleting a map will also delete all views of this map.

Example of a Javadoc for a VSE Java Bean

Figure 76 shows an example of a Javadoc that was generated from the source code of a VSE Java Bean. The VSE Java Bean belongs to the VSE Java Beans class library.

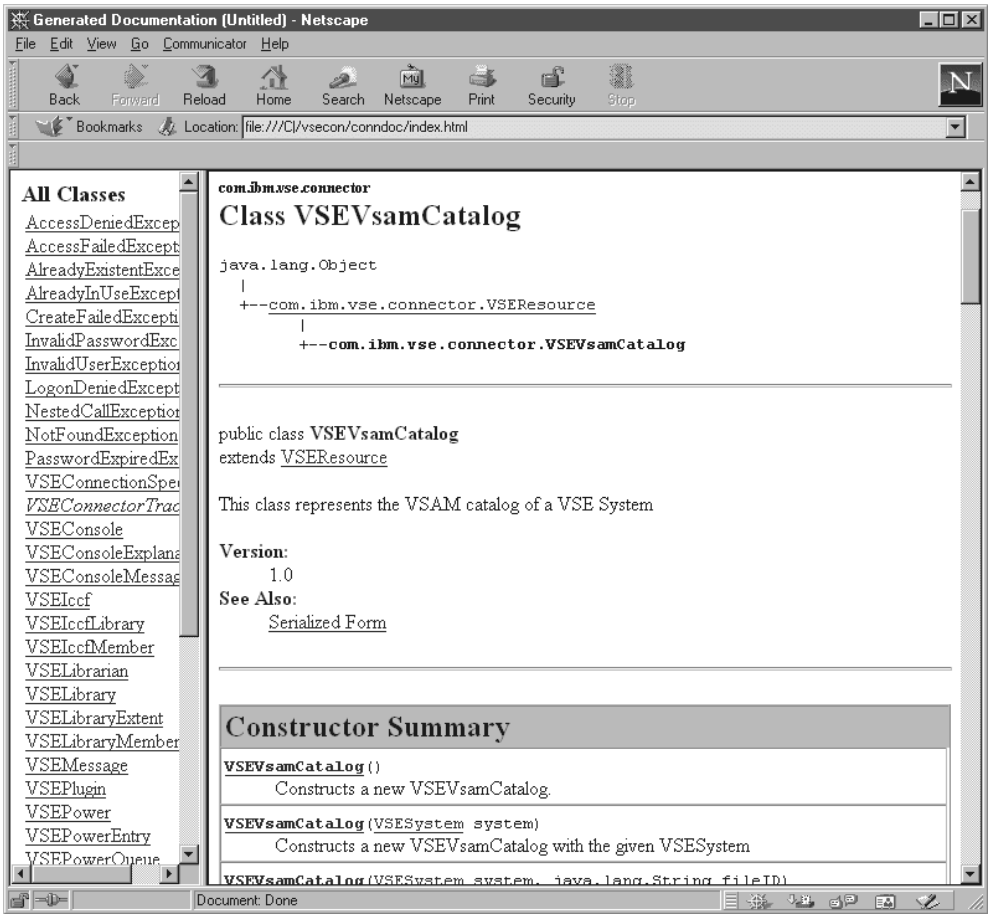


Figure 76. Example of a Javadoc Belonging to the VSE Java Beans Class Library

Note: The online documentation provided with the VSE Connector Client contains much information about the VSE Java Beans class library. See "Using the Online Documentation Options" on page 28 for details.

Using the Callback Mechanism of VSE Java Beans

You use the *callback mechanism* of VSE Java Beans to:

- Implement a callback function (a *VSEResourceListener*) that is called from the low level functions of the VSE Java Beans, which listen to the host connection, whenever data is received from the host.
- Add the resource listener to the VSE bean from which data shall be received Sending a request for data to the host.
- Get received data in the resource listener implementation, instance by instance.
- Remove the resource listener from the bean.
- Return data from the callback function to the caller.

This section first describes the callback mechanism in detail, and then describes how to access the various file systems and the operator console. You should also refer to the online documentation for other examples of resource listener implementations.

All VSE Java Beans methods returning lists of objects from the remote VSE host, return data instances through a callback routine. This means, the requesting function (the caller) returns after all data items are received from the host, but each data item can be processed immediately after it is received in the callback routine.

Note: When writing Enterprise Java Beans (EJBs) it is not possible to have callback routines. EJBs are not multi-thread enabled, nor do they support callbacks. Refer to the EJB samples for scenarios where EJBs are appropriate.

The VSE Java Beans class library provides the *VSEResourceListener* callback interface. This Java interface must be implemented by each application that wants to receive lists of VSE resources, such as lists of VSE libraries, sub-libraries, or members, as well as all kind of search results. The interface has only three methods:

listStarted(VSEResourceEvent event)
Called before the first data block is received from the host. Can be used for initialization purposes. The event does not contain any data.

listAdded(VSEResourceEvent event)
Called for each instance of a VSE resource, which is contained in the event.

listEnded(VSEResourceEvent event)
Called after the last data block has been received from the host. Can be used for cleanup purposes. The event does not contain any data.

Here is an example of a *VSEResourceListener* implementation. In addition to the above three required methods, additional methods are implemented which allow received data to be saved, and then returned to the caller.

The example shown in Figure 77 on page 160 listens for VSAM resources. The example also implements:

- a general listener, that listens to all possible VSE resources.
- specialized listeners, which listen only to some selected resources.

In this part of the example, two vectors are defined to store and accumulate all data objects that are received. As a result, a complete list of the data objects can be returned to the caller.

```
public class VsamListener implements VSEResourceListener
{
   public Vector catVector, fileVector;

   /**
    * constructs a new VsamListener. Two vectors are used to store
    * received resource instances.
    *
    */
   public VsamListener()
   {
      catVector = new Vector();
      fileVector = new Vector();
   }

   /**
    * allows the caller to reset the internal vectors.
    *
    */
   public void clear()
   {
      catVector.removeAllElements();
      fileVector.removeAllElements();
   }

   /**
    * returns the catalog list.
    *
    */
   public Vector getCatalogVector()
   {
      return catVector;
   }

   /**
    * returns the VSAM file list.
    *
    */
   public Vector getFileVector()
   {
      return fileVector;
   }
```

Figure 77. Example of a VSEResourceListener implementation (Part 1 of 2)

```
   /**
    * is called for the start of a list of VSEResources before
    * notifying about the list elements.
    * The event does not contain any data.
    *
    * @param event The VSEResourceEvent containing the source
    */
   public void listStarted(VSEResourceEvent event)
   {
      System.out.println("VsamListener: listStarted()");
   }
   /**
    * is called for each element of a list of VSEResources.
    * The VSEResourceEvent contains the data instance (see getData()).
    *
    * @param event The VSEResourceEvent containing the source and the data
    */
   public void listAdded(VSEResourceEvent event)
   {
      VSEResource resource = (VSEResource)(event.getData());
      if (resource instanceof VSEVsamCatalog)
      {
         VSEVsamCatalog cat = (VSEVsamCatalog)resource;
         catVector.addElement(cat);
      }
      else if (resource instanceof VSEVsamCluster)
      {
         VSEVsamCluster file = (VSEVsamCluster)resource;
         fileVector.addElement(file);
      }
   }
    /**
    * is called for the end of a list of VSEResources after
    * notifying about the list elements. The event does not
    * contain any data.
    *
    * @param event The VSEResourceEvent containing the source
    */
   public void listEnded(VSEResourceEvent event)
   {
      System.out.println("VsamListener: listEnded()");
   }
}
```

Figure 77. Example of a VSEResourceListener implementation (Part 2 of 2)

A typical flow for getting a list of VSAM catalogs would then look like this:

```
public static void main(String argv[ ]) throws IOException
{
   VSESystem system;       // the VSE host object
   VSEVsam vsam;           // the VSAM object
   VsamListener vl;        // implemented as shown above
   Vector vCatalogs;       // used to store the catalog list

   ...
   vsam = system.getVSEVsam();
   vl = new VsamListener();
   vsam.addVSEResourceListener(vl);
   vsam.getCatalogList();
   vsam.removeVSEResourceListener(vl);
   vCatalogs = vl.getCatalogVector();
   ...
}
```

Figure 78. Program Flow for Using VSE Java Classes to Obtain a List of VSAM Catalogs

Using VSE Java Beans to Connect to a VSE/ESA host

This section describes the definition of VSE host instances, and the setting up of the connection to the physical VSE/ESA host.

Each VSE host is represented by an object of class VSESystem. To be able to connect to a physical host, it is required to have a connection specification that holds all properties of the connection. Refer to the online documentation provided with the VSE Connector Client (see "Using the Online Documentation Options" on page 28) for further information about creating and reusing VSE host connections.

By default, a connection is given back to this pool after each host action. There is a method of the *VSESystem* class: *setConnectionMode* (true/false), which allows a connection to be held during the whole lifetime of your application (i.e. to not give it back to the pool after each host access). This should be done when implementing long running applications. The following shows a simple code example of defining a VSESystem and connecting to the host. These two steps are necessary to connect to a VSE host:

1. *Create a VSEConnectionSpec:*

```
...
/* Create connection specification. The connection spec */
/* holds information about the physical host connection and is */
/* stored permanently in the Common Connector Framework (CCF) */
try {
spec = new VSEConnectionSpec(InetAddress.getByName(ipAddr),
                                          2893,userID,password);
}
catch (UnknownHostException e)
{
   System.out.println("Unknown host : " + e);
   return;
}
spec.setMaxConnections(5);

/* Stay logon with this user for lifetime of this connection */
spec.setLogonMode(true);
```

Figure 79. Creating a VSEConnectionSpec for Connecting to the VSE/ESA Host

2. *Create a VSESystem:*

```
/* Create VSE system instance with this connection */
system = new VSESystem(spec);

/* Hold connection for the lifetime of our application */
system.setConnectionMode(true);
system.connect();
```

Figure 80. Creating a VSESystem for Connecting to the VSE/ESA Host

Notes:

1. It is not necessary to call the *connect()* method of a *VSESystem* to be able to communicate with the *VSESystem*. Instead, when calling a method that needs a host connection, the connection is opened.
2. For details of how to configure for SSL connections, see Figure 46 on page 109.

Using VSE Java Beans for Accessing VSAM Data

This example includes parts taken from the *VsamDisplayExample*, which is described in detail in the online documentation. The example displays the data of a VSAM file using a map, and adds a new record to a file. It assumes that there is a VSAM cluster FLIGHT.ORDERING.FLIGHTS containing a map FLIGHTS_MAP that describes the columns (data fields) of the record.

Note: Refer to the online documentation provided with the VSE Connector Client (see "Using the Online Documentation Options" on page 28) for details of how to create and re-use VSE/ESA host connections.

```
public class VsamDisplayExample
{
   public static void main(String argv[]) throws IOException
   {
      VSEConnectionSpec spec;
      VSESystem system;
      VSEVsam vsam;
      VSEVsamCatalog catalog;
      VSEVsamCluster cluster;
      VSEVsamRecord record, newRec=null;
      VSEVsamMap map;
      RecordListener rl;
      byte[ ] inputArray;
      String ipAddr, userID, password;
      Vector vRecords;
      String flightNum, newField="";

      String catName  = "VSESP.USER.CATALOG";
      String fileName = "FLIGHT.ORDERING.FLIGHTS";
      String mapName  = "FLIGHTS_MAP";

      /* Prompt for IP address, user ID and password. We have to remove */
      /* trailing blanks from the input strings before using them. */
      ...

      /* Get file system from host */
      System.out.println("Getting records from " + fileName + "...");
      vsam = system.getVSEVsam();

      /* Step 1: Create a listener that gets notified when objects */
      /* are retrieved from the host */
      rl = new RecordListener();
      vsam.addVSEResourceListener(rl);

      /* Step 2: Get VSAM records from host using the given map */
      map = new VSEVsamMap(system, catName, fileName, mapName);
      cluster = new VSEVsamCluster(system, catName, fileName);
      cluster.addVSEResourceListener(rl);
      cluster.selectRecords(map);
      cluster.removeVSEResourceListener(rl);
```

Figure 81. Example of Accessing VSAM Data Using VSE Java Beans (Part 1 of 3)

```
/* Get vector containing all records from listener */
vRecords = rl.getRecords();

/* Step 3: Display records ... */
int numMapFields = map.getNoOfFields();
System.out.println("Records in file " + fileName + " :");
for (int k=0;k<vRecords.size();k++)
{
   System.out.println("Record " + k + ":");
   record = (VSEVsamRecord)(vRecords.elementAt(k));
   for (int i=0;i<numMapFields;i++)
   {
      try {
         if (map.isFieldPartOfPrimaryKey(i))
            System.out.println(map.getFieldName(i) + " (Key) : " + record.getField(i).toString());
         else
            System.out.println(map.getFieldName(i) + " : " + record.getField(i).toString());
      }
      catch (Exception e)
      {
         System.out.println("writeRecord() : " + e);
      }
   }
   System.out.println("-----------------------------");
}

/* Step 4: Add a new flight to the cluster ... */
 boolean done = false;
 while (!done)
 {
    inputArray = new byte[map.getFieldLength(0)];
    System.out.println("Please enter a new flight number:");
    System.out.println("(Enter a negative value to quit)");
    System.in.read(inputArray);
    flightNum = new String(inputArray);
    flightNum = flightNum.trim();
    if ((new Integer(flightNum).intValue()) < 0)
       return;

     /* Check if new flight number already exists ... */
    try {
       newRec = new VSEVsamRecord(system, catName, fileName, mapName);
       newRec.setKeyField(0, new Integer(flightNum));
       newRec.add();
       done = true;
    }
    catch (Exception e)
    {
       System.out.println("Exception when adding new record.");
       if (e instanceof AlreadyExistentException)
          System.out.println("This flight number already exists.");
       else
          System.out.println("e = " + e);
    }
 }
```

Figure 81. Example of Accessing VSAM Data Using VSE Java Beans (Part 2 of 3)

```
      /* Add all other fields for new flight ... */
      for (int i=1;i<numMapFields;i++)
      {
         inputArray = new byte[map.getFieldLength(i)];
         System.out.println("Please enter value for field " + map.getFieldName(i) +
                            " (Length = " + map.getFieldLength(i) + ")");
         done = false;
         while (!done)
         {
            System.in.read(inputArray);
            newField = new String(inputArray);
            newField = newField.trim();
            if (newField.length() > 0)
               done = true;
         }

         if (map.getFieldType(i) == VSEVsamField.TYPE_STRING)
            newRec.setField(i, newField);
         else
            newRec.setField(i, new Integer(newField));
      }

      /* Make changes permanent... */
      newRec.commit();
      System.out.println("New record added to cluster.");
   }
}
```

Figure 81. Example of Accessing VSAM Data Using VSE Java Beans (Part 3 of 3)

Using VSE Java Beans for Accessing DL/I Data

This example includes parts taken from the *DliApiExample.Java*, which is described in detail in the online documentation. The example displays the data of a DL/I database. It assumes that the sample DL/I database has been defined and loaded, as described in Chapter 6, "Configuring DL/I for Access Via VSE Java Beans", on page 41.

Note: Refer to the online documentation provided with the VSE Connector Client (see "Using the Online Documentation Options" on page 28) for details of how to create and re-use VSE/ESA host connections.

```
/*****************************************************************************/
/*                                                                           */
/* MODULE NAME : DliApiExample.java                                          */
/*                                                                           */
/* DESCRIPTIVE NAME : Shows how to use VSE/DLI related classes.              */
/*                                                                           */
package com.ibm.vse.samples;

import java.lang.*;
import java.net.*;
import java.io.*;
import java.util.*;

/* Import Common Client Interface (CCI) classes */
import javax.resource.*;

/* Import VSE Connector classes */
import com.ibm.vse.connector.*;

public class DliApiExample
{
  protected static VSEConnectionSpec spec;
  protected static VSESystem system;
  protected static VSEDliPsb psb = null;
  protected static VSEDliPcb pcb = null;

  public static void main(String argv[])
  {
    String ipAddr, userID, password;
    byte[] ioarea;
    String[] ssas;

    try
    {
      /* Prompt for user ID and password. */
      BufferedReader r = new BufferedReader(
                              new InputStreamReader(System.in));
      System.out.println("Please enter your VSE IP address:");
      ipAddr = r.readLine();
      System.out.println("Please enter your VSE user ID:");
      userID = r.readLine();
      System.out.println("Please enter password:");
      password = r.readLine();
```

Figure 82. Example of Accessing DL/I Data Using VSE Java Beans (Part 1 of 5)

```
/* Create connection specification. */
try {
spec = new VSEConnectionSpec(InetAddress.getByName(ipAddr),
                                     2893,userID,password);
}
catch (UnknownHostException e)
{
   System.out.println("Unknown host : " + e);
   return;
}

/* Stay logon with this user for lifetime of this connection */
spec.setLogonMode(true);

/* Create VSE system instance with this connection */
system = new VSESystem(spec);

/* Get the DLI subsystem */
VSEDli dli = new VSEDli(system);

/* Step 1: schedule the PSB. */
System.out.println("Schedule the PSB");
psb = dli.getDliPsb("STBICLG");
psb.schedule();
System.out.println("  Num PCBs: "+psb.getNumberOfPCBs());
System.out.println("  IO len:   "+psb.getMaxLengthOfIOArea());
System.out.println();

/* Step 2: get a PCB. In this case, it's the first PCB of the PSB. */
System.out.println("Get a PCB");
pcb  = psb.getVSEDliPcb(0);
System.out.println("  DBDName = "+pcb.getDBDName());
System.out.println("  Processing Options: "+pcb.getProcessingOptions());
System.out.println();

/* Step 3: list all DL/I segments. The segments have the following
           data layout, as defined in the following COBOL copybook:
  01  STPIITM         REDEFINES IOAREA.
      02  ITNUMB    PIC X(6).
      02  ITDESC    PIC X(25).
      02  IQOH      PIC X(6).
      02  IQOR      PIC X(6).
      02  FILLER    PIC X(6).
      02  IUNIT     PIC 9(6).
      02  FILLER    PIC X(105).
*/
System.out.println("List all DL/I segments");
ssas = new String[1];
ssas[0] = "STPIITM  ";
do
{
  /* Get the next segment */
  ioarea = pcb.call("GN",null,ssas);
  System.out.println("  Status = "+pcb.getStatus());
  if (!pcb.getStatus().equals("  "))
    break;
```

Figure 82. Example of Accessing DL/I Data Using VSE Java Beans (Part 2 of 5)

```
  /* Print out the contents of the IOArea */
  System.out.println("  IOArea:");
  System.out.println("    ITNUMB  = "+VSEDliPcb.getStringFromBuffer(ioarea,0,6));
  System.out.println("    ITDESC  = "+VSEDliPcb.getStringFromBuffer(ioarea,6,25));
  System.out.println("    IQOH    = "+VSEDliPcb.getStringFromBuffer(ioarea,31,6));
  System.out.println("    IQOR    = "+VSEDliPcb.getStringFromBuffer(ioarea,37,6));
  System.out.println("    IUNIT   = "+VSEDliPcb.getStringFromBuffer(ioarea,49,6));
}
while(true);
System.out.println();

/* Step 4: Add/update a segment */
/*         First check if the segment is existing by issuing a GHU call,  */
/*         if yes, update the segment. Otherwise insert a new segment. */
System.out.println("Add/update a segment");

String item = "000700";  // item number to insert/update
ssas = new String[1];
ssas[0] = "STPIITM (STQIINO = "+item+")";

/* Get the segment */
ioarea = pcb.call("GHU",null,ssas);
System.out.println("  Status = "+pcb.getStatus());

if (pcb.getStatus().equals("GB") ||
    pcb.getStatus().equals("GE"))
{
   // segment not found -> do an insert
   System.out.println("  Segment not found, do an insert");

   // allocate a new ioarea
   ioarea = new byte[psb.getMaxLengthOfIOArea()];
   for (int i=0;i<ioarea.length;i++)
     ioarea[i] = 0x00;

   // set the new values into the IOArea
   VSEDliPcb.setStringToBuffer(ioarea,item,0,6);
   VSEDliPcb.setStringToBuffer(ioarea,"INSERTED ITEM",6,25);
   VSEDliPcb.setStringToBuffer(ioarea,"000001",31,6);
   VSEDliPcb.setStringToBuffer(ioarea,"000002",37,6);
   VSEDliPcb.setStringToBuffer(ioarea,"000000",43,6);
   VSEDliPcb.setStringToBuffer(ioarea,"000003",49,6);

   System.out.println("  IOArea:");
   System.out.println("    ITNUMB  = "+VSEDliPcb.getStringFromBuffer(ioarea,0,6));
   System.out.println("    ITDESC  = "+VSEDliPcb.getStringFromBuffer(ioarea,6,25));
   System.out.println("    IQOH    = "+VSEDliPcb.getStringFromBuffer(ioarea,31,6));
   System.out.println("    IQOR    = "+VSEDliPcb.getStringFromBuffer(ioarea,37,6));
   System.out.println("    IUNIT   = "+VSEDliPcb.getStringFromBuffer(ioarea,49,6));

   // do the insert
   ssas = new String[1];
   ssas[0] = "STPIITM  ";

   pcb.call("ISRT",ioarea,ssas);
}
```

Figure 82. Example of Accessing DL/I Data Using VSE Java Beans (Part 3 of 5)

```
else
{
   // segment already existing, do a update
   System.out.println("  Segment exists, do an update");
   System.out.println("  IOArea:");
   System.out.println("    ITNUMB  = "+VSEDliPcb.getStringFromBuffer(ioarea,0,6));
   System.out.println("    ITDESC  = "+VSEDliPcb.getStringFromBuffer(ioarea,6,25));
   System.out.println("    IQOH    = "+VSEDliPcb.getStringFromBuffer(ioarea,31,6));
   System.out.println("    IQOR    = "+VSEDliPcb.getStringFromBuffer(ioarea,37,6));
   System.out.println("    IUNIT   = "+VSEDliPcb.getStringFromBuffer(ioarea,49,6));

   // set the new values into the IOArea
   VSEDliPcb.setStringToBuffer(ioarea,item,0,6);
   VSEDliPcb.setStringToBuffer(ioarea,"UPDATED ITEM",6,25);
   VSEDliPcb.setStringToBuffer(ioarea,"000004",31,6);
   VSEDliPcb.setStringToBuffer(ioarea,"000005",37,6);
   VSEDliPcb.setStringToBuffer(ioarea,"000000",43,6);
   VSEDliPcb.setStringToBuffer(ioarea,"000006",49,6);

   // Print the IOArea contents ...
   System.out.println("  IOArea:");
   System.out.println("    ITNUMB  = "+VSEDliPcb.getStringFromBuffer(ioarea,0,6));
   System.out.println("    ITDESC  = "+VSEDliPcb.getStringFromBuffer(ioarea,6,25));
   System.out.println("    IQOH    = "+VSEDliPcb.getStringFromBuffer(ioarea,31,6));
   System.out.println("    IQOR    = "+VSEDliPcb.getStringFromBuffer(ioarea,37,6));
   System.out.println("    IUNIT   = "+VSEDliPcb.getStringFromBuffer(ioarea,49,6));

   // do the update
   ssas = new String[1];
   ssas[0] = "STPIITM  ";

   pcb.call("REPL",ioarea,ssas);
};
System.out.println("  Status = "+pcb.getStatus());
System.out.println();

/* Step 5: delete a segment */
System.out.println("Delete a segment");
item = "000700";  // item number to delete

// first verify if the segment is existing
ssas = new String[1];
ssas[0] = "STPIITM (STQIINO = "+item+")";

/* Get the segment */
ioarea = pcb.call("GHU",null,ssas);
System.out.println("  Status = "+pcb.getStatus());

if(pcb.getStatus().equals("GB") ||
   pcb.getStatus().equals("GE"))
{
  // segment not found
  System.out.println("  Segment not found");
}
```

Figure 82. Example of Accessing DL/I Data Using VSE Java Beans (Part 4 of 5)

```
      else
      {
        // segment existing, do a delete
        System.out.println("  Do the delete");
        System.out.println("  IOArea:");
        System.out.println("    ITNUMB  = "+VSEDliPcb.getStringFromBuffer(ioarea,0,6));
        System.out.println("    ITDESC  = "+VSEDliPcb.getStringFromBuffer(ioarea,6,25));
        System.out.println("    IQOH    = "+VSEDliPcb.getStringFromBuffer(ioarea,31,6));
        System.out.println("    IQOR    = "+VSEDliPcb.getStringFromBuffer(ioarea,37,6));
        System.out.println("    IUNIT   = "+VSEDliPcb.getStringFromBuffer(ioarea,49,6));

        // do the delete
        ssas = new String[1];
        ssas[0] = "STPIITM  ";

        pcb.call("DLET",null,ssas);

      };
      System.out.println("  Status = "+pcb.getStatus());
      System.out.println();

      /* Step 6: terminate the PSB */
      System.out.println("Terminate the PSB");
      psb.terminate();
      System.out.println();
    }
    catch(Throwable t)
    {
       t.printStackTrace();
       System.exit(0);
    }
  };
};
```

Figure 82. Example of Accessing DL/I Data Using VSE Java Beans (Part 5 of 5)

Using VSE Java Beans for Accessing VSE/POWER Data

This example includes code parts that are taken from the *PowerApiExample*, which is shown in detail in the *online documentation* (see page 28). Basically we deal with instances of *VSEPower*, *VSEPowerQueue*, and *VSEPowerEntry*. The example downloads a job from the reader queue and searches for list queue entries containing a given text string.

Note: Refer to the online documentation provided with the VSE Connector Client (see "Using the Online Documentation Options" on page 28) for details of how to create and re-use VSE/ESA host connections.

```
public class PowerApiExample
{
   protected static VSEConnectionSpec spec;
   protected static VSESystem system;
   protected static VSEPower power = null;

   public static void main(String argv[ ]) throws IOException
   {
      VSEPowerQueue readerQueue, listQueue;
      VSEPowerEntry entry;
      PowerListener pl;
      Vector vEntries;
      byte[] inputArray;
      String ipAddr, userID, password;

      /* Prompt for user ID and password. We have to remove trailing */
      /* blanks from the input strings before using them. */
      ...
      /* Get POWER reader queue object from host */
      power = system.getVSEPower();
      readerQueue = power.getReaderQueue();

      /* Get all reader entries with class 0. */
      /* Create a listener that gets notified when objects are */
      /* retrieved from the host */
      pl = new PowerListener();
      readerQueue.addVSEResourceListener(pl);
      readerQueue.getEntryList("*", "*", '0');

      /* Now remove the resource listener from the queue object */
      readerQueue.removeVSEResourceListener(pl);

      /* Now get the entry list from the listener */
      vEntries = pl.getEntryVector();
      for (int i=0;i<vEntries.size();i++)
      {
         entry = (VSEPowerEntry)(vEntries.elementAt(i));

         /* Look for the PAUSEBG job and download it */
         if (entry.getName().equals("PAUSEBG"))
         {
            File tempFile = new File("pausebg.job");
            System.out.println("Downloading pausebg...");
            entry.get(tempFile);
            System.out.println("pausebg downloaded.");
         }
      }
```

Figure 83. Accessing VSE/POWER Data Using VSE Java Beans (Part 1 of 2)

```
      /* Now scan some compile outputs in the list queue for */
      /* compile errors. Let's assume, we submitted a compile job  */
      /* called "compjob" in F4. The search() function now searches */
      /* all list queue entries (with all suffixes) that belong to */
      /* job "compjob", our user, with class = 4, for the string */
      /* "==ERROR" not case sensitive. */
      listQueue = power.getListQueue();
      listQueue.addVSEResourceListener(pl);
      pl.clearVector();
      listQueue.search("compjob", userID, '4', "==ERROR", true);
      listQueue.removeVSEResourceListener(pl);

      /* Get the results from the listener */
      vEntries = pl.getEntryVector();
      if (vEntries.size() == 0)
         System.out.println("No list queue entries contain the string" +
                            " \"==ERROR\".");

      for (int i=0;i<vEntries.size();i++)
      {
         entry = (VSEPowerEntry)(vEntries.elementAt(i));
         System.out.println("String found in : " + entry.getName() + "."
                           + entry.getNumber() + "["
                           + entry.getSuffix() + "]");
      }
   }
}
```

Figure 83. Accessing VSE/POWER Data Using VSE Java Beans (Part 2 of 2)

Using VSE Java Beans for Submitting Jobs

This example includes code parts that are taken from the *JobApiExample*, described in the online documentation. The example shows how to submit jobs, and track the status of submitted jobs. Two methods of doing so are described:

- The easiest way is to create a file on the local hard disk that contains the complete VSE/POWER job. This file is sent to the host, the job output is received into another local file.
- A second way, that does not need access to the local file system, is to create the JCL in memory and also receive the job output in memory line by line.

Note: Refer to the online documentation provided with the VSE Connector Client (see "Using the Online Documentation Options" on page 28) for details of how to create and re-use VSE/ESA host connections.

```
public class JobApiExample
{
   protected static VSEConnectionSpec spec;
   protected static VSESystem system;

   public static void main(String argv[]) throws IOException
   {
      JobInputStream job;
      JobOutputStream dest;
      byte[ ] inputArray;
      String ipAddr, userID, password, message;
      VSEPower power;
      File jobFile, outFile;
      Vector vLines;

      /* Prompt for user ID and password. We have to remove trailing */
      /* blanks from the input strings before using them. */
      ...

      /* Get POWER reader queue object from host */
      power = system.getVSEPower();

      /* 1. Submit a job file that is stored on the local disk */
      /* The execute() method returns as soon as the job output has been */
      /* transferred from the POWER list queue to outFile. */
      System.out.println("1. Submitting test.job");
      jobFile = new File("test.job");
      outFile = new File("out.txt");
      power.executeJob(jobFile, outFile);
      System.out.println("Output is now in out.txt in current directory.");

      /* 2. Create the job file in memory and send it to the host. */
      /* This has the advantage, that we don't need any access to */
      /* the local file system. */
      System.out.println("-------------------------------------");
      System.out.println("2. Creating job in memory...");
      job = new JobInputStream();
      dest = new JobOutputStream();
      power.executeJob(job, dest);
      vLines = dest.getAllLines();
      System.out.println("Number of output lines: " +
                         new Integer(vLines.size()).toString());
      //for (int i=0;i<vLines.size();i++)
      //{
      //   System.out.println((String)(vLines.elementAt(i)));
      //}
   }
}
```

Figure 84. Example of Submitting Jobs Using VSE Java Beans

Using VSE Java Beans for Accessing Librarian Data

This example includes code parts taken from the *LibrApiExample*, which is described in detail in the online documentation. The example gets a list of VSE libraries, then gets a list of sub-libraries within PRD2, then gets a list of members within PRD2.CONFIG. Finally it downloads the first member in PRD2.CONFIG to the local hard disk.

Note: Refer to the online documentation provided with the VSE Connector Client (see "Using the Online Documentation Options" on page 28) for details of how to create and re-use VSE/ESA host connections.

```
public class LibrApiExample
{
   protected static VSEConnectionSpec spec;
   protected static VSESystem system;
   protected static VSELibrarian libr = null;

   public static void main(String argv[]) throws IOException
   {
      LibrListener ll;
      VSELibrary myLib=null;
      VSESubLibrary mySublib=null;
      VSELibraryMember myMember=null;
      Vector vLibs, vSublibs, vMembers;
      int num, numLibs, numSublibs, numMembers;
      Calendar cal;
      byte[ ] inputArray;
      String ipAddr, userID, password;

      /* Prompt for IP address, user ID and password. We have to remove */
      /* trailing blanks from the input strings before using them. */
      ...

      /* Get Librarian file system from host */
      libr = system.getVSELibrarian();

      /* Create a listener that gets notified when objects are */
      /* retrieved from the host */
      ll = new LibrListener();
      libr.addVSEResourceListener(ll);

      /* Get a list of VSE libraries from this host. This method will */
      /* give control back when all resources are received, i.e. the */
      /* listEnded() method of the listener has been called. */
      libr.getLibraryList();
      /* Now remove the resource listener from the libr object */
      libr.removeVSEResourceListener(ll);

      /* Now get the list of libraries from the listener and get the */
      /* number of sublibraries in PRD2. */
      vLibs = ll.getLibVector();
      numLibs = vLibs.size();
      System.out.println("Number of libs on host " + system.toString() +
                         " : " + new Integer(numLibs).toString());
      for (int i=0;i<vLibs.size();i++)
      {
         myLib = (VSELibrary)(vLibs.elementAt(i));
         if (myLib.getName().equals("PRD2"))
            break;
      }
```

Figure 85. Example of Accessing Librarian Using VSE Java Beans (Part 1 of 2)

```
 if (myLib == null)
   return;
/* Now get a list of sublibraries in PRD2 and */
/* count them. Now we have to add the Listener to the */
/* sublibrary object. */
myLib.addVSEResourceListener(ll);
myLib.getSubLibraryList();
myLib.removeVSEResourceListener(ll);
numSublibs = myLib.getNumberOfSublibs();
System.out.println("Number of sublibs in " + myLib.getName() +
                   " : " + new Integer(numSublibs).toString());
vSublibs = ll.getSublibVector();
for (int i=0;i<vSublibs.size();i++)
{
   mySublib = (VSESubLibrary)(vSublibs.elementAt(i));
   if (mySublib.getName().equals("CONFIG"))
      break;
}
if (mySublib == null)
   return;
/* Now get a list of members in PRD2.CONFIG.*/
mySublib.addVSEResourceListener(ll);
mySublib.getMemberList();
mySublib.removeVSEResourceListener(ll);
vMembers = ll.getMemberVector();
numMembers = mySublib.getNumberOfMembers();
System.out.println("Number of members in " + mySublib.getName() +
                   " : " + new Integer(numMembers).toString());

/* Now display some properties of the first member */
myMember = (VSELibraryMember)(vMembers.elementAt(0));
System.out.println("First member in " + mySublib.getName() +
            " : " + myMember.getName() + "." + myMember.getType());
num = myMember.getNumberOfRecords();
System.out.println("Number of records: " +
                    new Integer(num).toString());
num = myMember.getLogicalRecordLength();
System.out.println("Logical record length: " +
                    new Integer(num).toString());
cal = myMember.getCreation();
System.out.println("Creation date: " + cal.getTime());
cal = myMember.getLastUpdate();
System.out.println("Last update: " + cal.getTime());
if (myMember.isBinary())
   System.out.println("Member is binary");
else
   System.out.println("Member format is text");

/* Finally download the member to the local harddisk  */
/* The download method gives control back when the download */
/* is finished. */
File localFile = new File(myMember.getName() + "." +
                          myMember.getType());
myMember.download(localFile);
 System.out.println(myMember.getName() + "." + myMember.getType() +
                   " downloaded.");
}
```

Figure 85. Example of Accessing Librarian Using VSE Java Beans (Part 2 of 2)

Using VSE Java Beans for Accessing VSE/ICCF Data

The example provided here includes code parts that are taken from the *IccfApiExample*, which is described in detail in the online documentation. The example:

1. Creates a host and gets a list of VSE/ICCF libraries.
2. Downloads member C$QCNBAT (a compile skeleton) out of VSE/ICCF library 2.
3. Displays some of the member's properties.

Notes:

1. Access to VSE/ICCF is read only.
2. Refer to the online documentation provided with the VSE Connector Client (see "Using the Online Documentation Options" on page 28) for details of how to create and re-use VSE/ESA host connections.

```
public class IccfApiExample
{
   protected static VSEConnectionSpec spec;
   protected static VSESystem system;
   protected static VSEIccf iccf = null;

   public static void main(String argv[ ]) throws IOException
   {
      IccfListener il;
      VSEIccfLibrary myLib=null;
      VSEIccfMember myMember=null;
      Vector vLibs, vMembers;
      int num, numLibs, numMembers;
      Calendar cal;
      byte[ ] inputArray;
      String ipAddr, userID, password;

      /* Prompt for IP address, user ID and password. We have to remove */
      /* trailing blanks from the input strings before using them. */
      ...

      /* Get ICCF file system from host */
      iccf = system.getVSEIccf();

      /* Create a listener that gets notified when objects are */
      /* retrieved from the host */
      il = new IccfListener();
      iccf.addVSEResourceListener(il);

      /* Get a list of ICCF libraries from this host. This method will */
      /* give control back when all resources are received, i.e. the */
      /* listEnded() method of the listener has been called. */
      iccf.getLibraryList();

      /* Now remove the resource listener from the iccf object */
      iccf.removeVSEResourceListener(il);
```

Figure 86. Example of Accessing ICCF Data Using VSE Java Beans (Part 1 of 3)

```
/* Get library numbers from listener object and download */
/* a compile skeleton from ICCF library 2. */
vLibs = il.getLibVector();
for (int i=0;i<vLibs.size();i++)
{
   myLib = (VSEIccfLibrary)(vLibs.elementAt(i));
   if (myLib.getLibrary() == 2)
   {
      /* Get memberlist of ICCF lib 2 */
      myLib.addVSEResourceListener(il);
      myLib.getMemberList();
      myLib.removeVSEResourceListener(il);
      vMembers = il.getMemberVector();
      for (int j=0;j<vMembers.size();j++)
      {
         myMember = (VSEIccfMember)(vMembers.elementAt(j));
         if (myMember.getName().equals("C$QCNBAT"))
         {
            myMember.download("C$QCNBAT.SKL");
         }
      }
   }
}
System.out.println("C$QCNBAT.SKL downloaded.");

/* A shorter way to download a specific member is to search */
/* it and then directly download it. */
iccf.addVSEResourceListener(il);
il.clear();
iccf.search(2, "C$QCNBAT", "*");
iccf.removeVSEResourceListener(il);
vMembers = il.getMemberVector();
if (vMembers.size() == 1)
{
   myMember = (VSEIccfMember)(vMembers.elementAt(0));
   myMember.download("C$QCNBAT.SKL2");
}
System.out.println("C$QCNBAT.SKL2 downloaded.");

/* A very short way to download the member. Simply create a */
/* member object and try to download. If the member does not */
/* exist on the host, we get an exception. */
myMember = new VSEIccfMember(system, 2, "C$QCNBAT");
myMember.download("C$QCNBAT.SKL3");
System.out.println("C$QCNBAT.SKL2 downloaded.");

/* Now display some properties of the skeleton file */
System.out.println("Properties of C$QCNBAT:");
cal = myMember.getDate();
System.out.println(" Date       : " + cal.getTime());
System.out.println(" User       : " + myMember.getUser());
if (myMember.isCommon())
   System.out.println(" Common     : yes");
else
   System.out.println(" Common     : no");
if (myMember.isCompressed())
   System.out.println(" Compressed : yes");
else
   System.out.println(" Compressed : no");
if (myMember.isDBCS())
   System.out.println(" DBCS       : yes");
else
   System.out.println(" DBCS       : no");
```

Figure 86. Example of Accessing ICCF Data Using VSE Java Beans (Part 2 of 3)

```
      if (myMember.isPrivate())
         System.out.println(" Private    : yes");
      else
         System.out.println(" Private    : no");
      /* Try a member that does not exist */
      try {
      myMember = new VSEIccfMember(system, 2, "MYMEMBER");
      myMember.download("MYMEMBER.TXT");
      }
      catch (Exception e)
      {
         System.out.println("Exception when downloading member, ");
         System.out.println("e = " + e);
         System.out.println("MYMEMBER does not exist in ICCF library 2.");
      }
   }
}
```

Figure 86. Example of Accessing ICCF Data Using VSE Java Beans (Part 3 of 3)

Using VSE Java Beans for Accessing the Operator Console

There are two ways of how to get console messages when sending a console command:

1. Using the *execute()* method of class *VSEConsole*, returns a complete list of messages.
2. Using *open()*, *setCommand()*, *getMessage()*, and *close()*, allows you to work with the first messages while others are still being retrieved.

This example includes code parts that are taken from the *ConsoleApiExample*, which is described in detail in the online documentation.

Note: Refer to the online documentation provided with the VSE Connector Client (see "Using the Online Documentation Options" on page 28) for details of how to create and re-use VSE/ESA host connections.

```
public class ConsoleApiExample
{
   protected static VSEConnectionSpec spec;
   protected static VSESystem system;

   public static void main(String argv[ ]) throws IOException
   {
      VSEConsole cons;
      Vector vConsOutput;
      byte[ ] inputArray;
      String ipAddr, userID, password;
      VSEConsoleMessage message;
      boolean finished = false;

      /* Prompt for user ID and password. We have to remove trailing */
      /* blanks from the input strings before using them. */
      ...

      /* Create console and send a command. The output is returned */
      /* in vector vConsOutput. Note: we must specify an end string */
      /* as 3rd parameter in cons.execute(). Otherwise we would not */
      /* have an end condition for the function to return immediately */
      /* after the last message. */
      cons = new VSEConsole(system);
      System.out.println("Example 1: Waiting for complete list of
                                                 console messages...");
      vConsOutput = cons.execute("map", null, "AR 0015 1I40I  READY", 30);
      for (int i=0;i<vConsOutput.size();i++)
      {
         System.out.println(((VSEConsoleMessage)
                             (vConsOutput.elementAt(i))).getMessage());
      }
```

Figure 87. Example of Accessing the Operator Console Using VSE Java Beans (Part 1 of 2)

```
      /* Do the same again using open() and setCommand(). Here we have */
      /* the advantage, that we can see the messages while they come in. */
      /* Using the execute() function, we can display the messages after */
      /* we got all of them. */
      System.out.println(" ");
      System.out.println("Example 2: Getting messages line by line...");
      cons.open();
      cons.setCommand("map");
      message = cons.getMessage();
      while (finished == false)
      {
         if (message != null)
         {
            System.out.println(message.getMessage());
            if (message.getMessage().indexOf("AR 0015 1I40I  READY") >= 0)
               finished = true;
         }
         message = cons.getMessage();
      }
      cons.close();
   }
}
```

Figure 87. Example of Accessing the Operator Console Using VSE Java Beans (Part 2 of 2)

Using the VSE Navigator Application

The *VSE Navigator* is a Java application that illustrates the use of the VSE Java Beans. It uses virtually all the VSE Java Beans shown in Table 4 on page 155. It implements a graphical user interface (GUI) that has a similar appearance to many of the currently-available file managers.

The *client-part* of the VSE Navigator, which communicates with the *VSE Connector Server*, provides you with a variety of functions. You can, for example:

- Access VSE file systems (POWER, Librarian, ICCF, VSAM).
- Create and submit jobs, including generating jobs based upon the skeletons stored in ICCF library 2.
- Work with the VSE operator console.
- Compare files, and perform a full-text search in VSE-based file systems.
- Interactively insert and edit VSAM records.
- Display:
  - the VSE hardware configuration, including the property dialogs for attached devices
  - the VSE system activity (CPU usage, and so on)
  - the current VSE service level
  - the system labels
  - the system tasks
  - the used and free VSAM space
  - VSAM data, via maps and views

Here is the GUI provided by the VSE Navigator:

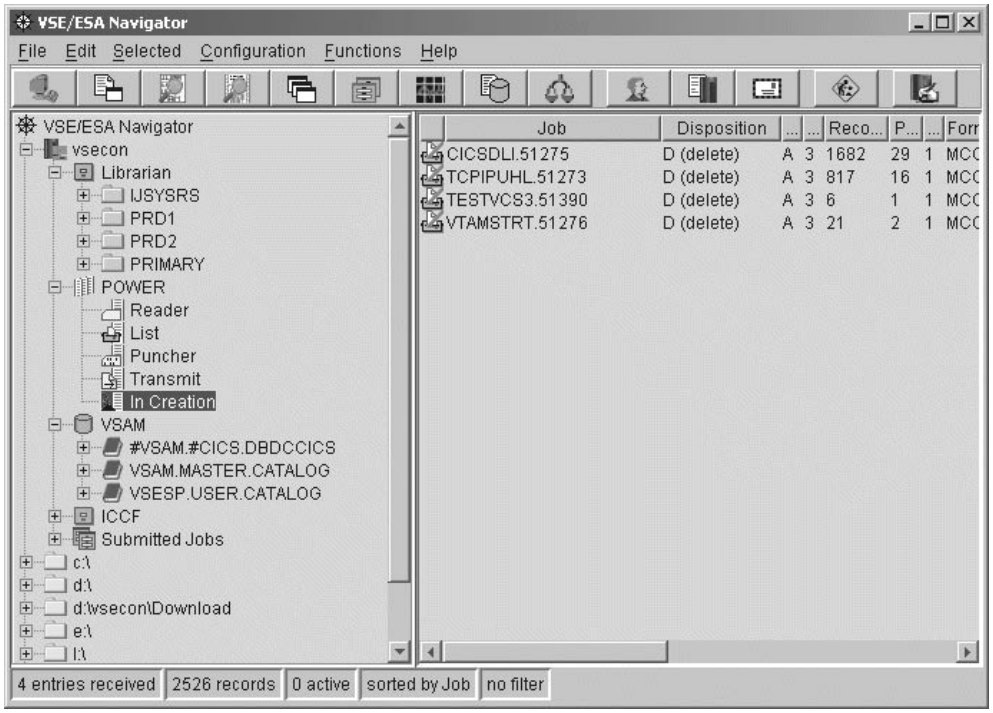


Figure 88. Graphical User Interface, as Provided by the VSE Navigator

Prerequisite for Using the VSE Navigator

Before installing and using the VSE Navigator, you should have installed the VSE Connector Client, as described in "Installing the VSE Connector Client" on page 25.

Migrating From Earlier Versions

If you currently have a pre-VSE/ESA 2.7 version of the VSE Navigator installed, you must migrate to the latest version that runs with VSE/ESA 2.7. To do so, simply download the version of the VSE Navigator that runs with VSE/ESA 2.7, and use it to replace your existing version.

For details, see "Installing the VSE Navigator" (below).

Installing the VSE Navigator

You install the VSE Navigator on a Java-enabled platform.

Before you begin, you must already have installed the Java Development Kit (JDK) 1.3 or higher on the development platform where you plan to install the VSE Navigator. If you do not have JDK 1.3 or higher installed, refer to "Installing and Configuring Java" on page 21 for details of how to install it.

The VSE Navigator is supplied as one Java-installation class file, **install.class**. To install the VSE Navigator, you should:

1. Obtain the VSE Script Server from the Internet, by starting your Web browser and proceeding to URL:

   `http://www.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/conmain.htm`

   From within the VSE Navigator section, select **Details and Download**. Then download the latest code to the directory where you wish to install the VSE Navigator, by selecting the file **vsenavi***nnn***.zip**. **Note:** *nnn* refers to the current VSE version (for example, **vsenavi270.zip**).
2. Unzip the file **vsenavi***nnn***.zip**, which contains these files:
   - install.class (contains the VSE Navigator code)
   - install.bat (an install batch file for Windows)
   - install.cmd (an install batch file for OS/2)
   - install.sh (an install script for Linux/Unix)
3. Start the batch file (by double-clicking the file) that is applicable to your operating-system platform.
4. The installation process now begins, and you are guided through various installation menus.
5. The *Setup Complete* window is displayed when the installation is complete. After making your selection, click **Finish**. The installation now completes.

   Note: Desktop icons are only created for Windows and OS/2 platforms (not for AIX, Linux, and so on).

Starting the VSE Navigator Client

To start the VSE Navigator client:

On Operating System...	The run file or shell script is ...
Windows	run.bat
OS/2	run.cmd
Unix/Linux	run.sh

Note: If you are using a Java Runtime Environment (JRE), you can use the **jre11.cmd** file, to start the VSE Navigator. However, the installation process does not create a desktop icon for this file.

On Windows and OS/2 you are provided with desktop objects you can use to start the VSE Navigator, access the online documentation (file **NaviReference.html**), and so on.

When starting the VSE Navigator client for the first time, you will be required to specify your local:
- utilities (for example, the Web browser you wish to use for displaying Help texts)
- directories

Adding Your Own VSE Navigator Plug-Ins

The VSE Navigator provides a Java programming interface that allows you to add your own "plug-in" functions to the VSE Navigator client. To write a VSE Navigator plug-in, you must:

1. Implement the methods of a Java interface.
2. Copy the class files into the VSE Navigator's plug-in directory. You may also create a new directory for your classes within the plug-in directory.
3. Restart the VSE Navigator client. Your plug-in is then dynamically loaded, and can be accessed from the toolbar and menus of the VSE Navigator. The online Programming Reference manual is HTML-based, and you can access it from **NaviReference.html**, which is stored in the directory where you installed the VSE Navigator.

Here is the GUI provided for using SSL with the VSE Navigator:

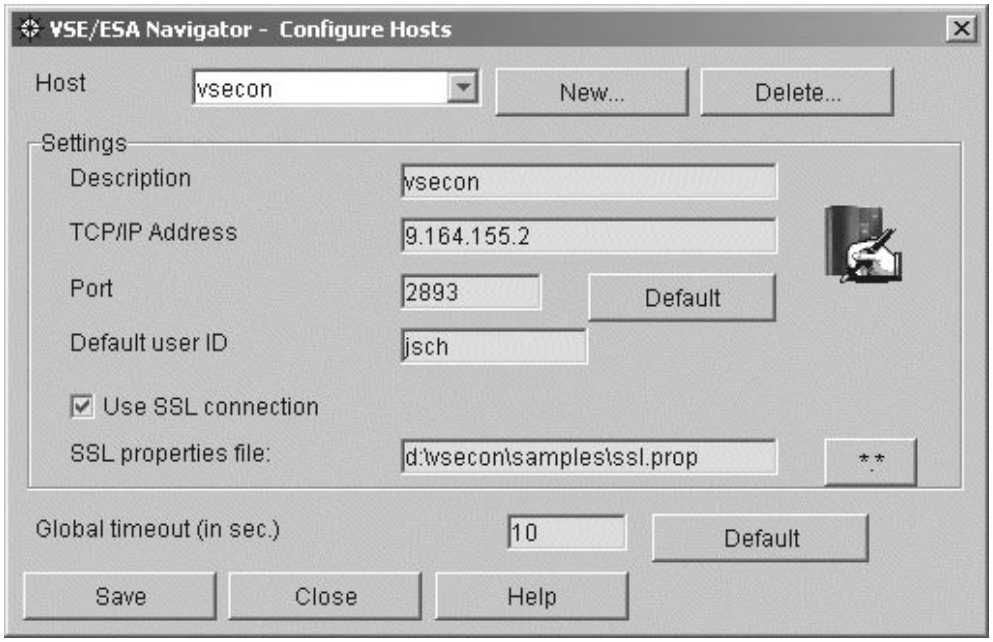


Figure 89. Configure Hosts for the VSE Navigator

Here is the GUI provided for using SSL with the VSE Navigator:

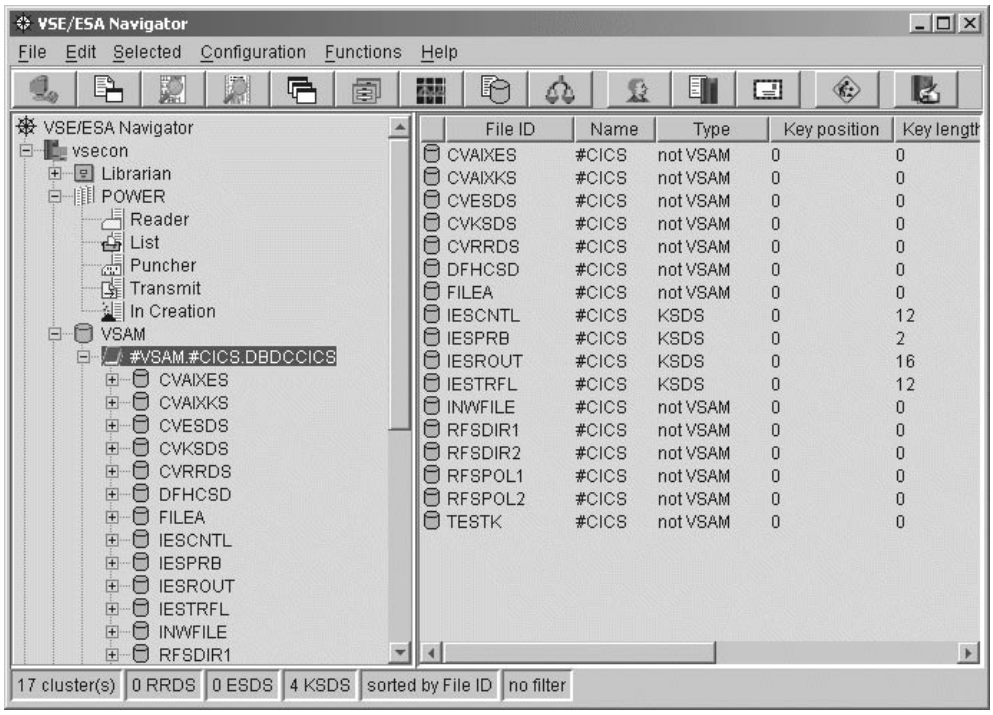


Figure 90. Using the VSE Navigator to Access CICS Data

Chapter 18. Using JDBC to Access VSAM Data

This chapter describes how you can setup and issue relational database queries and update requests against VSAM data using a *Java Database Connectivity* (JDBC) driver. To do so, the Java-based connector provides various classes for use with JDBC.

You can therefore use SQL constructs to access VSAM data, instead of coding against the VSE Java Beans interface. Currently, only a subset of the SQL syntax is supported (shown in "SQL Statements That Are Supported by JDBC"). However, this is the same subset that can be used with the VSAM SQL call level interface (CLI) from within a DB2-Stored Procedure (see "Using DB2 Stored Procedures to Access VSAM Data" on page 312 for details).

The advantages of using the JDBC driver instead of the VSE Java Beans interface are:

- JDBC and SQL are standard interfaces for accessing data in relational databases.
- Many products such as the *IBM Visual Age for Java* support the JDBC interface.
- You can integrate VSAM access into applications that were created using the *IBM Visual Age for Java* program, without needing to include VSE-specific code.

For details of how to map non-relational VSAM data to a relational structure, refer to Chapter 15, "Mapping VSE/VSAM Data to a Relational Structure", on page 129.

This chapter contains these main sections:

- "SQL Statements That Are Supported by JDBC"
- "Relational and VSE Java Beans Terminology" on page 190
- "Example of Using JDBC to Access VSAM Data" on page 191

SQL Statements That Are Supported by JDBC

These are the SQL statements that are supported by JDBC:

Table 5. SQL Statements Supported by JDBC

<insert statement> =	"INSERT" ["INTO"] <table name> [<column name list>] "VALUES" "(" <insert values list> ")" ("," "(" <insert values list> ")")* | <select statement>
<table name> =	<IDENTIFIER> "\" <IDENTIFIER> "\" <IDENTIFIER> ["\" <IDENTIFIER>]
<column name list> =	"(" <column name> ("," <column name>)* ")"
<column name> =	<IDENTIFIER>
<insert values list>=	"(" <insert value> ("," <insert values>)* ")"

© Copyright IBM Corp. 2000, 2003

Table 5. SQL Statements Supported by JDBC (continued)

<insert value> =	<NUMBER> | <CHAR LITERAL> | <PREPARED EXPR>
<update statement> =	"UPDATE" <table name> "SET" <column values> [<where clause>]
<column values> =	<column name> "=" <updated value> ("," <column name> "=" <updated value>)*
<updated value> =	<NUMBER> | <CHAR LITERAL> | <PREPARED EXPR>
<delete statement> =	"DELETE" ["FROM"] <table name> [<where clause>]
<query statement> =	<select statement>
<select statement> =	<select no order> [<order by clause>]
<select no order> =	"SELECT" ["ALL" | "DISTINCT"] <select list> <from clause> [<where clause>]
<select list> =	"*" | <select item> ("," <select item>)*
<select item> =	(<IDENTIFIER> "\" [<IDENTIFIER> "\" <IDENTIFIER> "\" [<IDENTIFIER> "\"]] "*") | (<sl element> ["AS"] [<IDENTIFIER> <QUOTED IDENTIFIER>])
<sl element> =	<sl mult expr> (("+" | "-") <sl mult expr>)*
<sl mult expr> =	<sl primary expr> (("*" | "/") <sl primary expr>)*
<sl primary expr> =	<table column> | <NUMBER> | <CHAR LITERAL> | <PREPARED EXPR> | "(" <sl element> ")"
<from clause> =	"FROM" <from item> [("," <from item>)+ | (<join clause>)+]

Table 5. SQL Statements Supported by JDBC (continued)

<from item> =	("(" <subquery> ")" | <table name>) ["AS"] [<IDENTIFIER>]
<join clause> =	<join type> <from item> [<join specification>]
<join type> =	"INNER" "JOIN" | "NATURAL" ["INNER"] "JOIN" | "LEFT" ["OUTER"] "JOIN" | "RIGHT" ["OUTER"] "JOIN" | "FULL" ["OUTER"] "JOIN" | "JOIN"
<join specification> =	"USING" "(" <IDENTIFIER> ("," <IDENTIFIER>)* ")" | "ON" <table column> "=" <table column>
<where clause> =	"WHERE" <where expr>
<where expr> =	<where and expr> ("OR" <where and expr>)*
<where and expr> =	<where rel expr> ("AND" <where rel expr>)*
<where rel expr> =	["NOT"] (<where primary expr> <relop> <where primary expr>) | "(" <where expr> ")"
<where primary expr> =	<table column> | <NUMBER> | <CHAR_LITERAL> | <PREPARED EXPR>
<order by clause> =	"ORDER" "BY" <table column> ["ASC" | "DESC"] ("," <table column> ["ASC" | "DESC"])*
<relop> =	"=" | "<>" | "<" | "<=" | ">" | ">="
<subquery> =	<select no order>

Table 5. SQL Statements Supported by JDBC (continued)

```
<table column> =        <IDENTIFIER>
                        [ "\" <IDENTIFIER>
                            [ "\" <IDENTIFIER> "\" <IDENTIFIER>
                                [ "\" <IDENTIFIER> ]]]
```

Relational and VSE Java Beans Terminology

Here are some guidelines for understanding how terms that are used with relational SQL, correspond to non-relational VSE Java Beans.

Table 6. Relational Terms and Their VSE Equivalents

SQL term	VSE Java Beans term
SQL table	A VSEVSAMCatalog, a VSEVsamCluster, together with a VSEVsamMap (and optionally VSEVsamView)
Database row	VSEVsamRecord, together with a VSEVsamMap describing the column names and data field properties.
Column name	VSEVsamField, describing the column's: Name Offset within the record Length Data type (string, integer, and so on)

Specifying Table Names

The following example illustrates how you specify table names with JDBC. Assume the cluster `MY.TEST.CLUSTER` resides in catalog `MY.USER.CATALOG`. A map has been defined for this cluster with the name `MY.TEST.MAP`.

The table name used for the VSAM JDBC driver will therefore be:

```
MY.USER.CATALOG\MY.TEST.CLUSTER\MY.TEST.MAP
```

Example of Using JDBC to Access VSAM Data

This section provides a detailed example of how you can access VSAM data using the JDBC driver. The example performs roughly the same processing as the *FlightOrderingServlet* described in "Example of How to Implement a Servlet" on page 246. It operates on the same VSAM clusters as the *FlightOrderingServlet*, but does not create these clusters. Therefore, if you have not already created these clusters (and filled them with data), refer to "Creating the VSAM Clusters for the Sample" on page 247 for details of how to do so.

The example servlet is implemented in the Java source file **JdbcFlightOrderingServlet.Java**, which is supplied with the online documentation (see "Using the Online Documentation Options" on page 28 for details).

Step 1. Include the Imports and Class Definition

The JDBC-related Java beans are stored in the **com.ibm.vse.jdbc** package. The **java.sql** classes are also required in order to create the JDBC driver.

```
import java.lang.*;
import java.net.*;
import java.io.*;
import java.util.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.vse.jdbc.*;

public class JdbcFlightOrderingServlet extends HttpServlet
{

.
.
.
    // Names of the Clusters and Maps.
    String   vsamCatalog    = "VSESP.USER.CATALOG";
    String   flightsCluster = "FLIGHT.ORDERING.FLIGHTS";
    String   ordersCluster  = "FLIGHT.ORDERING.ORDERS";
    String   flightsMapName = "FLIGHTS_MAP";
    String   ordersMapName  = "ORDERS_MAP";

.
.
.
```

Figure 91. JDBC Example: Imports and Class Definition

Step 2. Perform the doGet() Method

The doGet() method gets the HTTP request from your Web browser. It creates an instance of the VSAM JDBC driver, and gets a connection to the host. The numbers below refer to the numbers shown in Figure 92.

1. Create an initial context which is later used to perform a lookup of the data source. The data source is accessed via its JNDI name, which is defined during the deployment of the JDBC driver.
2. In a WebSphere environment, request a JDBC connection from the data source. The properties of the connection (such as hostname, user ID, and so on) are specified during the deployment of the JDBC driver. For an example of how to create an instance of a JDBC connection in a *non-WebSphere environment*, see Figure 75 on page 150.

```
.
.
.
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
   throws ServletException, IOException
   {
.
.
.
1     /* Create a instance of the VSAM JDBC Driver and get */
      /* a connection */
      java.sql.Driver      jdbcDriver = null;
      java.sql.Connection  jdbcCon    = null;

      try
      {
2       // Lookup a JDBC Connection
        Context ctx = new InitialContext();
        DataSource ds = (DataSource)ctx.lookup("jdbc/vsamjdbc");
        jdbcCon = ds.getConnection();
      }
      catch(Throwable t)
      {
.
.
.
      }
.
.
.
      // Get Parameters
.
.
.
      // check whitch action to do
.
.
.

      jdbcCon.close();
   }
```

Figure 92. JDBC Example: Perform the doGet() Method

Step 3. Display List of Available Flights

This method displays a list of available flights that can be ordered. The numbers below refer to the numbers shown in Figure 93.

1. Create the SQL statement.
2. Executes the request.

```
...
   public void doOrderStep1(PrintWriter out,
                            java.sql.Connection jdbcCon)
   {
      int flightNumber,price;
      String start,destination,departure,arrival,airline;

...
      // write out a Table containing all available flights

...
      try
      {
         // get a statement
1.        java.sql.Statement stmt = jdbcCon.createStatement();

         // execute the query
2.        java.sql.ResultSet rs = stmt.executeQuery("SELECT * FROM "+
               vsamCatalog+"\\"+flightsCluster+"\\"+flightsMapName);

         // walk through the results
         while(rs.next())
         {
            // get the values
            flightNumber = rs.getInt("FLIGHT_NUMBER");
            start        = rs.getString("START");
            destination  = rs.getString("DESTINATION");
            departure    = rs.getString("DEPARTURE");
            arrival      = rs.getString("ARRIVAL");
            price        = rs.getInt("PRICE");
            airline      = rs.getString("AIRLINE");

            // write out a line in the table

...
         }
         rs.close();
         stmt.close();
      }
      catch(SQLException t)
      {

...
      }

...
   }

...
```

Figure 93. JDBC Example: Display List of Available Flights

Step 4. Display Flight Properties and Entry Form

The method used here displays the flight's properties, together with a form in which the order's parameters can be entered. It illustrates how you can set up a more complex query. The numbers below refer to the numbers shown in Figure 94.

1. All question marks in the SQL query will be later replaced by actual values.
2. Replace the placeholders.
3. Execute the query.

```
...
   public void doOrderStep2(PrintWriter out,
                            java.sql.Connection jdbcCon,
                            Hashtable parameters)
   {
     int flightNumber,price,seats,reserved;
     String start,destination,departure,arrival,airline;

     // get the parameters

...

     // get the flight and display the properties
     try
     {
       // get a statement
1.     java.sql.PreparedStatement stmt = jdbcCon.prepareStatement(
                "SELECT * FROM "+vsamCatalog+"\\"+flightsCluster+"\\"+flightsMapName+
                " WHERE FLIGHT_NUMBER=?");

       // set the select filter
2.     stmt.setInt(1,flightNumber);
       // execute the query
3.     java.sql.ResultSet rs = stmt.executeQuery();

...
```

Figure 94. JDBC Example: Display Flight Properties and Entry Form

Step 5. Get and Display the Data Field Values

The result should be one flight only that matches the above *flightNumber*. Here we get the data field values of the flight, and display them in the HTML page.

```
       // get the values
       flightNumber = rs.getInt("FLIGHT_NUMBER");
       start        = rs.getString("START");
       destination  = rs.getString("DESTINATION");
       departure    = rs.getString("DEPARTURE");
       arrival      = rs.getString("ARRIVAL");
       price        = rs.getInt("PRICE");
       seats        = rs.getInt("SEATS");
       reserved     = rs.getInt("RESERVED");
       airline      = rs.getString("AIRLINE");

       rs.close();
       stmt.close();

       // display the fields

.
.
.

       // check if enough seats are available
       if (seats-reserved == 0)
       {

.
.
.
         return;
       }

       // write out a form where the user can enter his name
       // and select the number of seats to order

.
.
.
     }
     catch (SQLException t)
     {

.
.
.
     }

.
.
.
   }

.
.
.
```

Figure 95. JDBC Example: Get and Display the Data Field Values

Step 6. Update the VSAM Cluster

The method used here illustrates how you can update a VSAM cluster. It orders a flight, and then updates the FLIGHT.ORDERING.ORDERS cluster.

We select all records from the FLIGHTS cluster where the flight number matches the value of the variable *flightNumber*. This will be, of course, the flight for which we want to place an order.

```
⋮
   public void doOrderStep3(PrintWriter out,
                            java.sql.Connection jdbcCon,
                            Hashtable parameters)
   {
     int flightNumber,seatsToOrder,seats,reserved,price;
     String firstName,lastName;
     boolean nonSmoke;
     int num,counter,recNo;

     // get the parameters from the form

⋮

     try
     {
       // setup the SQL query
       java.sql.PreparedStatement stmt = jdbcCon.prepareStatement(
              "SELECT * FROM "+vsamCatalog+"\\"+flightsCluster+"\\"+flightsMapName+
              " WHERE FLIGHT_NUMBER=?");

       // set the select filter
       stmt.setInt(1,flightNumber);

       // execute the query
       java.sql.ResultSet rs = stmt.executeQuery();
```

Figure 96. JDBC Example: Update the VSAM Cluster

Step 7. Get Order Values and Check Availability

In this step, we obtain one record only, since the flight number is the key field in this cluster.

```
       // there should be only one result
       if (!rs.next())
       {

⋮
         return;
       }

       // get the values
       price        = rs.getInt("PRICE");
       seats        = rs.getInt("SEATS");
       reserved     = rs.getInt("RESERVED");

       rs.close();
       stmt.close();

       // check if enough seats are available
       if (reserved + seatsToOrder > seats)
       {

⋮
          return;
       }

       // display the order properties

⋮
```

Figure 97. JDBC Example: Get Order Values and Check Availability

Step 8. Create the New Order

In this step we create the new order. The numbers below refer to the numbers shown in Figure 98.

1. The SQL statement inserts a new record. Each question mark is a placeholder that is enumerated below when the data field values are specified.
2. The related values are specified.
3. The SQL UPDATE request is executed.

```
      // get a statement
1.   stmt = jdbcCon.prepareStatement(
       "INSERT INTO "+vsamCatalog+"\\"+ordersCluster+"\\"+ordersMapName+
       " (FIRST_NAME,LAST_NAME,FLIGHT_NUMBER,SEATS,NON_SMOKE) VALUES(?,?,?,?,?)");

      // set the values
2.      stmt.setString(1,makeString(firstName,20));
      stmt.setString(2,makeString(lastName,20));
      stmt.setInt(3,flightNumber);
      stmt.setInt(4,seatsToOrder);
      byte[] b = new byte[1];
      if(nonSmoke)
        b[0] = 0x01;
      else
        b[0] = 0x00;
      stmt.setBytes(5,b);

      // execute the query
3.    num = stmt.executeUpdate();
      stmt.close();

⋮
```

Figure 98. JDBC Example: Create the New Order

Step 9. Check Processing and Increase Reserved Seats

In this step, a check is made to see if the previous INSERT operation into the ORDERS cluster completed successfully. The variable *num* now contains the number of successfully-inserted rows.

If successful, the number of reserved seats in the FLIGHTS cluster of the corresponding flight is increased by the number of newly reserved seats.

```
       // check if creating was ok
       if (num == 1)
       {
         // now update the flight record's fields and increase
         // the reserved seats by the number of seats to order
         reserved += seatsToOrder;

         // get a statement
         stmt = jdbcCon.prepareStatement(
             "UPDATE "+vsamCatalog+"\\"+flightsCluster+"\\"+flightsMapName+
             " SET RESERVED=? WHERE FLIGHT_NUMBER=?");

         stmt.setInt(1,reserved);
         stmt.setInt(2,flightNumber);

         // execute the query
         num = stmt.executeUpdate();
       }
     }
     catch (SQLException t)
     {

.
.
.
        return;
     }

.
.
.
   }

.
.
.

.
.
.
}
```

Figure 99. JDBC Example: Check Processing and Increase Number of Reserved Seats

Chapter 19. Using Java Applets to Access Data

Java applets are Java programs that run inside the Java Virtual Machine of a Web browser. The main advantages of using applets are that:

- They can be accessed using any Java-enabled Web browser.
- There is no need to install any further programs on the Web client.
- They allow you to build an easy-to-use User Interface (UI).

Applets are by definition *secure.* They cannot access:

- any of the resources of the client workstation
- the client workstation's memory
- the network.

Here is an example of an applet tag:

```
<applet code="myapplet" archive="applets.jar">
</applet>
```

The ″applet code″ tag specifies the name of the main Java class of the applet, the optional ″archive″ tag specifies one or more archives where the class library (and all other required classes) are located. You can store the class and JAR files belonging to the applet in any directory of the middle-tier.

This chapter contains these main sections:

- "How Applets Are Used in 2-Tier Environments" on page 202
- "How Applets Are Used in 3-Tier Environments" on page 203
- "How the VSEAppletServer Is Used" on page 205
- "Disadvantages and Restrictions Of Using Applets" on page 206
- "Running the Sample Data-Mapping Applet" on page 207
- "Running the Sample VSAM Applet" on page 216
- "Running the Sample DL/I Applet" on page 230

© Copyright IBM Corp. 2000, 2003

How Applets Are Used in 2-Tier Environments

Figure 100 shows how an applet is used within the VSE/ESA 2-*tier environment*:

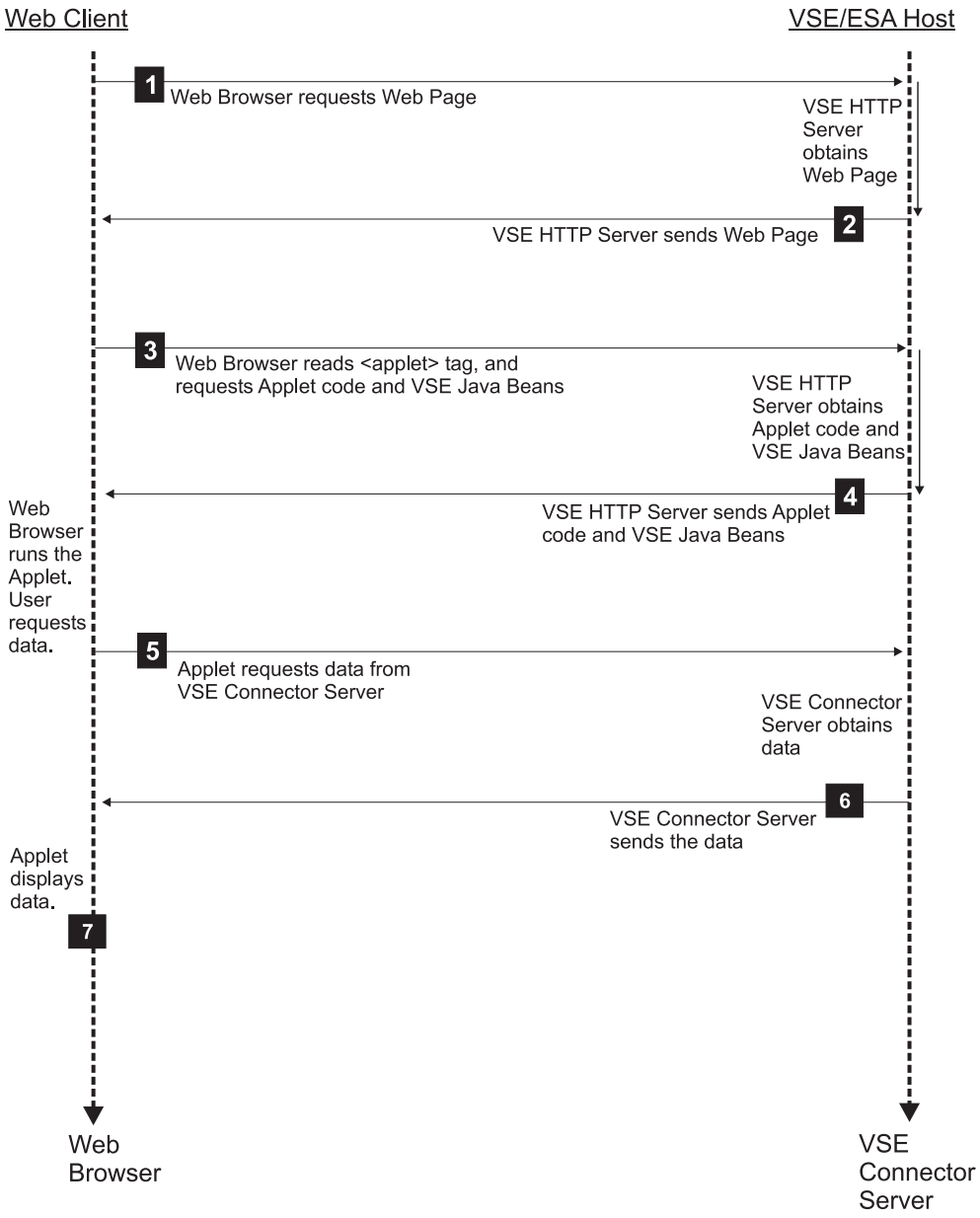


Figure 100. How Applets Are Used in the VSE/ESA 2-Tier Environment

HTTP Sessions are used between the Web Client and the VSE/ESA host for sending and receiving data.

The number of each list item below describes a step shown in Figure 100:

1. The client's Web browser requests an HTML page from the VSE HTTP Server running on the VSE/ESA host.
2. The VSE HTTP Server sends the Web Page to the client's Web browser.
3. The client's Web browser reads an <applet> tag and requests the applet code from the VSE HTTP Server on the VSE/ESA host. The applet code is

stored in one or more JAR files. The Web browser also requests the VSE Java Beans class library (**VSEConnector.jar**) from the VSE HTTP Server.

4 The VSE HTTP Server sends the applet code to the client's Web browser, together with the VSE Java Beans class library.

5 The client's Web browser runs the applet. The applet uses the VSE Java Beans class library (**VSEConnector.jar**) to build a connection to the VSE Connector Server. The end-user requests data that is stored on the VSE/ESA host. For an applet in a 2-tier environment, the data can be VSE/POWER, VSE/VSAM, VSE/ICCF, or Librarian data.

Note: An applet *cannot* obtain DB2, DL/I, or CICS data in a 2-tier environment. This is because the VSE Connector Server cannot access these systems. In addition, the use of MQSeries Servers is not possible in 2-tier environments.

6 The VSE Connector Server obtains the required data using "native" calls (using the standard access methods), and then sends the data to the client's Web browser via TCP/IP.

7 The client's Web browser runs the applet a second time, and displays the Web Page together with the requested data.

Although applets are mainly used in 2-tier environments, you can also use an applet within the *3-tier environment* providing you implement a "router" on the middle-tier, that serves as a gateway between the client and the VSE/ESA host. For details, see "How the VSEAppletServer Is Used" on page 205.

How Applets Are Used in 3-Tier Environments

Figure 101 on page 204 shows how an applet is used within the VSE/ESA *3-tier environment*. The number of each list item below describes a step shown in this figure.

1 The client's Web browser requests an HTML page from the IBM HTTP Server running on the middle-tier.

2 The IBM HTTP Server sends the Web Page to the client's Web browser.

3 The client's Web browser reads an <applet> tag and requests the applet code from the IBM HTTP Server running on the middle-tier. You can store the class and JAR files belonging to the applet in any directory of the middle-tier. The Web browser also requests the VSE Java Beans class library (**VSEConnector.jar**) from the IBM HTTP Server.

4 The IBM HTTP Server sends the applet code to the client's Web browser, together with the VSE Java Beans class library.

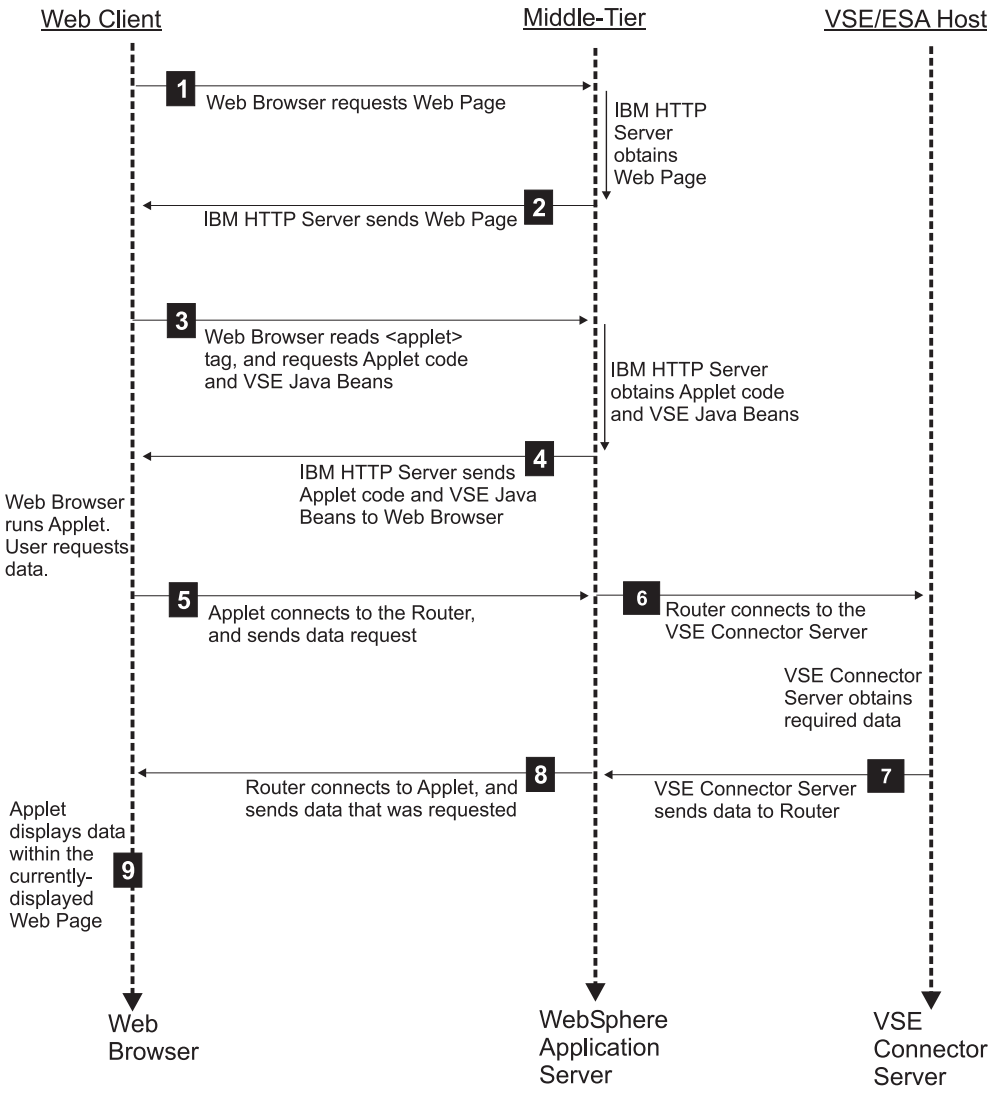


Figure 101. How Applets Are Used in the VSE/ESA 3-Tier Environment

5 The client's Web browser runs the applet, and the end-user requests data that is stored on the VSE/ESA host. The applet connects to the router on the middle-tier, and sends the request for data to the router. The router is the *VSEAppletServer* (described in "How the VSEAppletServer Is Used" on page 205) running on the middle-tier.

6 The router connects to the VSE Connector Server running on the VSE/ESA host, and forwards the request for data to it.

7 The VSE Connector Server retrieves the data using "native" calls (the standard access method), and sends the data back to the *VSEAppletServer* router running on the middle-tier (via TCP/IP).

Notes:

1. The VSE Connector Server can be used for accessing VSE/VSAM, VSE/POWER, VSE/ICCF, or Librarian data.
2. An alternate method for accessing VSAM data stored on the VSE/ESA host, is to use a DB2 Stored Procedure on the middle-tier which

communicates directly with the VSAM file system on the VSE/ESA host. This is described in "Using DB2 Stored Procedures to Access VSAM Data" on page 312.

8 The router connects to the applet running in the client's Web browser, and sends the data to the applet (also via TCP/IP).

9 The applet running inside the client's Web browser displays the data within the currently-displayed Web Page.

HTTP Sessions are used between the Web Client and the middle-tier for sending and receiving data. Connect Sessions are used between the middle-tier and the VSE/ESA host for sending and receiving data.

How the VSEAppletServer Is Used

Applets, by nature, have many restrictions, including:

- They can open a new network connection only to the platform from which they are downloaded. In a 3-tier environment this is the middle-tier.
- They can only access the file system of the platform from which they are downloaded. In a 3-tier environment this is a file system stored on the middle-tier.

To get around these problems and allow applets to get data from the VSE Connector Server running on the VSE/ESA host, a simple router (*VSEAppletServer*) is provided. The applet simply connects to this router. The router, which does not have this restriction, then connects to the VSE Connector Server to get VSE-based data, and pass it back to the applet.

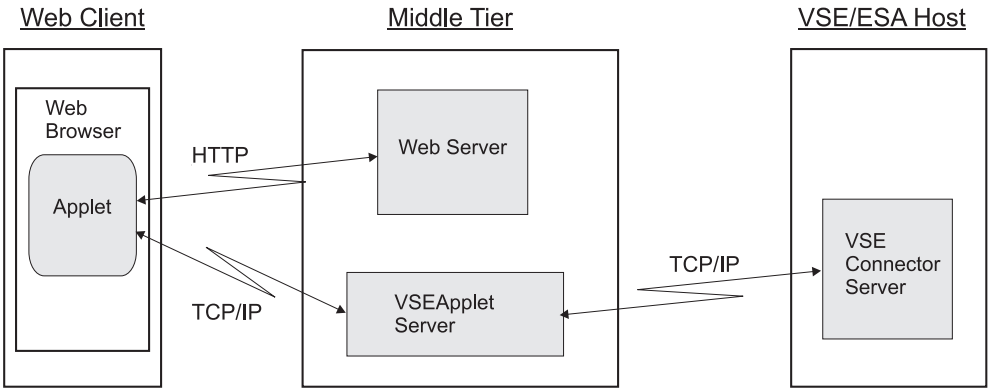


Figure 102. How the VSEApplet Server Is Used in the 3-Tier Environment

Disadvantages and Restrictions Of Using Applets

These are the disadvantages and restrictions when using applets in the VSE/ESA environment:

- Since the applet must run in a Web browser, you do not have the advantage of having an extremely fast middle-tier machine of the 3-tier environment. The speed of the process is instead dependent on the resources of each individual Web browser, and the network speed and bandwidth.
- You cannot use the MQSeries Java client as the connector for an applet, because the MQSeries server on VSE can only communicate with another MQSeries Server, and not directly with the MQSeries Java client.
- You can only use the **archive** tag with Netscape, or with the Microsoft Internet Explorer 4 (or later), not with the Microsoft Internet Explorer 3. Check the documentation belonging to your Web browser for details about supported applet parameters.
- You must store class and jar files as *binary* in the VSE library system. You can use the Librarian **LD** command to check the file format.
- When you write *applets*, you should ***never*** "hard-code" any user IDs and passwords in the applet code: when the applet is downloaded to a Web browser and is stored in the Web browser's cache, this information could possibly be displayed by unauthorized persons.

Running the Sample Data-Mapping Applet

The information in this section is based upon a sample applet, the *VSAM data-mapping applet* (also referred to as simply the *data-mapping applet*) that is provided with the VSE Connector Client.

Other examples of applets are similarly provided with the VSE Connector Client, such as the:

- *VsamSpaceUsage* (which displays the used and free VSAM space).
- *DB2ConnectorJDBCApplet* (which calls a DB2 Stored Procedure to access VSAM data via the VSAMSQL *Call Level Interface*, abbreviated to *CLI*). For details, see "Running the Sample VSAM Applet" on page 216.
- *VSAM Applet* (described on page 216).
- *DL/I Applet* (described on page 230).

Description of the Data-Mapping Applet

The data-mapping applet describes a Java applet that allows you to create and maintain maps and views for specific VSAM files. It does not show you how to display or modify VSAM data itself. To display or modify VSAM data, you must either use:

- an applet with the same functionality as the sample VSAM applet (described in "Example of How to Implement a Servlet" on page 246).
- a servlet (described in "Example of How to Implement a Servlet" on page 246).

Note!
The data-mapping applet *might not run* with certain combinations of operating system and Web browser.

The applet performs this general processing:

1. The applet displays some window controls, which allow the user to enter the IP address of a VSE/ESA host, a VSE user ID, and a password.
2. The applet connects to the VSE Connector Server, and retrieves a list of VSAM catalogs.
3. By double-clicking on an item in the catalog list, a list with all clusters in this catalog is displayed.
4. By double-clicking on a cluster, a list showing all maps defined to this cluster is displayed.
5. By double-clicking on a map, two lists are displayed. The first list shows all data fields of this map, the second list shows all views of this map.
6. By double-clicking on a view, a list with all fields of this view is displayed.

Figure 103 on page 208 shows the applet running in the Web browser window.

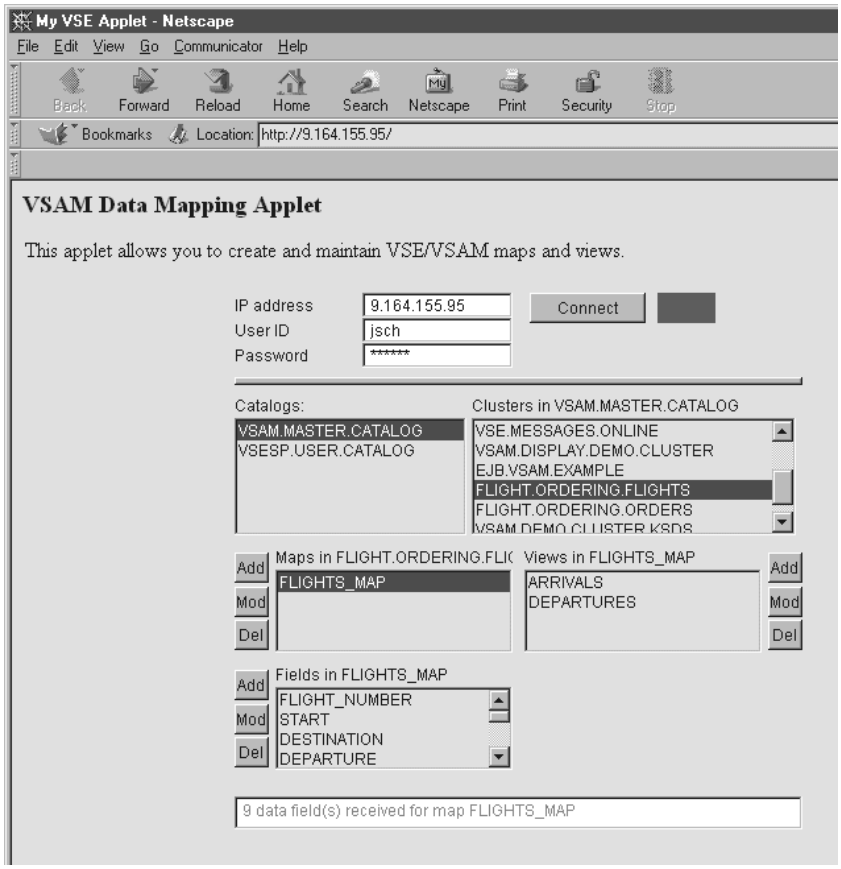


Figure 103. Window for VSAM Data-Mapping Applet

The VSAM file FLIGHT.ORDERING.FLIGHTS contains data records describing flights. Each flight consists of data fields (DEPARTURE, ARRIVALS, SEATS, and so on). These data fields are contained in the map FLIGHTS_MAP that describes the complete record.

Using push-buttons that are displayed next to the lists shown in Figure 103, you can add, change, or delete, the maps, views, and fields. By clicking a push button to add or modify a map, view, or field, a dialog is displayed that allows you to enter new data.

The map has two different data views, DEPARTURES and ARRIVALS, that provide subsets of the data fields of the map. All maps and views are stored in a VSAM file which contains the VSAM mapping definitions. For details, see "How Maps Are Stored on the VSE/ESA host" on page 130.

Activities Required on the VSE/ESA Host

Before you can run the VSAM data-mapping applet, you must perform these activities on your VSE/ESA host:

1. Define a Web server on VSE/ESA, by entering this TCP/IP command on the VSE/ESA console:

   ```
   DEFINE HTTPD,ID=MYHTTPD,ROOT=PRIMARY.TEST
   ```

   This will start an HTTP daemon with root library PRIMARY.TEST.

2. Create a file with the name **index.html**, that will be read when a Web browser connects to the VSE/ESA host. Here is the **index.html** file you should use with the sample applet:

```
<html>
<head>
<title>VSAM Data Mapping Example Applet</title>
</head>
<body>
<h2>VSAM Data Mapping Applet</h2>
This applet can be used to create and maintain VSAM maps and views.
Please logon to your VSE host. When the connection is established, a
list of VSAM catalogs is displayed.
<p>
<center>
<applet code="com.ibm.vse.samples.VsamMappingApplet" width=440 height=420
        archive="applets.jar, vsecon.jar">  1
</applet>
</center>
</body>
</html>
```

1 The archive tag specifies the names of the Java archives (.JAR) containing all classes that are required to run the applet. In this example:

- File **applets.jar** contains the applet-specific code.
- File **vsecon.jar** is identical to **VSEConnectors.jar**, except that it has been given a short file name so it can be placed in a VSE library.

3. Place file **index.html** in the HTTP server's *root* library.

Deploying the Data-Mapping Applet

To deploy the data-mapping applet, you must:

1. From the **vsecon\samples** directory, compile the Java sources, and create the JAR archive. To do so, use the following statements:

```
call javac com\ibm\vse\samples\VsamMappingApplet.java
call javac com\ibm\vse\samples\VsamAppletListener.java
call jar c0fv applets.jar com\ibm\vse\samples\VsamMappingApplet.class
                          com\ibm\vse\samples\VsamAppletListener.class
```

2. Send the created JAR archive to the VSE HTTP server's root directory (in *binary* format). To do so, you can either use an emulator program, or FTP. The JAR utility is part of your local Java installation. For details of how to use FTP, see page 25.

Calling the Data-Mapping Applet

To run the data-mapping applet (or any other applet), it must be:

1. Downloaded from the VSE/ESA host to a local workstation.
2. Executed in the Java Virtual Machine of a Web browser installed on the local workstation.

If the VSE HTTP Server is running, to display the above HTML file, you simply enter the IP address or symbolic name of your VSE/ESA host in the Web browser's *address/location* field. To start the VSE HTTP Server, you enter this TCP/IP command at the VSE/ESA console:

```
xx q httpds
```

where *xx* is the reply-ID of the TCP/IP partition.

How Various Web Browsers Search for JAR and Class Files

- The *Netscape Communicator* does not search in the local classpath. Instead, it always takes the classes from the path specified in the applet's archive tag.
- The *Microsoft Internet Explorer 3* cannot process the archive tag.
- The *Microsoft Internet Explorer 4 and 5* place your local system classpath in front of the path of JAR files that is specified in the applet's archive tag. As a result, if you have same classes locally, the VSE/ESA host-based classes will not be loaded!. The Web browser will instead load the *local* classes.

Setting Up the Data-Mapping Applet Class

In general, an applet extends the Java applet class (supplied with the Java Development Kit). The provided data-mapping applet however, also implements an *ActionListener* that detects push-button actions.

```
/* Import applet classes */
import java.applet.*;
...

public class VsamMappingApplet extends Applet
implements ActionListener, ItemListener
{
   VSESystem system;
   VSEConnectionSpec spec;
   VSEVsam vsam;
   VSEVsamCatalog catalog;
   VSEVsamCluster cluster;
   VSEVsamMap map;
   VSEVsamView view;
   VSEVsamField field, newField;
   VsamAppletListener vl;
   ...
```

Figure 104. Data-Mapping Applet Code for Setting Up the Java Class

Initializing the Data-Mapping Applet

The *init()* method is called from the Web browser when the applet is first started. In this example:

1. The logon controls are displayed.
2. A frame is created that is required as the parent frame of the various dialogs for adding, modifying, or deleting maps, views, and data fields.

```
public void init()
{
   f = new Frame();
   pgl = new PowerGridLayout(100, 66); // a box with 100 x 66 units  1
   setLayout(pgl);
   vMapFields = new Vector();
   vViewFields = new Vector();
   displayLogonDialog();       // show the dialogbox
   repaint();
}
```

Figure 105. Sample Code for Initializing the Data-Mapping Applet

1 The *PowerGridLayout* class is a Java layout manager that is much used in

the *VSE Navigator* function, and some examples contained in the VSE Connector Client. The *PowerGridLayout* class is contained in the **com.ibm.vse.utilities** package.

Re-Displaying or Leaving an HTML Page

The applet *start()* method is called from the Web browser whenever the HTML page is to be re-displayed:

```
public void start()
{
}
```

The applet *stop()* method is called from the Web browser when leaving the HTML page:

```
public void stop()
{
}
```

Using the Data-Mapping Applet to Add a Map to a VSAM Cluster

This code shows how to define a *VSEVsamMap* using the data-mapping applet. The same basic method is also used for defining views and fields:

1. A local map object is created using its constructor.
2. The local map object is then created on the VSE/ESA host, using the *create()* method.

Accessing the VSE/ESA host can produce an *IOException* or a *ConnectorException* error. Therefore this call must be contained in a "try-catch" clause.

```
public int addMap()
{
   ...

   /* Get map name from textfield on dialogbox */
   String name = tfName.getText().toUpperCase();

   /* Create local object */
   map = new VSEVsamMap(   1
           system,
           ((VSEVsamCatalog)(vCatalogs.elementAt(catIndex))).getFileID(),
           ((VSEVsamCluster)(vClusters.elementAt(cluIndex))).getFileID(),
           name);

   /* Create map on host */
   try {
      map.create();   2

      /* Add new mapname to map list ... */
      ...
   }
   catch (Exception e)
   {
      ...
   }
   ...
}
```

Figure 106. Data-Mapping Applet Code for Adding a Map to a VSAM Cluster

Running the Data-Mapping Applet

The number below refer to the numbers in Figure 106 on page 211:

1. Here a new local instance of class *VSEVsamMap* is created. At this stage, no host action has been performed.
2. The *create()* method is used to create the map definition on the host.

Using the Data-Mapping Applet to Modify a Map

A map object can be renamed using the *rename()* method, which sends a request to the VSE/ESA host to change the map's name. Accessing the VSE/ESA host can produce an *IOException* or a *ConnectorException* error. Therefore this call must be contained in a "try-catch" clause.

```
public int modMap()
{
   ...
   /* Get name from textfield ... */
   String name = tfName.getText().toUpperCase();
   try {
      /* Change map on host */
      map.rename(name);
      ...
   }
   catch (Exception e)
   {
      ...
   }
   ...
}
```

Figure 107. Sample Applet Code for Modifying a Map

This dialog window allows you to change a map's properties:

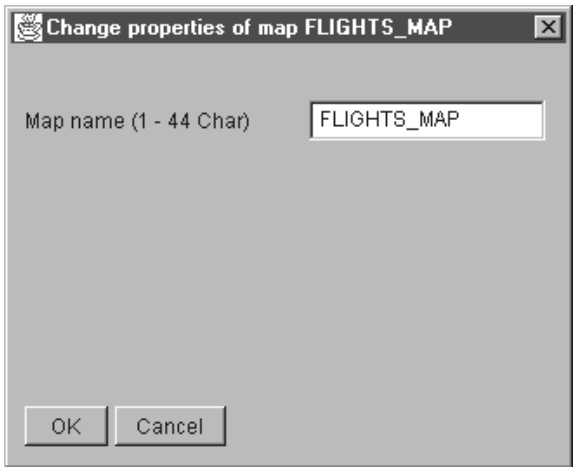


Figure 108. Window for Changing a Map's Properties

Using the Data-Mapping Applet to Modify a Map's Data Fields

The example below shows how to modify a field contained within a map, in these steps:

1. The map is retrieved by searching for the selected item in the list of maps.
2. Map-related methods (such as *setFieldName()*, *setFieldType()*, and so on) are used to change the properties of a field.
3. After modifying the field on the host, the list of local map fields is updated with the new name. In addition, the local field object is updated in the vector of map fields.

```
public int modMapfield()
{
   int i,type;
   ...
   /* Get new name from dialogbox */
   String name = tfName.getText().toUpperCase();

   /* Check if this name is already there. It's possible */
   /* to leave the name unchanged, but it's not possible */
   /* to change the name to another existing name. */
   ...

   /* Get other values from dialogbox */
   if (cb1.getState() == true)
      type = VSEVsamMap.TYPE_STRING;
   else if (cb2.getState() == true)
      type = VSEVsamMap.TYPE_BINARY;
   else if (cb3.getState() == true)
      type = VSEVsamMap.TYPE_PACKED;
   else if (cb4.getState() == true)
      type = VSEVsamMap.TYPE_SIGNED;
   else
      type = VSEVsamMap.TYPE_UNSIGNED;
   String len = tfLength.getText().toUpperCase();
   String offset = tfOffset.getText().toUpperCase();

   ...
   /* Get related map */
   i = mapList.getSelectedIndex();
   map = (VSEVsamMap)(vMaps.elementAt(i));    1
   ...
```

Figure 109. Sample Applet Code for Modifying a Map's Data Fields (Part 1 of 2)

```
   /* Modify this field on host */
   try {
     map.setFieldName(oldName, name);
     map.setFieldType(map.getIndex(name), type);
     map.setFieldLength(map.getIndex(name),new Integer(len).intValue());
     map.setFieldOffset(map.getIndex(name),new Integer(offset).intValue());
   }
   catch (Exception e)
   {
     ...
   }

   /* Modify local field */
   i = mapFieldList.getSelectedIndex();
   mapFieldList.replaceItem(name, i);
   field = (VSEVsamField)(vMapFields.elementAt(i));
   field.setName(name);
   field.setType(type);
   field.setLength(new Integer(len).intValue());
   field.setOffset(new Integer(offset).intValue());
   ...
}
```

Figure 109. Sample Applet Code for Modifying a Map's Data Fields (Part 2 of 2)

1 Here, we must keep local map, view, and field instances in a vector, because it is not possible to store such objects in an AWT list.

This dialog window corresponds to the code shown above, and allows you to modify a map's data fields:

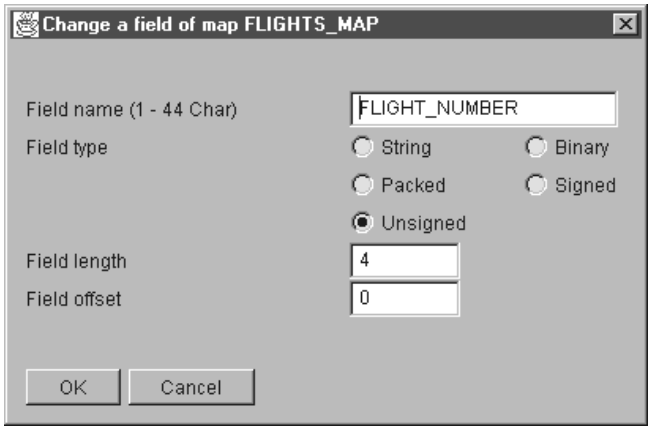


Figure 110. Window for Changing a Map's Data Fields

Note: Checking for valid length and offset values are not performed. You can also overwrite fields (specify the same offset but different lengths for different fields).

Running the Data-Mapping Applet Locally Using the AppletViewer

For a quick test of your installation, you can run the data-mapping applet *locally*: you are not required to first place the code in the VSE/ESA host. The VSE Connector Client online documentation contains a file **DataMapping.html**, which you can use to run the sample applet using your local appletviewer. The appletviewer is supplied with your Java installation.

```
<html>
<body>
<h2>VSAM Data Mapping Applet</h2>
<applet code="com.ibm.vse.samples.VsamMappingApplet"
        codebase="." archive="../VSEConnector.jar"
        width=460 height=460>
</applet>
</body>
</html>
```

Please note that:

- The codebase and archive tags are different from those of the VSE/ESA host-based HTML file. Here, we specify using the codebase tag, that all applet-related code is located in the current directory (including sub-directories).
- The archive tag points to the original **VSEConnector.jar** file that contains the VSE Java Beans class library.
- You are not required to create a second JAR file containing the applet-specific code, since the applet viewer can take these classes directly from your local file system.

Go to the **vsecon\samples** directory and call the applet in this way:

```
set classpath=.;..\VSEConnector.jar;%classpath%
AppletViewer  MappingApplet.html
```

(which assumes the **VSEConnector.jar** file is stored in the next upper directory).

The related Unix shell script would therefore be as follows:

```
#! /bin/sh
export CLASSPATH=.:../VSEConnector.jar:$CLASSPATH
appletviewer MappingApplet.html
```

Running the Sample VSAM Applet

The information in this section is based upon a sample applet, the *VSAM applet*, that is provided with the VSE Connector Client.

Other examples of applets are similarly provided with the VSE Connector Client, such as the:

- *VsamSpaceUsage* applet (which displays the used and free VSAM space).
- *Data-Mapping Applet* (described on page 207).
- *DL/I Applet* (described on page 230).

Description of the VSAM Applet

The sample VSAM applet is an example of how you can use an applet together with the DB2-Based Connector. It allows you to display and modify VSAM data stored in a sample VSAM data cluster.

When you start the VSAM applet, it connects to the DB2 Connect database-alias **db2vsewm**. The VSAM applet can then communicate with the VSE/ESA host database **sqlds**, *via* **db2vsewm**.

The VSAM applet calls a sample DB2 Stored Procedure (VSAMSEL in this example) to access VSAM data via the VSAMSQL Call Level Interface (CLI). For a detailed description of the CLI, see "Using DB2 Stored Procedures to Access VSAM Data" on page 312.

Before you can run the VSAM applet, you must have customized:

- The DB2-Based Connector (see Steps 1 to 6 of Chapter 9, "Customizing the DB2-Based Connector", on page 71 for details).
- The sample DB2 Stored Procedures (see "Step 7: Customize the DB2-Based Connector for VSAM Data Access" on page 83 for details).
- DB2 Connect on your middle-tier (see "Step 10: Install DB2 Connect and Establish Client-Host Connection" on page 85 for details). During this step, you define **sqlds** to DB2 Connect on your middle-tier (where **sqlds** is the sample database on the VSE/ESA host that is used with the VSAM applet).

Note!
The VSAM applet *might not run* with certain combinations of operating system and Web browser.

How the sample VSAM applet is used within a 3-tier environment, is shown in Figure 111 on page 217:

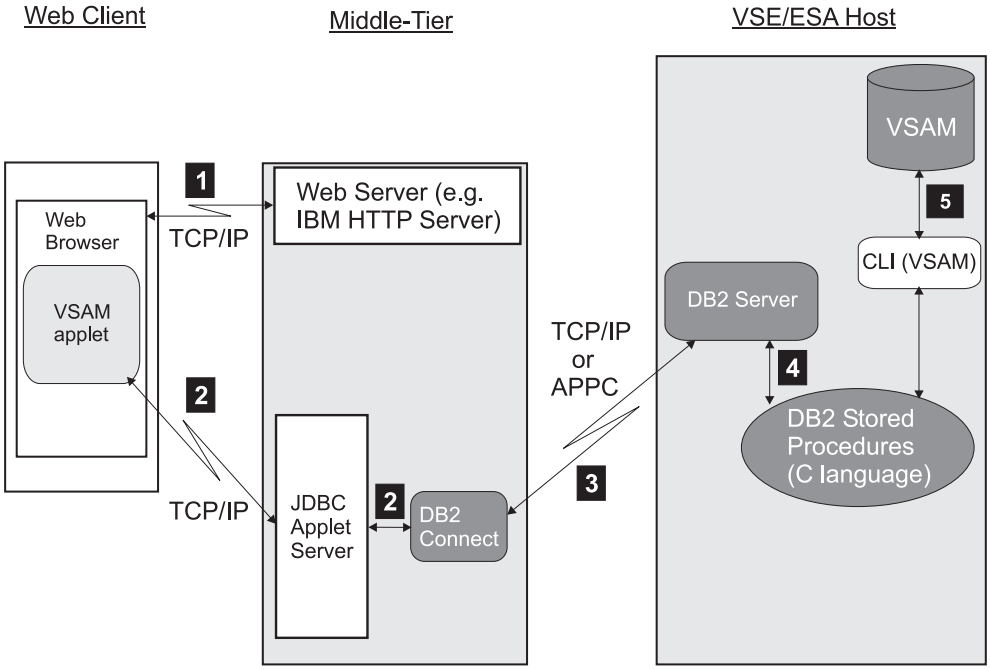


Figure 111. Using the Sample VSAM Applet to Access VSAM Data

1 The clients's Web browser requests an HTML page from the IBM HTTP Server (or another Web server) running on the middle-tier. The HTML page contains the applet tag for the *VSAM applet*. The VSAM applet is loaded into the Java Virtual Machine of the Web browser, and starts to run.

2 The VSAM applet opens a connection to DB2 Connect running on the middle-tier of the 3-tier platform. It does so *via the JDBC Applet Server* (where "JDBC" is an abbreviation for *Java Database Connectivity*). The use of the JDBC Applet Server overcomes the restrictions that applets can only:

- Open a new network connection to the platform from which they are downloaded (in this case, the middle-tier).
- Access the file system of the platform from which they are downloaded (in this case, the middle-tier).

The VSAM applet then calls a sample DB2 Stored Procedure (VSAMSEL in this example).

3 DB2 Connect communicates with the DB2 Server for VSE, via the DB2 Connect database-alias **db2vsewm**. DB2 Connect can now access to the VSE/ESA host database **sqlds**. It uses the DRDA (Distributed Relational Database Architecture). The underlying protocol can be either APPC or TCP/IP.

4 The DB2 Server for VSE executes a sample DB2 Stored Procedure (VSAMSEL in this example), using the Stored Procedure Server.

5 The sample DB2 Stored Procedure can now carry out the VSAM applet's request, by accessing the VSAM data stored on the VSE/ESA host via VSAMSQL CLI.

Notes:

1. The WebSphere Application Server is not required on the middle-tier of the process described above.
2. The VSE Connector Server is not required on the VSE/ESA host of the process described above.

Getting Started With the Sample VSAM Applet

Before you can run the VSAM applet, you must perform the steps described in this section.

1. Create an HTML File to Call the VSAM applet

You must write an HTML file, from which the VSAM applet can be called. Each time this HTML page is displayed, the applet will be loaded and executed in the Web browser's JVM (Java Virtual Machine). The VSAM applet then connects to the middle-tier database using JDBC.

Here is an example of such an HTML file:

```
<html>
<head>
<title>JDBC Example Applet to call a DB2 Stored Procedure</title>
</head>
<body>
<h2>JDBC Example Applet that access VSE/VSAM using the DB2-based connector</h2>
This applet lets you browse, insert, update and delete any records from the
VSE/VSAM sample cluster for the DB2-based connector. You can also browse all VSAM records
using the view OFFER. The JDBC Applet calls the corresponding Stored Procedures defined
within your DB2 Server for VSE.
<p>
<center>
<applet code="DB2ConnectorJDBCApplet.class" archive="db2applt.jar, db2java.zip"
           width=440 height=420>
</applet>
</center>
</body>
</html>
```

2. Compile VSAMSEL.C

Because DB2 Stored Procedures that use the VSAMSQL CLI are written in *C language*, you compile VSAMSEL.C (the sample DB2 Stored Procedure used in this example) using the IBM LE/VSE *C for VSE* compiler. All DB Stored Procedures must also be LE/VSE-compliant. Use job stream SKCPSTP, which is located in ICCF library 59.

The sample DB2 Stored Procedures do not contain SQL statements, since they use instead the VSAMSQL Call Level Interface (CLI). Therefore, you are not required to run the SQL precompiler. However, if you decide to include both SQL statements and VSAMSQL CLI to access your own data, you must run an additional SQL precompile step.

Here is the JCL (taken from SKCPSTP) that you can use to compile the *VSAMSEL.C* sample DB2 Stored Procedure. You compile the other sample DB2 Stored Procedures (VSAMINS.C, VSAMUPD.C, and VSAMDEL.C) in a similar way.

```
// JOB SKCPSTP COMPILE SAMPLE STORED PROCEDURE
// DLBL SYSMSGS,'CVSE.COMP.MSGS',0,VSAM,RECSIZE=3000,RECORDS=35,       X
               CAT=VSESPUC
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.DBASE,PRD1.BASE)
// SETPARM CATALOG=1
// IF CATALOG = 1 THEN
// GOTO CAT
// OPTION ERRS,SXREF,SYM,LIST,NODECK
// GOTO ENDCAT
/. CAT
// LIBDEF PHASE,CATALOG=LIB.SUBLIB
// OPTION ERRS,SXREF,SYM,NODECK,CATAL
   PHASE VSAMSEL,*
/. ENDCAT
```

```
 INCLUDE @@TRT
 INCLUDE IESVSQLO
// EXEC EDCCOMP,SIZE=EDCCOMP,PARM='LONGNAME RENT SS SOURCE            X
               INFILE(DD:PRD1.BASE(VCLIUTIL.C))'
// EXEC EDCCOMP,SIZE=EDCCOMP,PARM='LONGNAME RENT SS SOURCE            X
               INFILE(DD:PRD1.BASE(VSAMSEL.C))'
/*
// IF CATALOG NE 1 OR $MRC GT 4 THEN
// GOTO NOLNK
// EXEC EDCPRLK,SIZE=EDCPRLK,PARM='NATLANG(ENU)/UPCASE'
/*
// EXEC LNKEDT,SIZE=256K
/. NOLNK
/&
```

3. Define VSAMSEL to the DB2 Server for VSE

Now you must make VSAMSEL (the DB2 Stored Procedure used in this example) known to your DB2 Server for VSE system. To do so, you must:

1. Place the VSAMSEL *phase* (compiled in the previous step) into a library that is contained in the *Stored Procedure Server's* search path.
2. Use the CREATE PROCEDURE statement to define VSAMSEL to the database manager. You can use the job stream SKCRESTP located in ICCF library 59 for this purpose.

   Here is the CREATE PROCEDURE statement for VSAMSEL:

```
CREATE PROCEDURE VSAMSEL (IN char(180),INOUT INT,OUT INT,OUT CHAR(20),-
OUT CHAR(20),OUT CHAR(20),OUT INT,OUT INT,OUT CHAR(20),OUT CHAR(20),-
OUT CHAR(20),OUT INT,OUT INT,OUT CHAR(20),OUT CHAR(20),OUT CHAR(20),-
OUT INT,OUT INT,OUT CHAR(6),OUT INT,OUT CHAR(252)) EXTERNAL,LANGUAGE C,-
STAY RESIDENT YES,SERVER GROUP,PARAMETER STYLE GENERAL
```

4. Define the VSAM Data Cluster

You use job SKVSSAMP (located in ICCF library 59) to:

- Create the VSAM data cluster used by the sample VSAM applet.
- Load sample records into the VSAM data cluster.
- Create a sample map and view for the VSAM data cluster, using the RECMAP command. For detailed information about RECMAP, see "Defining a Map Using RECMAP" on page 132.

```
// JOB SKVSSAMP LOAD VSE/VSAM CONNECTOR SAMPLE DATA CLUSTER
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
*                                                                     *
*   NOTE: IF YOU SPECIFY A DIFFERENT CATALOG THAN                     *
*         VSESP.USER.CATALOG, YOU HAVE TO CHANGE THE PATH FOR         *
*         THE RECORD MAP (CATALOG/CLUSTER/MAP/VIEW) IN THE            *
*         CLIENT PROGRAMS THAT CALL THE STORED PROCEDURE AS WELL      *
*         (CVSAMSEL.SQC ETC.)                                         *
*                                                                     *
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
*
*   DEFINING THE VSAM CONNECTOR SAMPLE DATA CLUSTER 'VCSAMPD'
*
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
// EXEC IDCAMS,SIZE=AUTO
 DELETE VSAM.CONN.SAMPLE.DATA PURGE CATALOG(VSESP.USER.CATALOG)
 DEFINE CLUSTER ( -
        NAME ( VSAM.CONN.SAMPLE.DATA ) -
        RECORDS ( 30 30 ) -
        SHAREOPTIONS ( 2 ) -
        RECORDSIZE ( 120  120 ) -
        VOLUMES ( DOSRES SYSWK1 ) -
        NOREUSE -
        INDEXED -
        FREESPACE ( 15 7 ) -
```

```
        KEYS ( 4   0 ) -
        NOCOMPRESSED ) -
        DATA ( NAME ( VSAM.CONN.SAMPLE.DATA.@D@ ) -
        CONTROLINTERVALSIZE ( 4096  ) ) -
        INDEX ( NAME ( VSAM.CONN.SAMPLE.DATA.@I@ )) -
        CATALOG (VSESP.USER.CATALOG )
   IF LASTCC NE 0 THEN CANCEL JOB
/*
// OPTION STDLABEL=DELETE
                 VCSAMPD
/*
// OPTION STDLABEL=ADD
// DLBL VCSAMPD,'VSAM.CONN.SAMPLE.DATA',,VSAM,                          X
              CAT=VSESPUC
/*
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
*                                                                     *
*               NOW LOADING THE SAMPLE DATA                           *
*                                                                     *
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
// LIBDEF *,SEARCH=(PRD1.BASE)
// EXEC VSAMSMPD,SIZE=AUTO
// ON $RC>0 GOTO FINISH
/*
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
*                                                                     *
*    DEFINE MAPS AND VIEWS USING THE RECORD MAPPING UTILITY           *
*                                                                     *
* $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
// EXEC IDCAMS,SIZE=AUTO
 RECMAP DEFINE ( MAP( USEDCARS )  -
   MAPCOLUMN( -
    ( ARTICLENO    FIELD( O(0)   L(4)   T(SINTEG))  POS(1) )   -
    ( MANUFACTURER FIELD( O(4),  L(20)  T(STRING))  POS(2) )   -
    ( TYPE         FIELD( O(24)  L(20)  T(STRING))  POS(3) )   -
    ( MODEL        FIELD( O(44)  L(20)  T(STRING))  POS(4) )   -
    ( HP           FIELD( O(64)  L(2)   T(SINTEG))  POS(5) )   -
    ( DISPLACEMENT FIELD( O(66)  L(2)   T(SINTEG))  POS(6) )   -
    ( CYLINDERS    FIELD( O(68)  L(2)   T(SINTEG))  POS(7) )   -
    ( COLOUR       FIELD( O(70)  L(20)  T(STRING))  POS(8) )   -
    ( FEATURES     FIELD( O(90)  L(20)  T(STRING))  POS(9) )   -
    ( PRICE        FIELD( O(110) L(4)   T(SINTEG))  POS(10))   -
   ) -
 ) -
 CATALOG( VSESP.USER.CATALOG )  -
 CLUSTER( VSAM.CONN.SAMPLE.DATA )
 RECMAP DEFINE ( MAP( USEDCARS )  -
   VIEW( OFFER ) -
    VIEWCOLUMN( ( ARTICLENO     REFCOLUMN( ARTICLENO    )) -
     ( MANUFACTURER  REFCOLUMN( MANUFACTURER )) -
     ( TYPE          REFCOLUMN( TYPE         )) -
     ( MODEL         REFCOLUMN( MODEL        )) -
     ( PRICE         REFCOLUMN( PRICE        )) -
   ) -
 ) -
 CATALOG( VSESP.USER.CATALOG )  -
 CLUSTER( VSAM.CONN.SAMPLE.DATA )
 RECMAP LIST (CLUSTERS)
/*
/. FINISH
/*
/&
```

5. Create the JAR File for the VSAM applet

Before running the VSAM applet, you must:

1. Create a JAR file by copying the applet-related class files into this JAR (Java Archive) file. To do so, go to the samples directory of your VSE Connector Client installation, and execute these statements:

   ```
   call jar c0fv db2applt.jar com\ibm\vse\db2\DB2ConnectorJDBCApplet.class
      com\ibm\vse\db2\MessageDialog.class
      com\ibm\vse\db2\PowerGridLayout.class
      com\ibm\vse\db2\PowerGridLayoutInfo.class
   ```

2. Copy the JAR file of (1.) above, to the HTML directory of your Web Server (for example the IBM HTTP server, or Apache server) on the middle-tier platform.

   Note: Because the VSAM applet runs in a *3-tier environment*, as an alternative to (1.) above, you could copy the class files directly to the HTML directory of your Web Server. This is because the applet code is not stored on the VSE/ESA host, and you are therefore not required to store class files there in a short-name archive.

3. Start the JDBC applet server on the middle-tier platform. This server handles the requests that are initiated by the VSAM applet. You must also choose an unused TCP/IP port number that can be used by the JDBC applet server (all ports are defined in the *services* file). Therefore if you choose TCP/IP port **6789** (the default), you would enter:

   ```
   db2jstrt 6789
   ```

Calling the VSAM Applet

When the HTML page is loaded, the VSAM applet is:

1. Downloaded from the middle-tier server to a local workstation.
2. Executed in the Java Virtual Machine of a Web browser installed on the local workstation.

The VSAM applet can be called in two ways:

- Using the applet viewer directly (file **AppletViewer.exe** for Windows and OS/2), which is part of your local Java installation. To do so, from a command prompt you simply enter:

  ```
  AppletViewer db2index.html
  ```

  where **db2index.html** contains the HTML tags shown in "1. Create an HTML File to Call the VSAM applet" on page 218.
- Using a Web browser. To do so, you must enter the symbolic name or IP address of your middle-tier platform, followed by the name of the sample HTML file in your Web browser's address field. In our example, you would enter:

  ```
  http://ebusvse/db2/db2index.html
  ```

After being called, the VSAM applet displays the main window shown in Figure 112 on page 222.

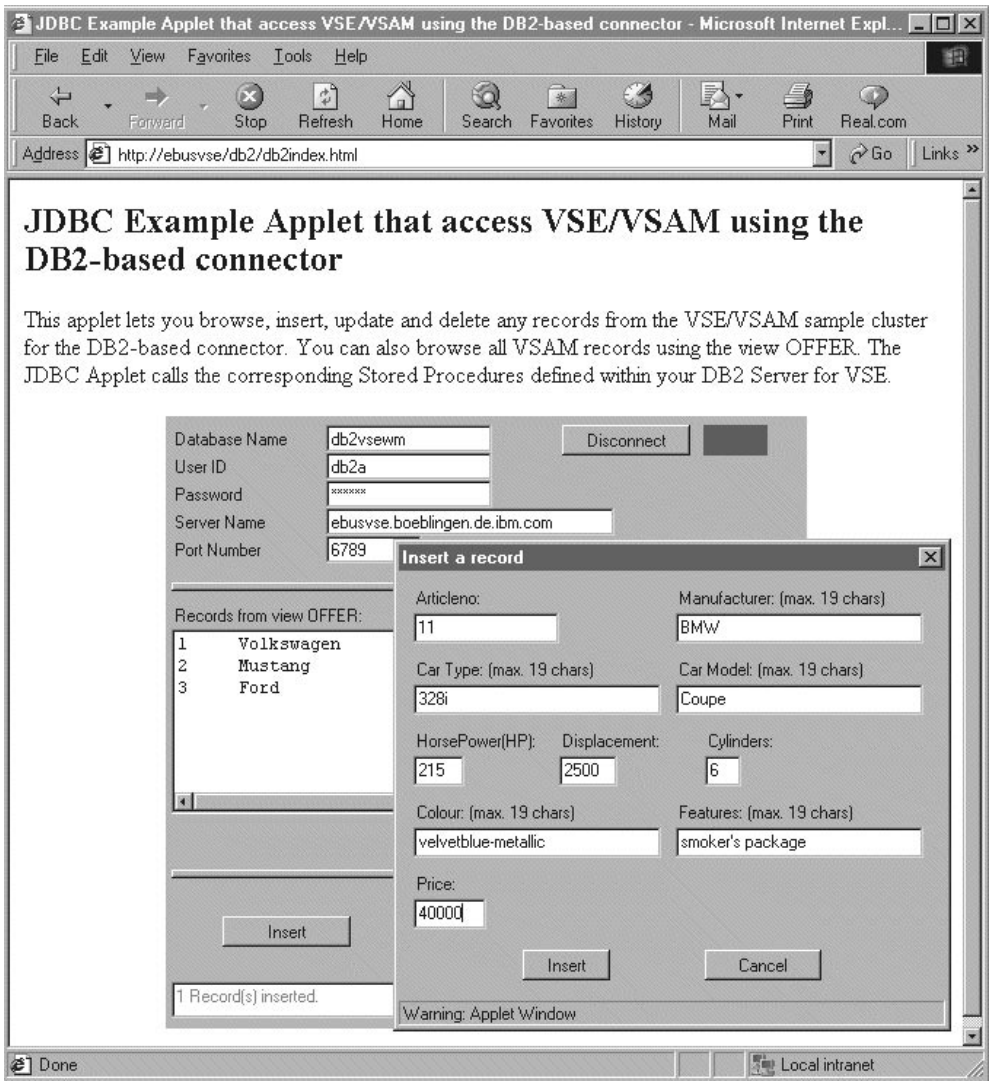


Figure 112. Window Displayed by the Sample VSAM Applet

Figure 112 includes a rectangle positioned to the right of the **Disconnect** button. This indicates the status of the connection between the VSAM applet and the **sqlds** sample database (which is accessed using the DB2 Connect database-alias **db2vsewm**). When this rectangle is:

- *green* the applet is connected to **sqlds**, and the button displays **Disconnect**.
- *red* the applet is not connected to **sqlds**, and the button displays **Connect**.

The main window shown in Figure 112 includes an *Insert a record* dialog window, which you use to insert a VSAM record in the sample VSAM cluster. You can use other similar dialog windows to refresh, update, or delete, records in the sample VSAM data cluster. Depending upon the dialog window that is currently displayed:

If You Press	Then ...
Connect	The connection to VSE/ESA is reestablished via DB2 Connect, and then DB2 Stored Procedure **VSAMSEL** is used together with the **OFFER** view, to select all records from a sample VSAM data cluster.

Insert	The DB2 Stored Procedure **VSAMINS** is used to insert a new record into the sample VSAM data cluster.
Update	The DB2 Stored Procedure **VSAMUPD** is used to replace a record in the sample VSAM data cluster.
Delete	The DB2 Stored Procedure **VSAMDEL** is used to delete a VSAM record from the sample VSAM data cluster.
Refresh	The DB2 Stored Procedure **VSAMSEL** is used, together with the **OFFER** view, to select all records from a sample VSAM data cluster.

At the bottom of Figure 112 on page 222 is a "status line", which displays error or status messages related to the action you are currently performing.

The sample VSAM data cluster is described in "4. Define the VSAM Data Cluster" on page 219.

Description of DB2ConnectorJDBCApplet.java (the Client-Side Program)

This section describes the *main steps* of *DB2ConnectorJDBCApplet.java*, which is used for most of the VSAM applet's functions. It runs in the Web browser's JVM of Figure 111 on page 217.

In addition, the VSAM applet uses these *helper classes*:
- *MessageDialog.java*, which displays a message-dialog window containing a specific row of text, together with an **OK** button.
- *PowerGridLayout.java* and *PowerGridLayoutInfo.java* which are a type of Java layout manager, and are much used by the VSE Connector Client samples. This Java layout manager is supplied in the *com.ibm.vse.utilities* package. You can use this Java layout manager when writing your own applications.

Step 1. Import the JDBC (Java Database Connectivity) Classes

In the first step, the JDBC classes are imported. JDBC is required for calling VSAMSEL (one of the sample DB2 Stored Procedures) that is used with the DB2 Server for VSE.

In addition, the applet-specific classes are imported. Since an applet generally *extends* the Java Applet class, the VSAM applet implements the:
- *ActionListener*, that listens to mouse-clicks and push-button actions.
- *WindowListener*, that handles window actions.

```
 ...
/* import JDBC classes */
import java.sql.*;

/* Import AWT classes */
import java.awt.*;
import java.awt.event.*;

/* Import applet classes */
import java.applet.*;


public class DB2ConnectorJDBCApplet extends Applet
implements ActionListener, WindowListener
{
```

Step 2. Load the Required JDBC Driver Class

In the second step, the required JDBC driver class is loaded. This is normally done in a static section. The *net* driver class is used, which ensures that the applet can run on any client workstation.

```
...
// register the JDBC driver with DriverManager
static
{
   try
   {
      Class.forName ("COM.ibm.db2.jdbc.net.DB2Driver");
   }
   catch (Exception e)
   {
      System.out.println ("\n  Error loading DB2 Driver...\n");
      e.printStackTrace ();
   }
} // end static block
...
```

Step 3. Implement the init() Method

In the third step, the *init()* method is implemented. The *init()* method is called from the Web browser when the VSAM applet is first started. A frame is created, that is the parent of the various dialogs used for refreshing, inserting, updating, or deleting, VSAM records in the sample VSAM data cluster (described in "4. Define the VSAM Data Cluster" on page 219).

```
...
 public void init()
 {
    /* Create a frame that is needed for the dialogs */
    /* to insert/update/delete records and to display message dialogs */
    f = new Frame();

    msgDialog = new MessageDialog(f, true);
    pgl = new PowerGridLayout(100, 66);
    setLayout(pgl);

    displayMainDialog();
    repaint();
 }
 ...
```

Step 4. Establish the Connection to VSE/ESA Database via DB2 Connect

In the fourth step, a connection is established to the VSE/ESA host database **sqlds**, via the DB2 Connect database-alias **db2vsewm**. DB2 Connect can then route database requests to the VSE/ESA host.

The following JDBC URL format is used to set up the connection:

```
<protocol>:<subprotocol>://<hostname or tcpip address>:<port number>/<database name>
```

For the database specified under <*database name*>, the JDBC *getConnection()* call supplies:

- the JDBC URL
- a user ID
- a password

The *getConnection()* call retrieves these values from the text fields of the Main Window's *connect* section.

```
...

public void connectToDB()
{
   String url    = "jdbc:db2://" + DBServerAddr.getText() + ":"
                     + DBServerPort.getText() + "/"
                     + DBName.getText();
   ...
   try
   {
      // connect with user-provided username and password
      con = DriverManager.getConnection(url, userid.getText(), passw.getText());
   }
   catch (SQLException sqlExc)
   {
      ...
   }
   ...
}
```

Step 5. Call VSAMSEL

In the final step, VSAMSEL (the sample DB2 Stored Procedure used in this example) is called.

VSAMSEL is also called when the end-user presses the **Refresh** button.

```
...
/**
 * call DB2-based connector Stored Procedure on VSE/ESA
 * This will retrieve all records from view OFFER, which then will be
 * put in the corresponding listbox of the applet.
 * The Stored Procedure VSAMSEL uses the VSAMSQL CLI to access the VSAM
 * records.
 */
 public void callStpVSAMSEL()
 {
    ...
 }
```

For details of how JDBC issues a call to a DB2 Stored Procedure, refer to the JDBC application samples supplied with the VSE Connector Client online documentation. (The online documentation is described in “Using the Online Documentation Options” on page 28).

Description of VSAMSEL

This section describes the *main steps* of VSAMSEL (the sample DB2 Stored Procedure used in this example) that runs on the VSE/ESA host of Figure 111 on page 217. The other sample DB2 Stored Procedures (VSAMINS, VSAMUPD, and VSAMDEL) are not described in this section.

VSAMSEL is called when either the **Connect** or **Refresh** buttons of Figure 112 on page 222 are pressed, and it demonstrates how VSAM data is accessed via the VSAMSQL CLI (Call Level Interface).

Step 1. Include Header File iesvsql.h in VSAMSEL

In the first step, the header file **iesvsql.h** is included in VSAMSEL. This header file is required by *all* DB2 Stored Procedures that use the VSAMSQL CLI interface to access VSAM data clusters.

Header file **iesvsql.h**:

- Maintains all function prototypes and VSAMSQL CLI definitions.
- Contains the prototype for the *check_error()* function, which provides examples of error-handling routines for VSAMSQL CLI calls. The *check_error()* function is called after each VSAMSQL CLI call.

```
...
// include VSAMSQL CLI
#include "iesvsql.h"
// include error utility functions
#include "vcliutlh.h"
...
```

Step 2. Initialize the VSAMSQL CLI Environment

In the second step, the *VSAMSQL CLI* (Call Level Interface) environment is initialized. The VSAMSQL CLI is described in "Using DB2 Stored Procedures to Access VSAM Data" on page 312.

These *handles* are initialized:

- The *environment handle*, which provides access to global information (such as valid connection handles and active connection handles).
- The *connection handle*, which provides access to connection information (such as the valid statement and descriptor handles on the connection).
- The *statement handle*, which provides access to statement information (such as error messages and status information for VSAMSQL statement processing).

```
....

/*************************************************************/
/*      initialize VSAMSQL CLI Environment                   */
/*************************************************************/
// allocate Environment
rc = VSAMSQLAllocHandle(VSAMSQL_HANDLE_ENV,VSAMSQL_NULL_HANDLE, &hEnv);
cont = check_error(VSAMSQL_HANDLE_ENV,hEnv,rc,
                   o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-1");

//allocate Connection
if (cont == STP_CONT) {
   rc = VSAMSQLAllocHandle(VSAMSQL_HANDLE_DBC,hEnv,&hDBC);
   cont = check_error(VSAMSQL_HANDLE_DBC,hDBC,rc,
                      o_sqlstate,o_message,&o_native_error,SYSLST,VSAMSEL-2");
} // end if

//allocate Statement
if (cont == STP_CONT) {
```

```
   rc = VSAMSQLAllocHandle(VSAMSQL_HANDLE_STMT,hDBC,&hStmt);
   cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
                      o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-3");
} // end if
...
```

Step 3. Initiate the Read of the VSAM Records

In the third step, the query statement to initiate the read of the VSAM records is prepared and executed. All fields contained in the view *i_recordmap* are selected (the view *OFFER* is passed from the client program within this example).

During the third step:

1. The *VSAMSQLPrepare()* function call associates the SELECT statement string with the statement handle.
2. *VSAMSQLBindParameter()* binds *i_lastkey* to the WHERE clause within the SELECT statement (where **?** is the corresponding placeholder).
3. *i_lastkey* is passed as an input parameter for VSAMSEL (the sample DB2 Stored Procedure).
4. *VSAMSQLExecute()* executes the statement, after it was successfully prepared. The statement handle *hStmt* qualifies the query statement.

```
...

/*****************************************************************/
/*      prepare and execute query statement                      */
/*****************************************************************/
if (cont == STP_CONT) {
   sprintf(vsamsqlstmt, "SELECT * FROM %s WHERE ARTICLENO > ? ",
           i_recordmap);
   rc = VSAMSQLPrepare(hStmt, vsamsqlstmt, VSAMSQL_NTS);
   cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
                      o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-4");
} // end if

// Bind local Variables to Stmt
if (cont == STP_CONT) {
   rc = VSAMSQLBindParameter(hStmt,1,VSAMSQL_PARAM_INPUT,
                             VSAMSQL_C_LONG,VSAMSQL_INTEGER,
                             0,0,&i_lastkey,0,NULL);
   cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
                      o_sqlstate,o_message,&o_native_error,SYSL"VSAMSEL-5");

} // end if

if (cont == STP_CONT) {
   rc = VSAMSQLExecute(hStmt);
   cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
                      o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-6");
} // end if
...
```

Step 4. Obtain the Results of the Query Statement

In the fourth step, since the query statement has executed, the results can now be fetched. But primarily all columns from the result set must be associated with a local variable (*tmp.articleno*). This is done using *VSAMSQLBindCol()* function calls, which must be issued for each column in the result set.

Within the *for()-loop*, a single record from the view *OFFER* is retrieved using *VSAMSQLFetch()*.

VSAMSEL can fetch and return three result records to the client program:

1. If less than three result records exist, condition VSAMSQL_NO_DATA_FOUND appears.
2. The *VSAMSQLFetch()* function call outside the *for()-loop* checks if more than three result records exist.
3. If more than three result records exist, *o_resultrows* is increased by one. This value is used as an indicator by the client program to determine if VSAMSEL should be called, in order to retrieve the next records.

```
...

/*****************************************************************/
/*      retrieve result set                                      */
/*****************************************************************/
// bind columns to local variables and retrieve results
if (cont == STP_CONT) {
   // bind parameter articleno
   rc = VSAMSQLBindCol(hStmt,1,VSAMSQL_C_LONG,
                       &tmp.articleno,0,&buf_len);
   cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
                      o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-7");
} // end if
...

// fetch result row(s)
for (i=0; i < NUM_ROWS; i++)
{
   if (cont == STP_CONT) {
      rc = VSAMSQLFetch(hStmt);
      cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
                     o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-C");
   } // end if

   // if successful - store retrieved columns
   if ( cont == STP_CONT && rc != VSAMSQL_NO_DATA_FOUND) {
      o_resultrows ++;
      o_parm[i].articleno = tmp.articleno;
      strcpy(o_parm[i].manufacturer,tmp.manufacturer);
      strcpy(o_parm[i].type,tmp.type);
      strcpy(o_parm[i].model,tmp.model);
      o_parm[i].price     = tmp.price;
   } // end if
   ...
} // end for

// check if more result exist
if (cont == STP_CONT) {
   rc = VSAMSQLFetch(hStmt);
   cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
                   o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-D");
   if (rc != VSAMSQL_NO_DATA_FOUND)
      o_resultrows ++;
} // end if
...
```

Step 5. Deallocate the VSAMSQL CLI Environment

In the fifth step, since all VSAMSQL CLI processing has been completed, the VSAMSQL CLI environment can be de-allocated using the function *VSAMSQLFreeHandle()*. This is done in the reverse order to the allocation step ("Step 2. Initialize the VSAMSQL CLI Environment" on page 226).

```
                              ...
/*************************************************************/
/*      free VSAMSQL CLI Environment                         */
/*************************************************************/
// free handle Statement
rc = VSAMSQLFreeHandle(VSAMSQL_HANDLE_STMT,hStmt);
cont = check_error(VSAMSQL_HANDLE_STMT,hStmt,rc,
                   o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-E");

// free handle Connection
rc = VSAMSQLFreeHandle(VSAMSQL_HANDLE_DBC,hDBC);
cont = check_error(VSAMSQL_HANDLE_DBC,hDBC,rc,
                   o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-F");

// free handle Environment
rc = VSAMSQLFreeHandle(VSAMSQL_HANDLE_ENV,hEnv);
cont = check_error(VSAMSQL_HANDLE_ENV,hEnv,rc,
                   o_sqlstate,o_message,&o_native_error,SYSLST,"VSAMSEL-10");
...
```

Step 6. Assign Local Output Variables to Host Output Variables

In the final step, the local output variables that are to be returned to the client program, must be assigned to the corresponding host output variables.

```
...
/*************************************************************/
/*  assign number rows with valid data, if >=4 more data exists */
/*************************************************************/
*(VSAMSQLINTEGER *)argv[2] = o_resultrows;       // rows returned

// copy result rows to output parameters
j=0;
for (i=0; i < NUM_ROWS; i++)
{
   *(VSAMSQLINTEGER *)argv[j+3] = o_parm[i].articleno;
   strcpy(argv[j+4], o_parm[i].manufacturer);
   strcpy(argv[j+5], o_parm[i].type);
   strcpy(argv[j+6], o_parm[i].model);
   *(VSAMSQLINTEGER *)argv[j+7] = o_parm[i].price;
   j = j + 5;
} // end for
...
```

Running the Sample DL/I Applet

The information in this section is based upon a sample applet, the *DL/I applet*, that is provided with the VSE Connector Client.

Other examples of applets are similarly provided with the VSE Connector Client, such as the:

- *VsamSpaceUsage* applet (which displays the used and free VSAM space).
- *Data-Mapping Applet* (described on page 207).
- *VSAM Applet* (described on page 216).

Description of the DL/I Applet

The sample DL/I applet is an example of how you can use an applet together with the DB2-Based Connector, and allows you to display and modify DL/I data stored in a sample DL/I database.

When you start the DL/I applet, it connects to the DB2 Connect database-alias **db2vsewm**. The DL/I applet can then communicate with the VSE/ESA host database **sqlds** via **db2vsewm**.

The DL/I applet then calls a sample DB2 Stored Procedure (in this example DLIREAD) to access DL/I data via the *AIBTDLI interface*. For a description of the AIBTDLI interface, "Overview of the AIBTDLI Interface" on page 321.

Before you can run the DL/I applet, you must have customized:

- The DB2-Based Connector (see Steps 1 to 6 of Chapter 9, "Customizing the DB2-Based Connector", on page 71 for details).
- The sample DB2 Stored Procedures (see "Step 7: Customize the DB2-Based Connector for VSAM Data Access" on page 83 for details).
- DB2 Connect on your middle-tier (see "Step 10: Install DB2 Connect and Establish Client-Host Connection" on page 85 for details). During this step, you define **sqlds** to DB2 Connect on your middle-tier (where **sqlds** is the sample database on the VSE/ESA host that is used with the VSAM applet).

Note!
The DL/I Applet *might not run* with certain combinations of operating system and Web browser.

How the sample DL/I applet is used within a 3-tier environment, is shown in Figure 113 on page 231:

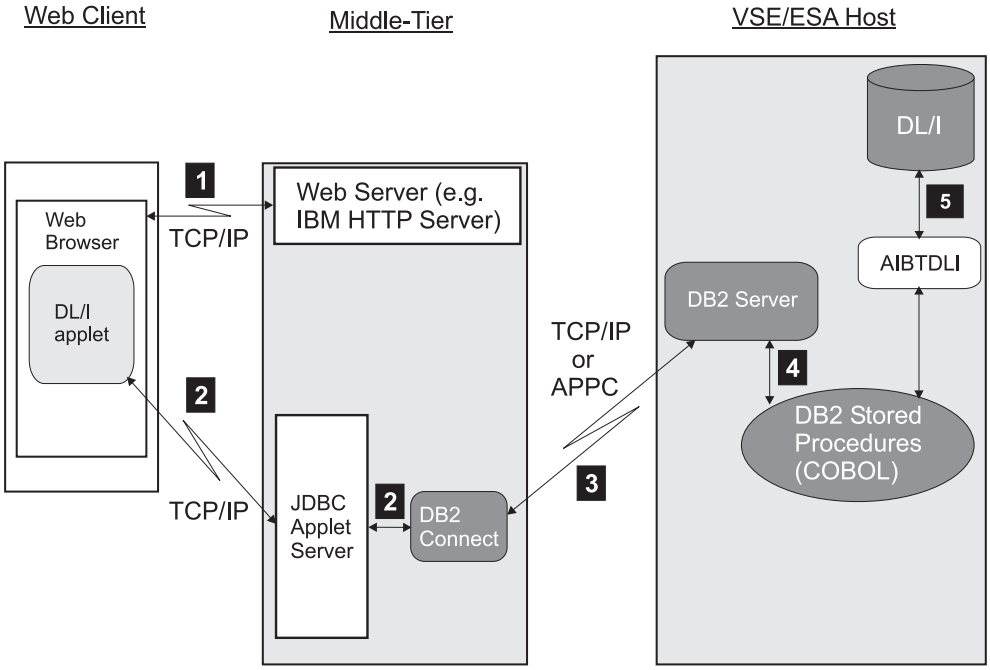


Figure 113. Using the Sample DL/I Applet to Access DL/I Data

1 The clients's Web browser requests an HTML page from the IBM HTTP Server (or another Web server) running on the middle-tier. The HTML page contains the applet tag for the *DL/I applet*. The DL/I applet is loaded into the Java Virtual Machine of the Web browser, and starts to run.

2 The DL/I applet opens a connection to DB2 Connect running on the middle-tier. It does so *via the JDBC Applet Server* (where "JDBC" is an abbreviation for *Java Database Connectivity*). The use of the JDBC Applet Server overcomes the restrictions that applets can only:

- Open a new network connection to the platform from which they are downloaded (in this case, the middle-tier).
- Access the file system of the platform from which they are downloaded (in this case, the middle-tier).

The DL/I applet then calls a sample DB2 Stored Procedure (DLIREAD in this example).

3 DB2 Connect communicates with the DB2 Server for VSE, via the DB2 Connect database-alias **db2vsewm**. DB2 Connect can now access the VSE/ESA host database **sqlds**. It uses the DRDA (Distributed Relational Database Architecture). The underlying protocol can be either APPC or TCP/IP.

4 DB2 Server for VSE manages the execution of a sample DB2 Stored Procedure (DLIREAD in this example), using the Stored Procedure Server.

5 The DB2 Stored Procedure can now execute the DL/I applet's request by accessing the DL/I data stored on the VSE/ESA host via the *AIBTDLI interface* (described in "Overview of the AIBTDLI Interface" on page 321).

Notes:

1. The WebSphere Application Server is not required on the middle-tier of the process described above.
2. The VSE Connector Server is not required on the VSE/ESA host of the process described above.

Getting Started With the Sample DL/I Applet

Before you can run the DL/I applet, you must perform the steps described in this section.

1. Create an HTML File to Call the DL/I applet

You must write an HTML file, from which the DL/I applet can be called. Each time this HTML page is displayed, the applet will be loaded and executed in the Web browser's JVM (Java Virtual Machine). The applet then connects to the middle-tier database using JDBC.

Here is an example of such an HTML file:

```
<html>
<head>
<title>
JDBC Example Applet that accesses DL/I using the DB2-based connector
</title>
</head>
<body>
<h2>JDBC Example Applet that accesses DL/I data</h2>
This applet lets you browse, insert, update and delete segments from/into the
DL/I inventory sample database via the DB2-based connector.
The JDBC Applet calls the corresponding Stored Procedures defined
within your DB2 Server for VSE using JDBC interface.
The Stored Procedures use the AIBTDLI interface to access the DLI/VSE data.
<p>
<center>
<applet code="DB2DLIConnectorJDBCApplet.class"
           archive="db2dliapplt.jar, db2java.zip"
           width=440 height=420>
</applet>
</center>
</body>
</html>
```

2. Compile DLIREAD.C

Because DB2 Stored Procedures that uses the AIBTDLI interface are written in *COBOL*, you compile DLIREAD.C (the DB2 Stored Procedure used in this example) using the IBM LE/VSE *COBOL* compiler. The compile job stream, SKDLICMP, is located in ICCF library 59.

DLIREAD.C does not contain SQL statements, since it uses instead the AIBTDLI interface. Therefore, you are not required to run the SQL precompiler. However, if you decide to include both SQL statements and DL/I calls to access your own data, you must run an additional SQL precompile step.

To compile the other sample DB2 Stored Procedures (DLIUPINS.C and DLIDEL.C) that are used for accessing DL/I data, use the same skeleton SKDLICMP.

3. Define DLIREAD to the DB2 Server for VSE

Now you must make DLIREAD known to your DB2 Server for VSE system. To do so, use the job skeleton SKDLISTP (located in VSE/ICCF library 59) to create DB2 Stored Procedures used for accessing the DL/I sample database.

1. Place the DLIREAD *phase* (compiled in the previous step) into a library that is contained in the *Stored Procedure Server's* search path.
2. Use the CREATE PROCEDURE statement to define DLIREAD to the database manager. You can use the job stream SKDLISTP located in ICCF library 59 for this purpose.

   Here is an example of the CREATE PROCEDURE statement for DLIREAD:

```
CREATE PROCEDURE DLIREAD (INOUT CHAR(6),OUT SMALLINT,
                          OUT CHAR(6),OUT CHAR(25),
                          OUT INT,OUT CHAR(6),OUT CHAR(6),
                          OUT CHAR(6),OUT CHAR(25),
                          OUT INT,OUT CHAR(6),OUT CHAR(6),
                          OUT CHAR(6),OUT CHAR(25),
                          OUT INT,OUT CHAR(6),OUT CHAR(6),
                          OUT CHAR(4),OUT CHAR(120))
                          EXTERNAL,LANGUAGE COBOL,
                          STAY RESIDENT YES,
                          SERVER GROUP,
                          PARAMETER STYLE GENERAL;
```

4. Define the DL/I Database

Before you can use DL/I applet, you must have defined a DL/I database. You use job SKDLISMP (located in ICCF library 59) to define and load the *sample* DL/I database.

5. Create the JAR File for the DL/I applet

Before running the DL/I applet, you must:

1. Create a JAR file by copying the applet-related class files into this JAR (Java Archive) file. To do so, go to the samples directory of your VSE Connector Client installation, and execute these statements:

   ```
   call jar c0fv db2dliapplt.jar com\ibm\vse\db2\DB2DLIConnectorJDBCApplet.class
      com\ibm\vse\db2\MessageDialog.class
      com\ibm\vse\db2\PowerGridLayout.class
      com\ibm\vse\db2\PowerGridLayoutInfo.class
   ```

2. Copy the JAR file of (1.) above to the HTML directory of your Web Server (for example the IBM HTTP server, or Apache server) on the middle-tier platform.

   Note: Because the DL/I applet runs in a *3-tier environment*, as an alternative to (1.) above, you could copy the class files directly to the HTML directory of your Web Server. This is because the applet code is not stored on the VSE/ESA host, and you are therefore not required to store class files there in a short-name archive.

3. Start the JDBC applet server on the middle-tier platform. This server handles the requests that are initiated by the DL/I applet. You must also choose an unused TCP/IP port number that can be used by the JDBC applet server. Therefore if you choose TCP/IP port **6789** (the default), you would enter:

   ```
   db2jstrt 6789
   ```

Calling the DL/I Applet

When the HTML page is loaded, the DL/I applet is:

1. Downloaded from the middle-tier server to a local workstation.
2. Executed in the Java Virtual Machine of a Web browser installed on the local workstation.

The DL/I applet can be called in two ways:

- Using the applet viewer directly (file **AppletViewer.exe** for Windows and OS/2), which is part of your local Java installation. To do so, from a command prompt you simply enter:

  ```
  AppletViewer index.html
  ```

  where **index.html** contains the HTML tags shown in "1. Create an HTML File to Call the DL/I applet" on page 232.

- Using a Web browser. To do so, you must enter the symbolic name or IP address of your middle-tier platform, followed by the name of the sample HTML file in your Web browser's address field. In our example, you would enter:

```
http://ebusvse/db2dli/index.html
```

After being called, the DL/I applet displays the main window shown in Figure 114.

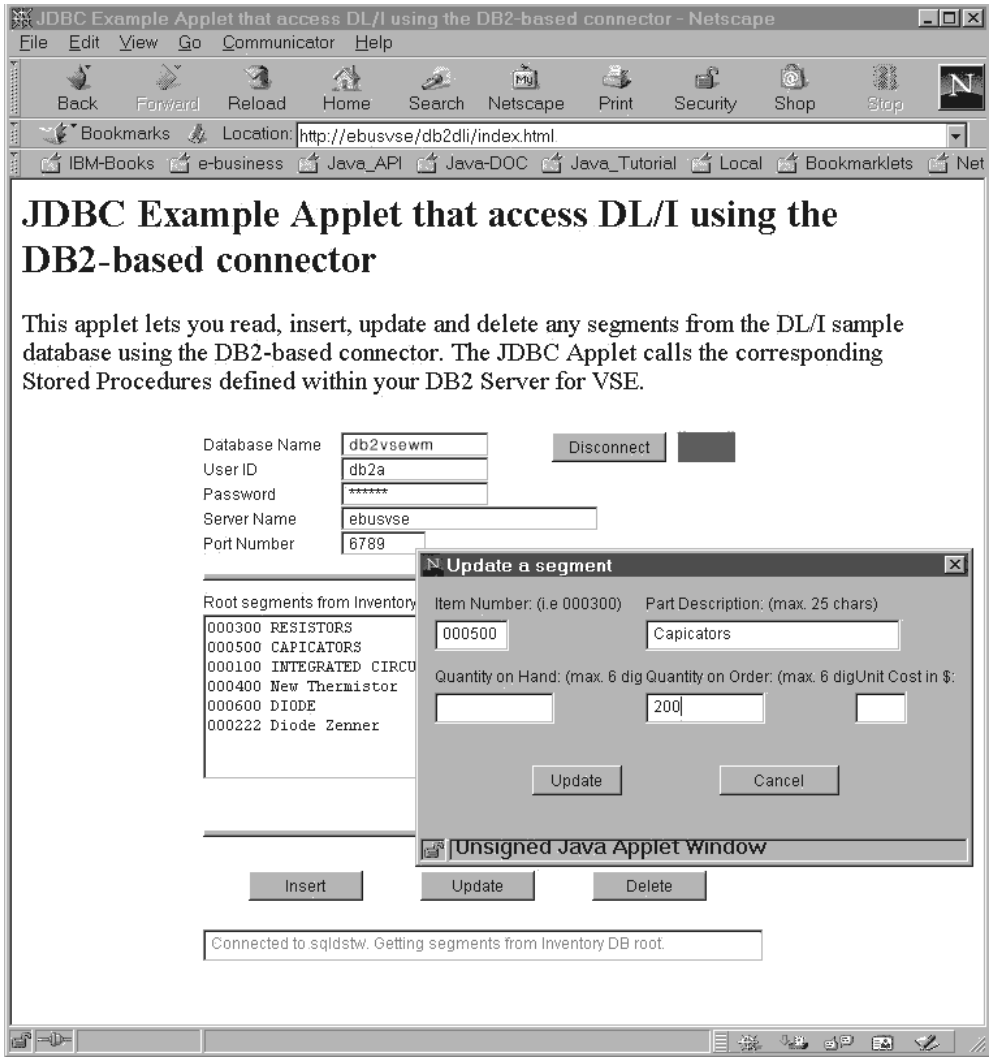


Figure 114. Window Displayed by the Sample DL/I Applet

Figure 114 includes a rectangle positioned to the right of the **Disconnect** button. This rectangle indicates the status of the connection between the DL/I applet and the **sqlds** sample database on the VSE/ESA host (which is accessed using the DB2 Connect database-alias **db2vsewm**). When this rectangle is:

- *green* the applet is connected to **sqlds**, and the button displays **Disconnect**.
- *red* the applet is not connected to **sqlds**, and the button displays **Connect**.

The main window shown in Figure 114 includes an *Update a segment* dialog window, which you use to insert a DL/I segment in the sample DL/I database.

You can use other similar dialog windows to refresh, insert, or delete, segments in the sample DL/I database. Depending upon the dialog window that is currently displayed:

If You Press	Then ...
Connect	The connection to VSE/ESA is re-established via DB2 Connect, and then the DB2 Stored Procedure **DLIREAD** is used to read all segments in the sample DL/I database.
Insert	The DB2 Stored Procedure **DLIUPINS** is used to insert a new segment in the sample DL/I database.
Update	The DB2 Stored Procedure **DLIUPINS** is used to update a segment in the sample DL/I database.
Delete	The DB2 Stored Procedure **DLIDEL** is used to delete a segment in the sample DL/I database.
Refresh	The DB2 Stored Procedure **DLIREAD** is used to read all segments in the sample DL/I database.

At the bottom of Figure 114 on page 234 is a “status line”, which displays error or status messages related to the action you are currently performing.

Description of DB2DLIConnectorJDBCApplet.java (the Client-Side Program)

This section describes the *main steps* of *DB2DLIConnectorJDBCApplet.java*, which is used for most of the DL/I applet’s functions. It runs in the Web browser’s JVM of Figure 113 on page 231.

In addition, method *callStpDLIREAD()* is the part of the DL/I applet that performs the Read/Browse of a DL/I database. All input and output variables must be set to the appropriate parameter placeholders (which are shown in the DL/I applet samples as question marks).

Step 1. Prepare the SQL Statement to Call DLIREAD

Before the first step can be carried out, a connection object *con* must exist to the VSE/ESA host. This connection object is opened using method *connectToDB()* (described in “Step 4. Establish the Connection to VSE/ESA Database via DB2 Connect” on page 224).

```
public void callStpDLIREAD()
{
   CallableStatement stmt; // SQL Statement Handle
   String sql =            // JDBC Stored Procedure Call String
       "Call " + name + "(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";

   // Prepare Stored Procedure Call Statement
   try
   {
      stmt = con.prepareCall(sql);
      // add header lines into applet list box
      RecordList.add("ITEMNO ITEM-DESCRIPTION          QUAN-H QUAN-O PRICE");
      RecordList.add("=================================================");
      do
      {
         //** set input variables **
         stmt.setString (1, nextkey);

         //** register output parameters **
         // next key returned from procedure
         stmt.registerOutParameter(1, Types.CHAR);
```

```
 stmt.registerOutParameter(2, Types.SMALLINT);

 // 3 data rows that can hold data from root segment of
 // inventory DB
for (int i=3; i <= 13; i=i+5)
{
   stmt.registerOutParameter ( i  , Types.CHAR);
   stmt.registerOutParameter ( i+1, Types.CHAR);
   stmt.registerOutParameter ( i+2, Types.INTEGER);
   stmt.registerOutParameter ( i+3, Types.CHAR);
   stmt.registerOutParameter ( i+4, Types.CHAR);
}

// Variables for error handling
stmt.registerOutParameter(18, Types.CHAR); // return code
stmt.registerOutParameter(19, Types.CHAR); // error message
```

Step 2. Call DLIREAD

In the second step, DLIREAD (the sample DB2 Stored Procedure used in this example) is called by executing the statement created in Step 1.

```
stmt.executeQuery();
```

Step 3. Check the Return Code from DLIREAD

In the third step, the return code is checked that was returned by DLIREAD.

```
// Get return code from Stored Procedure
ret_code = stmt.getString(18);

// Check if the Stored Procedure returned an error
if ((ret_code.compareTo("0000") == 0) ||
    (ret_code.substring(0,1).compareTo("G") == 0))
{
   // no error
   // get number of result rows (1-3)
   res_rows  = stmt.getShort(2);
```

Providing an error is not found, the output variables from DLIREAD are retrieved. The output parameters are read, and the results then displayed by the DL/I applet.

```
// Get returned fields from root segment
for (int i = 0, j = 3; i < 3;  i++, j=j+5)
{
   itemno[i]       = stmt.getString(j);
   itemdesc[i]     = stmt.getString(j+1);
   unit_cost[i]    = stmt.getInt(j+2);
   quan_hand[i]    = stmt.getString(j+3);
   quan_ord[i]     = stmt.getString(j+4);

   // add fields from root segment into listbox
   if (i < res_rows)
   {
      String temp = Integer.toString(unit_cost[i]);
      while(temp.length()<5)
         temp += ' ';
      RecordList.add(itemno[i] + " " + itemdesc[i]
          + " " + quan_hand[i] + " "
          + quan_ord[i] + " " + temp + "$ ");
   }
}
```

The program now:

1. Checks to see if there are further results. If there *are* further results, a DLIREAD call is made.

2. Uses the *nextkey* variable to check if there are further data result rows. If there *are* further data rows, DLIREAD is called again for the next 3 rows.

```
      nextkey    = stmt.getString(1);
      if (nextkey == "000000")
         moreresults = false;
      else  // more data available in the database
         moreresults = true;
   }
   else
   {
      // error occured
      // check if DB is empty
     if ((ret_code.substring(0,2) == "GB") &&
         (RecordList.getItemCount() == 0))
        setStatus("Inventory Database is empty!!");
     else {
        DLIerrmsg = stmt.getString(19);  // get DLI error message
        setStatus("AIBTDLI Return Code: 0x" + ret_code);
        msgDialog.setTitle("Stored Procedure Error");
        System.out.println("DLI Error: " + DLIerrmsg);
        msgDialog.setMessage("Check Java Console for DLZ messages.");
        msgDialog.setVisible(true);
     } // end if
   } // end if
 }
 while ( moreresults && ret_code.compareTo("0000") == 0);
```

Step 4. Reset the Connection to the sqlds Database

In the final step, whenever the end-user presses the **Disconnect** button of Figure 114 on page 234, the program resets the connection to the **sqlds** database located on the VSE/ESA host. **sqlds** is accessed via the DB2 Connect database-alias **db2vsewm**.

```
        if ((ret_code.compareTo("0000") == 0) ||
            (ret_code.substring(0,2).compareTo("GB") == 0))   // no error
        {
           setStatus(RecordList.getItemCount()-2 +
                    " segments retrieved from Inventory DB.");
        }

        // close Statement Handle
        stmt.close();
      }
      catch (SQLException sqlExc)   // handle SQL exceptions
      {
        closeConnection();
        ...
        return;
      }
    }
   ...
 }
 ...
```

For details of how to insert, update, or delete DL/I segments, you should refer to the complete DL/I applet source code, supplied with the VSE Connector Client online documentation. (The online documentation is described in "Using the Online Documentation Options" on page 28).

Description of DLIREAD

This section describes the *main steps* of DLIREAD, which is the sample DB2 Stored Procedure used in this example. It is written in COBOL and issues DL/I calls via the *AIBTDLI interface* (described in "Overview of the AIBTDLI Interface" on page 321). DLIREAD runs on the VSE/ESA host of Figure 113 on page 231, and is called when the **Connect** or **Refresh** buttons of Figure 114 on page 234 are pressed.

The complete source code is contained in file DLIREAD.COB, which is supplied with the VSE Connector Client. The other sample DB2 Stored Procedures (DLIUPINS and DLIDEL) are not described in this section.

Using one call to DLIREAD, you can read and return up to three segments from the sample DL/I database:

- The data is returned via DLIREAD's output parameters, where each data field of a DL/I segment has a corresponding output parameter.
- Other output parameters include *error-indicator variables*, that contain information about errors that might have occurred in DB2 or DL/I, or indicate the status of a request from a DB2 Stored Procedure.
- The DL/I applet contains a variable which it uses to check if there are further (more than three) DL/I segments waiting to be retrieved. If more than three records *are* to be retrieved, the client program calls DLIREAD again, using an ascending key.

Step 1. Define Variables for AIBTDLI, and I/O Area

In the first step, these are defined:

- The variables for the DL/I AIBTDLI interface.
- The I/O area used to communicate with DL/I.

```
...
*
*  UNQUALIFIED SSA FOR GETTING FIRST ROOT SEGMENT
*
   77  SSAUNQ        PIC X(9)  VALUE 'STPIITM  '.
*
*  QUALIFIED SSA FOR GETTING ROOT SEGMENT VIA KEY
*
   01  SSAQUAL.
     02  FILLER    PIC X(19) VALUE 'STPIITM (STQIINO = '.
     02  ROOTKEY   PIC X(6).
     02  FILLER    PIC X(1)  VALUE ')'.
*
* I-O AREA FOR RECEIVING ALL SEGMENTS
*
   01  IOAREA        PIC X(160).
   01  STPIITM         REDEFINES IOAREA.
     02  ITNUMB    PIC X(6).
     02  ITDESC    PIC X(25).
     02  IQOH      PIC X(6).
     02  IQOR      PIC X(6).
     02  FILLER    PIC X(6).
     02  IUNIT     PIC 9(6).
     02  FILLER    PIC X(105).
  LINKAGE SECTION.
```

Step 2. Define the Parameters for DLIREAD

In the second step, the parameters for DLIREAD are defined. These parameters build the interface between the DL/I applet, and the DB2 Server for VSE running on the VSE/ESA host.

In the example provided here, the parameters for three rows are defined. Therefore, up to three rows can be retrieved from the DL/I sample database, using a single call to DLIREAD.

```
 *
 * STORED PROCEDURE PARAMETERS
 *
   01  NEXT-KEY     PIC X(6).
   01  NUM-ROWS     PIC S9(4)   COMP.

 * FIRST RESULT ROW
   01  ITEM-NUMB1   PIC X(6).
   01  ITEM-DESC1   PIC X(25).
   01  UNIT-COST1   PIC S9(6)   COMP.
   01  QUAN-HAND1   PIC X(6).
   01  QUAN-ORD1    PIC X(6).

 * SECOND RESULT ROW
   01  ITEM-NUMB2   PIC X(6).
   01  ITEM-DESC2   PIC X(25).
   01  UNIT-COST2   PIC S9(6)   COMP.
   01  QUAN-HAND2   PIC X(6).
   01  QUAN-ORD2    PIC X(6).

 * THIRD RESULT ROW
   01  ITEM-NUMB3   PIC X(6).
   01  ITEM-DESC3   PIC X(25).
   01  UNIT-COST3   PIC S9(6)   COMP.
   01  QUAN-HAND3   PIC X(6).
   01  QUAN-ORD3    PIC X(6).
   01  RET-CODE     PIC X(4).
   01  DLI-ERR-MSG  PIC X(120).
   COPY DLIAIB.
   ...
```

Step 3. Define DLIREAD's Parameters to COBOL

In the third step, in the PROCEDURE DIVISION of DLIREAD, the parameters are defined that represent the external interface to DLIREAD.

```
  ....
 ******************************
 * BEGIN OF PROGRAMMING SECTION
 ******************************
   PROCEDURE DIVISION USING
             NEXT-KEY NUM-ROWS
             ITEM-NUMB1 ITEM-DESC1
             UNIT-COST1 QUAN-HAND1
             QUAN-ORD1
             ITEM-NUMB2 ITEM-DESC2
             UNIT-COST2 QUAN-HAND2
             QUAN-ORD2
             ITEM-NUMB3 ITEM-DESC3
             UNIT-COST3 QUAN-HAND3
             QUAN-ORD3
             RET-CODE DLI-ERR-MSG.
```

Step 4. Obtain the Results of the Query Statement

In the fourth step, the DL/I processing starts:

1. DLIREAD issues up to three DL/I calls, to retrieve the segments.
2. If there are no further segment to retrieve, after the third call has been made, the processing ends .
3. If there are result-rows to be retrieved, DLIREAD is called another three times.

```
 * ******************************************************
 * SCHEDULE THE PSB
 * ******************************************************
    SCHEDULE-PSB.
    MOVE 'STBICLG' TO PSB-NAME.
    CALL 'AIBTDLI' USING FUNCT-PCB, PSB-NAME, ADDRESS OF DLIAIB.
    IF AIBFCTR NOT = LOW-VALUES  GO TO BASERR
    SET ADDRESS OF PCB-PTRS TO AIBPCBAL.
    SET ADDRESS OF INV-L-PCB TO B-PCB1-PTRS.

 * ******************************************************
 * DO A GET NEXT CALL TO RETRIEVE THE FIRST ROOT SEGMENT
 * ******************************************************
    MOVE NEXT-KEY TO ROOTKEY.
    IF NEXT-KEY = '000000'
       CALL 'AIBTDLI' USING FUNCT-GN, INV-L-PCB, IOAREA, SSAUNQ
```

The calls to DL/I are made for a maximum of 3 segments. The values are returned to the DL/I applet via the corresponding parameters defined for the DLIREAD.

```
 ...
 * ******************************************************
 * DO A GET UNIQUE CALL TO RETRIEVE A SEGMENT VIA KEY
 * ******************************************************
    ELSE
      CALL 'AIBTDLI' USING FUNCT-GU, INV-L-PCB, IOAREA, SSAQUAL
    END-IF.
    IF AIBFCTR NOT = LOW-VALUES
      GO TO BASERR.

 * ******************************************************
 * ASSIGN VALUES TO OUTPUT PARAMETERS FOR ROW 1
 * ******************************************************
    ADD 1 TO COUNTR.
    MOVE ITNUMB  TO ITEM-NUMB1.
    MOVE ITDESC  TO ITEM-DESC1.
    MOVE IQOH    TO QUAN-ORD1.
    MOVE IQOR    TO QUAN-HAND1.
    MOVE IUNIT   TO UNIT-COST1.
    MOVE COUNTR TO NUM-ROWS.

Issue second call to the stored procedure to retrieve second result row:

 * ******************************************************
 * ISSUE GET NEXT CALL
 * ******************************************************
  CALL 'AIBTDLI' USING FUNCT-GN, INV-L-PCB, IOAREA,
          SSAUNQ.
  IF AIBFCTR NOT = LOW-VALUES
    GO TO BASERR.

 * ******************************************************
 * ASSIGN VALUES TO OUTPUT PARAMETERS FOR ROW 2
 * ******************************************************
   ADD 1 TO COUNTR.
   MOVE ITNUMB  TO ITEM-NUMB2.
   MOVE ITDESC  TO ITEM-DESC2.
```

```
  MOVE IQOH    TO QUAN-ORD2.
  MOVE IQOR    TO QUAN-HAND2.
  MOVE IUNIT   TO UNIT-COST2.
  MOVE COUNTR TO NUM-ROWS.
```

Below, the third (and final) call to DLIREAD is issued. The third result-row is retrieved.

```
 * ****************************************************
 * ISSUE GET NEXT CALL
 * ****************************************************
   CALL 'AIBTDLI' USING FUNCT-GN, INV-L-PCB, IOAREA, SSAUNQ.
   IF AIBFCTR NOT = LOW-VALUES
     GO TO BASERR.

 * ****************************************************
 * ASSIGN VALUES TO OUTPUT PARAMETERS FOR ROW 3
 * ****************************************************
   ADD 1 TO COUNTR.
   MOVE ITNUMB  TO ITEM-NUMB3.
   MOVE ITDESC  TO ITEM-DESC3.
   MOVE IQOH    TO QUAN-ORD3.
   MOVE IQOR    TO QUAN-HAND3.
   MOVE IUNIT   TO UNIT-COST3.
   MOVE COUNTR TO NUM-ROWS.
```

Step 5. Check for Further DL/I Segments

In the fifth step, a check is made to see if there are any further DL/I segments. The variable NEXTKEY is set to the next DL/I segment key that satisfies the initial condition.

```
 * ****************************************************
 *  DETERMINE IF FURTHER SEGMENTS EXIST AND SAVE KEY
 * ****************************************************
   CALL 'AIBTDLI' USING FUNCT-GN, INV-L-PCB, IOAREA, SSAUNQ.*

 * MORE SEGMENTS AVAIL - SAVE KEY FOR THE NEXT PROCEDURE CALL
 *
      MOVE INV-KEY-FBCK(1:6) TO NEXT-KEY.
      GO TO END-PROC.
```

Step 6. Run the Error-Handling Routines

In the final step, DLIREAD uses error-handling routines to determine if the request has been successful.

```
*
 *  DLI-CALL ERROR HANDING
  ...
```

Running the DL/I Applet

Chapter 20. Using Java Servlets to Access Data

This chapter contains these main sections:

- "How Servlets Are Used in 3-Tier Environments"
- "Compiling and Calling Servlets" on page 245
- "Example of How to Implement a Servlet" on page 246

How Servlets Are Used in 3-Tier Environments

Servlets are used in the middle-tier of a 3-tier environment shown in Figure 115 in this way (where this illustration assumes that a VSE Connector Server running on the VSE/ESA host is used for obtaining data):

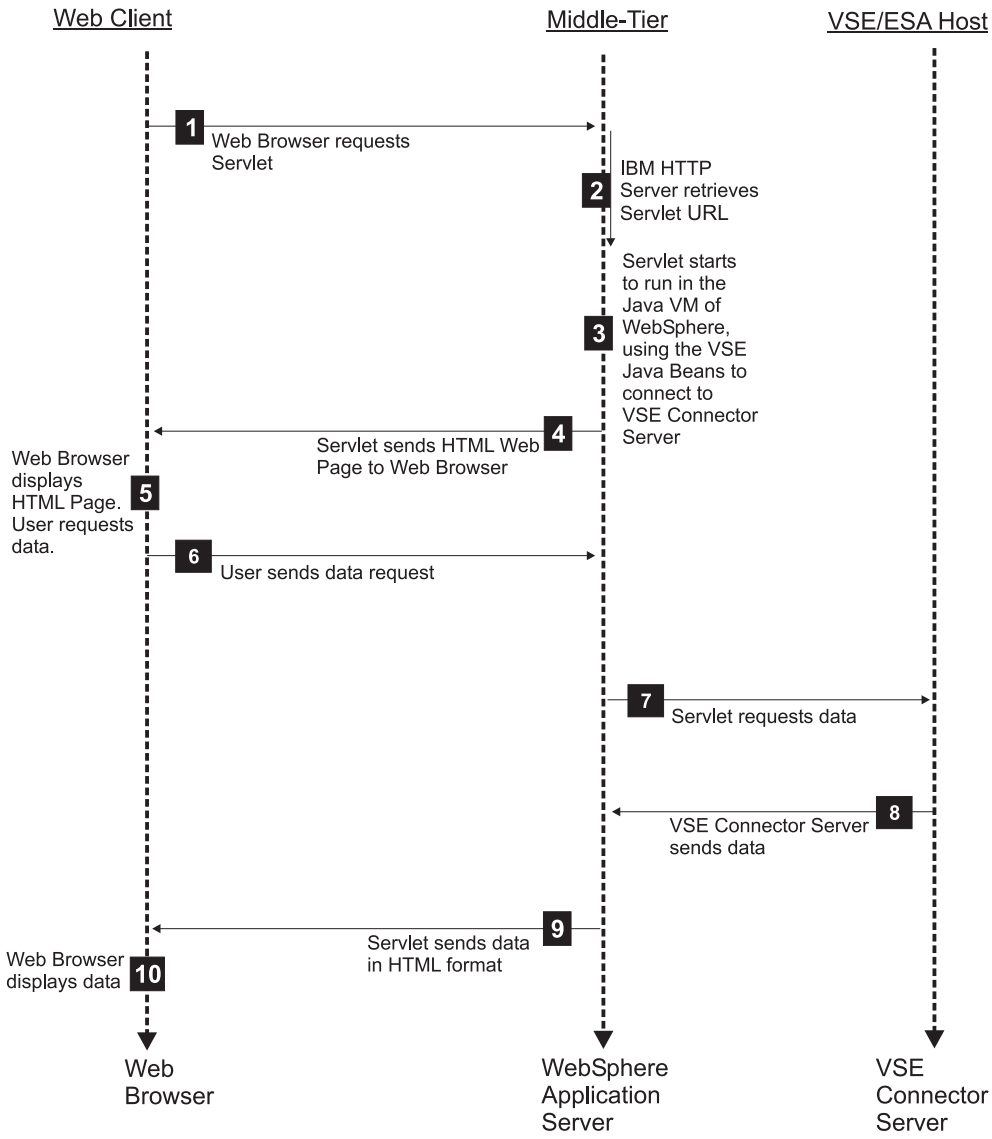


Figure 115. How Servlets Are Used in the VSE/ESA 3-Tier Environment

© Copyright IBM Corp. 2000, 2003

Using Java Servlets

HTTP Sessions are used between the Web Client and the middle-tier for sending and receiving data. Socket connections are used between the middle-tier and the VSE/ESA host for sending and receiving data.

The number of each list item below describes a step shown in Figure 115 on page 243:

1. The client's Web browser sends a servlet request to the IBM HTTP Server running on the middle-tier.
2. The IBM HTTP Server retrieves the servlet URL and passes this to the WebSphere Application Server.
3. The WebSphere Application Server runs the servlet in its Java Virtual Machine (JVM). The servlet also uses the VSE Java Beans class library (**VSEConnector.jar**) to build a connection to the VSE Connector Server running on the VSE/ESA host.
4. The servlet sends the dynamically-created HTML Web Page to the client's Web browser, via TCP/IP.
5. The client's Web browser displays the Web Page. The end-user now requests data that is stored on the VSE/ESA host.
6. The client's Web browser sends the request for data to the servlet, which is still loaded in the Java Virtual Machine of the WebSphere Application Server.
7. The servlet uses the VSE Java Beans to communicate with the VSE Connector Server running on the VSE/ESA host, requesting the VSE-based data.
8. The VSE Connector Server obtains the data and sends it to the servlet.

   Notes:

   1. The VSE Connector Server can be used for accessing VSE/VSAM, VSE/POWER, VSE/ICCF, or Librarian data.
   2. An alternate method for accessing VSAM data stored on the VSE/ESA host is to use a DB2 Stored Procedure on the middle-tier, which communicates with the VSAM file on the VSE/ESA host. This is described in "Using DB2 Stored Procedures to Access VSAM Data" on page 312.
9. The servlet generates a dynamic Web Page consisting of HTML code, and sends this together with the requested data back to the client's Web browser.
10. The client's Web browser re-displays the Web Page together with the requested data.

Compiling and Calling Servlets

To compile a servlet, you require servlet-related Java classes. These classes are normally provided together with your WebSphere Application Server installation. These classes are contained in a JAR file that must be included into your local classpath (for your compilations), as well as in the WebSphere Application Server's classpath.

Servlets can then be called in various ways. This section describes the most commonly-used way:

From the command line

You must simply enter:

```
http://server_host_name/servlet_engine_name/servlet_name
```

For example, from the command line of your Web browser you could enter:

```
http://MyComputer/webapp/VseServletEngine/MyServlet
http://9.164.123.456/webapp/VseServletEngine/MyServlet
```

How the WebSphere Application Server Stores Session Information

These TCP/IP connections are short-lived:

- From the Web client to the middle-tier platform.
- From the middle-tier to the VSE/ESA host.

The servlet's *HTTPSession* object allows Web clients to be identified over a series of HTTP requests. WebSphere does this internally, for example by storing a ″cookie″ on the Web client.

Each time the servlet is invoked, it checks whether the Web client already had a connection in the past, by searching for the *HTTPSession* object (which might be bound to the "cookie"). If the session does already exist, the servlet then checks if the session contains information about a previously-used session.

The *httpSession* instance is created when the servlet is invoked the first time (in this case, the session does not already exist). Therefore two actions must be performed:

1. Create the httpSession instance and store it in the HTTPServletRequest.
2. Define certain session properties, and store them in the httpSession.

Figure 116 on page 246 shows the ″logical view″ of how session information is stored and processed. *httpSession* contains the properties, and is stored on the Web client as a cookie.

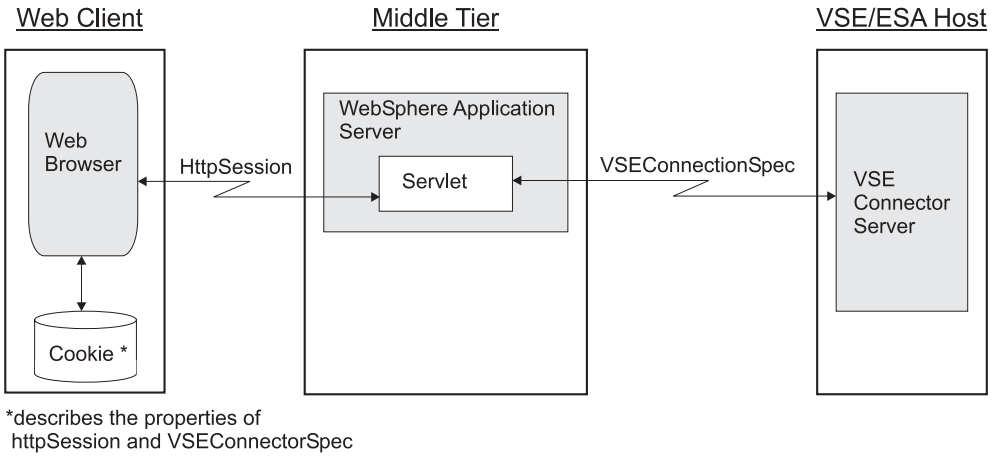


Figure 116. How Session Information Is Re-Used by the WebSphere Application Server

Example of How to Implement a Servlet

This section describes the *FlightOrderingServlet* sample, which is supplied as part of the online documentation. This servlet sample provides the capability to maintain flights and orders information in a simple VSAM-based flight booking system. It first defines, and then uses, two VSAM clusters which hold the data.

For details of how to define the data maps and views for the clusters used in this sample, refer to "Running the Sample Data-Mapping Applet" on page 207.

General Description of the Sample Servlet

The servlet is implemented in the Java source file **FlightOrderingServlet.java**. This source file implements the main servlet code, together with some inner classes, to handle flights, orders, create the VSAM clusters, and so on.

The servlet performs this general processing:

1. A Web page is displayed in which the user can choose between several actions. From this first Web page, the user can create the required VSAM clusters and fill them with data, order a flight, and cancel a flight.
2. When the user clicks **Order a flight**, a table with all available flights is displayed. Here, the user can select a flight number. When selecting a flight number, the servlet displays another Web page containing the properties of this flight, and the window controls required for the user to be able to place an order. The user can enter:
   - Name
   - Number of seats to order
   - Whether or not the reserved seats should be in the non-smoking area.
3. After clicking the push button to order the flight, the properties of the VSAM record that contains the flight details, are updated. In addition, a new record is created in the ORDERS cluster.

The following sections describe the most important parts of the sample servlet's Java code. The complete source code is contained in the VSE Connector Client online documentation.

You might also refer to the VSE Connector Client online documentation for these additional servlet samples:

- *SearchServlet*: a servlet that shows you how to search VSE library systems (POWER, ICCF, VSE Libraries, VSAM) for files, including those containing specific text.
- *SdlServlet*: a servlet that shows you how to display a list of VSE phases loaded into the SVA.

Creating the VSAM Clusters for the Sample

These are the two VSAM clusters you require, in order to run the sample. Both are predefined in the VSAM catalog VSAM.VSESP.USER.CATALOG (VSESPUC) of VSE/ESA 2.5 onwards:

```
FLIGHT.ORDERING.FLIGHTS (FLIGHTS) - KSDS
Offset Length Type      Key    Field Name     Description
-------------------------------------------------------------
0      4      UNSIGNED  yes    FLIGHT_NUMBER  Flight Number
4      20     STRING    no     START          Start
24     20     STRING    no     DESTINATION    Destination
44     5      STRING    no     DEPARTURE      Departure (hh:mm)
49     5      STRING    no     ARRIVAL        Arrival   (hh:mm)
54     4      UNSIGNED  no     SEATS          Seats
58     4      UNSIGNED  no     RESERVED       Seats reserved
62     4      PACKED    no     PRICE          Price
66     20     STRING    no     AIRLINE        Airline
```

Figure 117. VSAM Structure of FLIGHT.ORDERING.FLIGHTS

```
Offset Length Type      Key    Field Name     Description
-------------------------------------------------------------
0      20     STRING    no     FIRST_NAME     First Name
20     20     STRING    no     LAST_NAME      Last Name
40     4      UNSIGNED  no     FLIGHT_NUMBER  Flight Number
44     4      UNSIGNED  no     SEATS          Seats
48     1      BINARY    no     NON_SMOKE      Non Smoke 0=no
```

Figure 118. VSAM Structure of FLIGHT.ORDERING.ORDERS

HTML Constructs Used With the Sample

As servlets are used for generating dynamic *Web pages*, HTML syntax is therefore an essential part of every servlet. Here are two HTML constructs that are typically used in servlets:

How a Servlet Can Create Tables in HTML

A table is defined in HTML in this general way:

```
<table>
<tr><td>"First table cell"</td><td>"Second cell"</td></tr>  (First row)
<tr>  .... </tr>                                           (Second row)
...
</table>
```

A servlet can create *dynamic tables* by writing values of program variables into a table. A dynamic table might look like this:

```
out.println("<table>");
out.println("<tr><td>" + string1 + </td></tr>");
out.println("<tr><td>" + string2 + </td></tr>");
...
out.println("</table>");
```

where:
- *string1* and *string2* are string variables.
- *out* is the PrintWriter instance that was obtained from *HttpServletResponse* in the servlet's *doGet()* or *doPost()* methods.

Using Forms to Obtain a User's Input

After showing a Web page, the servlet must be able to obtain the user's input in order to process the next servlet request. This is usually done using *forms*, which can display window controls such as text fields or push buttons. When the user performs the action associated with the form (usually a push button), the servlet is called again with this action and input parameters taken from text fields, check boxes and other input controls.

```
out.println("<form action=\"/servlet/FlightOrderingServlet\" method=get>"); 1
out.println("<input type=hidden name=\"action\" value=\"order3\">");
out.println("<input type=hidden name=\"flight\" value=\"" +
                                                    flightNumber + "\">");
out.println("<table>");
out.println("<tr><td><b>First Name:</b></td>");
out.println("<td><input size=20 maxlength=20 name=\"firstname\"></td>");
out.println("</tr>");

out.println("<tr><td><b>Last Name:</b></td>");
out.println("<td><input size=20 maxlength=20 name=\"lastname\"></td>");
out.println("</tr>");
...

out.println("<tr><td><b>Non smoking:</b></td>");
out.println("<td><input type=checkbox name=\"nonsmoke\"
                                          value=\"yes\">Yes</td>");
out.println("</tr>");
out.println("</table><p>");
out.println("<input type=submit value=\"Order It!\">");
out.println("</form>");
```

Note: The string used here to perform a servlet call is dependent on the:
- Version of the WebSphere Application Server you have installed.
- Name you have defined for your servlet engine.

Figure 119. Example of a Servlet Using Forms to Obtain a User's Input

By specifying *method=get* (at 1), the *doGet()* method of the servlet will be called. However:

- You could also specify *method=post*, which would cause the *doPost()* method to be called.
- The difference between *method=get* and *method=post* is that:
  - Using *doGet()* will display the generated servlet invocation string in the Web browser's address/location field.
  - Using *doPost()* will suppress the Web browser's address/location field.
- You are recommended to use *doPost()* if the form contains any password fields. Passwords will then appear as clear text, when using *doGet()*.

The HTML code shown in Figure 119 on page 248 will display the window controls shown below:

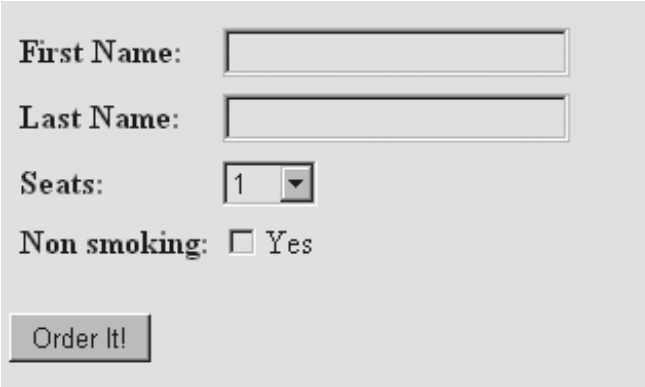


Figure 120. Example of Using Forms to Display Window Controls

If the user presses the push-button **Order It!**, this servlet invocation string will be generated:

```
http://computername/servlet/FlightOrderingServlet?action=order3&flight=34
&firstname=name1&lastname=name2&seats=1&nonsmoke=no
```

where:
- *name1* is the string that was entered in the *firstname* field.
- *name2* is the string that was entered in the *lastname* field.
- *computername* is the name of your workstation in your network, or its IP address.

You could also enter this string manually in your Web browser's address/location field.

Sample Servlet Step 1: Display a List of Flights

In this step, a list of available flights is displayed, from which the user can place an order. The *selectRecords()* method receives all records in the FLIGHT.ORDERING.FLIGHTS cluster, and displays them.

```
public class FlightOrderingServlet extends HttpServlet
{
   // Names of the Clusters and Maps.
   String   vsamCatalog    = "VSESP.USER.CATALOG";
   String   flightsCluster = "FLIGHT.ORDERING.FLIGHTS";
   String   ordersCluster  = "FLIGHT.ORDERING.ORDERS";
   String   flightsMapName = "FLIGHTS_MAP";
   String   ordersMapName  = "ORDERS_MAP";
   ...

   public void doOrderStep1(PrintWriter out,VSESystem system)
   {
      VSEVsamCluster flights = null;
      VSEVsamMap  flightsMap = null;
      FlightsListener fl;

      // create the instances of the flights cluster with its map
      flights = new VSEVsamCluster(system,vsamCatalog, flightsCluster);  1
      flightsMap = flights.getVSEVsamMap(flightsMapName);
      ...

      try
      {
         // use the Listener to build the table
         fl = new FlightsListener(out);  2
         flights.addVSEResourceListener(fl);

         // select all records of the flights cluster (no filter)
         flights.selectRecords(flightsMap);  3
         flights.removeVSEResourceListener(fl);
      }
      catch(Throwable t)
      {
         ...
      }
      ...
   }
   ...
```

Figure 121. Sample Servlet Code for Displaying a List of Flights

The numbers below refer to the numbers in Figure 121:

1. A local instance *flights* of a *VSEVsamCluster* is created. Next, a new map instance *flightsMap* with the specified name is created. This map will be filled with the actual data fields, during Step 2 below. During Step 1, however, these objects are only local, and no host access has been made.
2. A *VSEResourceListener* is created, to receive the VSAM records from the host.
3. All records are retrieved from the host. When the *selectRecords()* method returns, all VSAM records are displayed in the current HTML page.

Sample Servlet Step 2: Get Flight Instances from the Host

The following code is an extract of the inner class *FlightsListener*. The *listAdded()* method is a callback function that is called for each received VSAM record instance. For further details about callback functions, see "Using the Callback Mechanism of VSE Java Beans" on page 159.

```
public void listAdded(VSEResourceEvent event)
{
   String flightNumber, start, destination, departure;
   String arrival, price, airline;

   // The event data has to be a VSEVSamRecord
   if (!(event.getData() instanceof VSEVsamRecord))
     return;

   // Get the record -> it is a record of the flights cluster
   VSEVsamRecord flight = (VSEVsamRecord)(event.getData());   1
   try
   {
      // Get the fields of the record ...
      flightNumber = ((Integer)flight.getKeyField(0)).toString();
      start        = fli ght.getField(1).toString().trim();
      destination  = flight.getField(2).toString().trim();
      departure    = flight.getField(3).toString().trim();
      arrival      = flight.getField(4).toString().trim();
      price        = ((Integer)flight.getField(7)).toString().trim();
      airline      = flight.getField(8).toString().trim();

      // Write out an HTML line in the table
      ...  2

   }
   catch(Throwable t)
   {
         ...
  }
```

Figure 122. Sample Servlet Code for Getting Flight Instances from the Host

The numbers below refer to the numbers in Figure 122:

1. An explicit cast is necessary to get the *VSEVsamRecord* instance from the *VSEResourceEvent* data.
2. Refer to the VSE Connector Client online documentation for further code details.

The Java code described in Figure 122 displays this Web page:

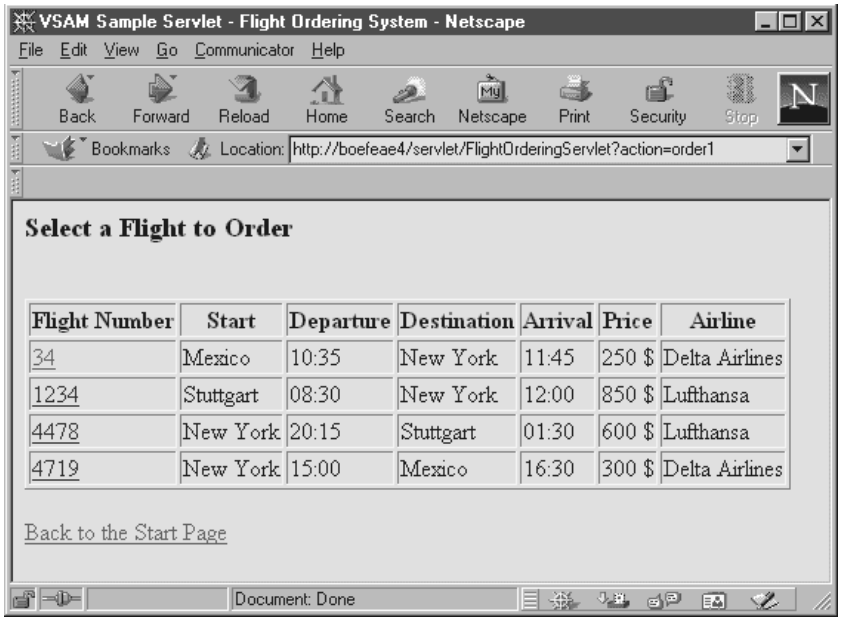


Figure 123. Flight Order Selection Window, As Generated by the Sample Servlet

In Figure 123, the first servlet call has finished. This means, the servlet code is still loaded into memory on the middle-tier platform by the WebSphere Application Server, but no further processing will be done until the servlet is called again.

Notes:

1. The next servlet call is in fact a new program call, and there is no way to store information in global variables, from one call to the next call. Therefore, all input parameters must be passed as *servlet parameters*.
2. In fact, there is one property stored over the lifetimes of a servlet: the VSE host connection specification (*VSEConnectionSpec*). This is stored in the client connection (*HttpSession*), which is always the same for *all* servlet calls. For further details, refer to the online documentation provided with the VSE Connector Client (see "Using the Online Documentation Options" on page 28).

Sample Servlet Step 3: Display the Properties of a Flight

In the method described here, the properties of a flight are displayed, together with the controls required to place an order. An HTML form is used to:

- Get the user's input.
- Initiate a new servlet call.

```
public void doOrderStep2(PrintWriter out,VSESystem system,
                         Hashtable parameters)
{
   VSEVsamCluster flights = null;
   VSEVsamMap     flightsMap = null;
   VSEVsamRecord  flight = null;
   int flightNumber, price, seats, reserved;
   String start, destination, departure, arrival, airline;

   // create the instances of the flights cluster and its map
   flights = new VSEVsamCluster(system, vsamCatalog, flightsCluster);
   flightsMap = flights.getVSEVsamMap(flightsMapName);

   // get a instance of a record of the flights cluster
   flight = flights.getVSEVsamRecord(flightsMap);   1

   // get the parameters
   flightNumber = 0;
   try
   {
      flightNumber = Integer.parseInt(getParameterValue(parameters,   2
                                                        "FLIGHT",true));
   }
   catch (Throwable t) {}
   ...

   // get the record and display the properties
   try
   {
      // set the key to identify the locat record with the record
      // on the host
      flight.setKeyField(0,new Integer(flightNumber));   3

      // now get the other fields
      start       = flight.getField(1).toString().trim();
      destination = flight.getField(2).toString().trim();
      departure   = flight.getField(3).toString().trim();
      arrival     = flight.getField(4).toString().trim();
      seats       = ((Integer)flight.getField(5)).intValue();
      reserved    = ((Integer)flight.getField(6)).intValue();
      price       = ((Integer)flight.getField(7)).intValue();
      airline     = flight.getField(8).toString().trim();

      // display the fields
      ...

      // check if enough seats are available
      ...

      // write out a form where the user can enter its name and select
      // the number of seats to order
      out.println("<form action=\"/servlet/FlightOrderingServlet\"   4
                                                          method=get>");
      ...
```

Figure 124. Sample Servlet Code for Displaying Properties of a Flight (Part 1 of 2)

```
            out.println("<input type=submit value=\"Order It!\">");
            out.println("</form><p>");
        }
        catch(Throwable t)
        {
            ...
        }
        ...
    }
```

Figure 124. Sample Servlet Code for Displaying Properties of a Flight (Part 2 of 2)

The numbers below refer to the numbers in Figure 124 on page 253:

1 A local instance of class *VSEVsamRecord* is created. No properties have been assigned at this point. This code simply creates an object. You could also use a constructor of the class to obtain the same result.

2 Function *getParameterValue()* is used to obtain the flight number from the form. The source code of this method is shown in the VSE Connector Client online documentation.

3 This statement now identifies one specific record in the FLIGHTS cluster. The flight number is provided in the user's input. Now, all other properties of this flight can be retrieved from this object.

4 Window controls required for the user to place the order, are displayed.

The coding described in Figure 124 on page 253 will display this Web page:

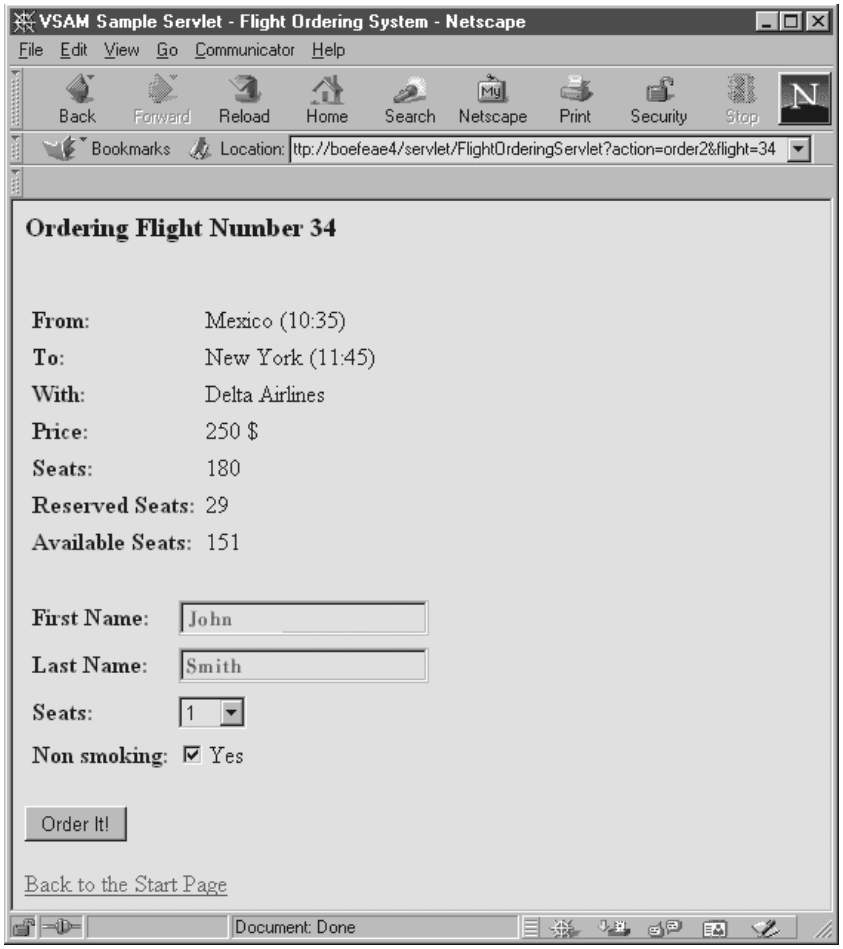


Figure 125. Flight Order Entry Window, As Generated by the Sample Servlet

Sample Servlet Step 4: Place an Order

The method described in this section is used to place an order. It is called when the user presses the **Order It** button:

1. The user's input is read from the form.
2. The flight's properties are updated (the number of reserved seats for this flight is increased by the number of ordered seats).
3. A new record is added to the ORDERS cluster.

```
public void doOrderStep3(PrintWriter out, VSESystem system,
                         Hashtable parameters)
{
   VSEVsamCluster flights = null, orders = null;
   VSEVsamMap flightsMap = null, ordersMap = null;
   VSEVsamRecord flight = null, order = null;
   int flightNumber, seatsToOrder, seats, reserved, price;
   String firstName, lastName;
   boolean nonSmoke, ok;

   // get the parameters from the form
   try
   {
      flightNumber = Integer.parseInt(   1
                         getParameterValue(parameters,"FLIGHT",true));
      seatsToOrder = Integer.parseInt(
                         getParameterValue(parameters,"SEATS",true));
      firstName = getParameterValue(parameters,"FIRSTNAME",false);
      lastName = getParameterValue(parameters,"LASTNAME",false);

      // Check user input ...
      ...

      nonSmoke = false;
      if (getParameterValue(parameters,"NONSMOKE",true) != null)
        nonSmoke = true;
   }
   catch(Throwable t)
   {
      // if not all parameters has been specified -> redisplay Step 2
      doOrderStep2(out,system,parameters);
      return;
   }

   // create instances of the flight cluster and its map
   flights = new VSEVsamCluster(system,vsamCatalog,flightsCluster);   2
   flightsMap = flights.getVSEVsamMap(flightsMapName);

   // get a instance of a record of the flights cluster
   flight = flights.getVSEVsamRecord(flightsMap);

   // create instances of the orders cluster and its map
   orders = new VSEVsamCluster(system,vsamCatalog,ordersCluster);   3
   ordersMap = orders.getVSEVsamMap(ordersMapName);

   // get a instance of a record of the orders cluster
   order = orders.getVSEVsamRecord(ordersMap);

   // Write HTML header
   ...
```

Figure 126. Sample Servlet Code for Placing an Order (Part 1 of 2)

```
    try
    {
       // get the flight record
       // set the key to identify the local record with
       // the record on the host
       flight.setKeyField(0,new Integer(flightNumber));   4

       // get some fields of interest
       seats        = ((Integer)flight.getField(5)).intValue();
       reserved     = ((Integer)flight.getField(6)).intValue();
       price        = ((Integer)flight.getField(7)).intValue();

       // check if enough seats are available
       ...

       // display the order properties
       ...
       // use the OrderCounter to get teh highest record number.
       // This is necessarry because RRDS can only add a record
       // with a non existing record number
       OrderCounter oc = new OrderCounter();   5

       // select all records to find the highest record number
       orders.addVSEResourceListener(oc);
       orders.selectRecords(ordersMap);
       orders.removeVSEResourceListener(oc);

       // now create the new order
       ok = createOrder(out,order,oc.getHighestRecNo()+1,   6
                 firstName,lastName,flightNumber,seatsToOrder,nonSmoke);

       out.println("</table><p>");

       // check if creating was ok ...
       if (ok)
       {
          // now update the flight record's fields
          // increase the reserved seats by the number of seats to order
          reserved += seatsToOrder;

          // set the field in the local record
          flight.setField(6,new Integer(reserved));

          // and commit the changes to make them permanent
          flight.commit();   7
       }
    }
    catch (Throwable t)
    {
       ...
    }

    // write out some status information and HTML footer
    ...

 }
```

Figure 126. Sample Servlet Code for Placing an Order (Part 2 of 2)

The numbers below refer to the numbers in Figure 126 on page 256:

1 Obtains the servlet parameters from the *form*.

2 Creates a local instance of the FLIGHTS cluster and one record belonging to this cluster. Up to this point, no data has been associated with these objects.

3 Creates a local instance of the ORDERS cluster and one record belonging to this cluster. Up to this point, no data has been associated with these objects.

4 Sets the key field of the FLIGHTS record, which is required before any further processing of the record can take place. The information is required internally in order to access the record on the host.

5 For RRDS clusters, new records are added using unique relative record numbers. Therefore, the highest relative record number must be determined, before a new record is added. The *OrderCounter* class is used to receive all records, count them, and return the highest relative record number.

6 Method *createOrder()* is now used to add a new record to the ORDERS cluster (in the code that follows).

7 The *commit()* method is used to make the updates permanent.

Sample Servlet Step 5: Create a New Flight

The method described here adds a new record to the FLIGHTS cluster. If the record already exists (as identified by its key), an appropriate error message is generated.

```
public boolean createFlight(PrintWriter out,VSEVsamRecord flight,
                        int flightNumber,String start,String
                        destination,String departure, String arrival,
                        int seats,int reserved,int price,String airline)
throws IOException,ConnectorEx ception
{
   try
   {
       // Set the key of the record
       flight.setKeyField(0,new Integer(flightNumber));   1

       // Set all other fields
       flight.setField(1,makeString(start,20));
       flight.setField(2,makeString(destination,20));
       flight.setField(3,makeString(departure,5));
       flight.setField(4,makeString(arrival,5));
       flight.setField(5,new Integer(seats));
       flight.setField(6,new Integer(reserved));
       flight.setField(7,new Integer(price));
       flight.setField(8,makeString(airline,20));
       // try to add this record
       flight.add();   2

       // Add new row to table
       out.println("<tr><td>" + flightNumber + "</td>");
       out.println("<td>" + start + "</td>");
       out.println("<td>" + destination + "</td>");
       out.println("<td>" + departure + "</td>");
       ...
       out.println("</tr>");
   }
   catch (AlreadyExistentException e)
   {
      // already existing
      return(false);
   }
   return(true);
}
```

Figure 127. Sample Servlet Code for Creating a New Flight

The numbers below refer to the numbers in Figure 127:

1. For *KSDS clusters* all key fields must be set for a given record, before performing any actions against the record.
2. Adds the new flight record to the cluster.

Sample Servlet Step 6: Create a New Order

The method described here adds a new record to the ORDERS cluster.

```
public boolean createOrder(PrintWriter out,VSEVsamRecord order,int recNo,
                        String firstName,String lastName,int flightNumber,
                        int seats,boolean nonSmoke)
throws IOException,ConnectorException
{
   byte[] b = new byte[1];
   try
   {
      // Set the relative record number
      order.setRelRecNo(recNo);   1

      // set all other fields
      order.setField(0,makeString(firstName,20));
      order.setField(1,makeString(lastName,20));
      order.setField(2,new Integer(flightNumber));
      order.setField(3,new Integer(seats));
      if(nonSmoke)
        b[0] = (byte)0x01;
      else
        b[0] = (byte)0x00;
      order.setField(4,b);

      // Add the new record
      order.add();   2

      // Write table row
      out.println("<tr><td>"+firstName+"</td>");
      out.println("<td>"+lastName+ ... +"</td></tr>");
   }
   catch (AlreadyExistentException e)
   {
      // Already existing ...
      return(false);
   }
   return(true);
}
```

Figure 128. Sample Servlet Code for Creating a New Order

The numbers below refer to the numbers in Figure 128:

1. For *RRDS clusters,* the relative record number must be set before a new record can be added.
2. Adds the new order record to the cluster.

The coding described in Figure 128 will display this Web page:

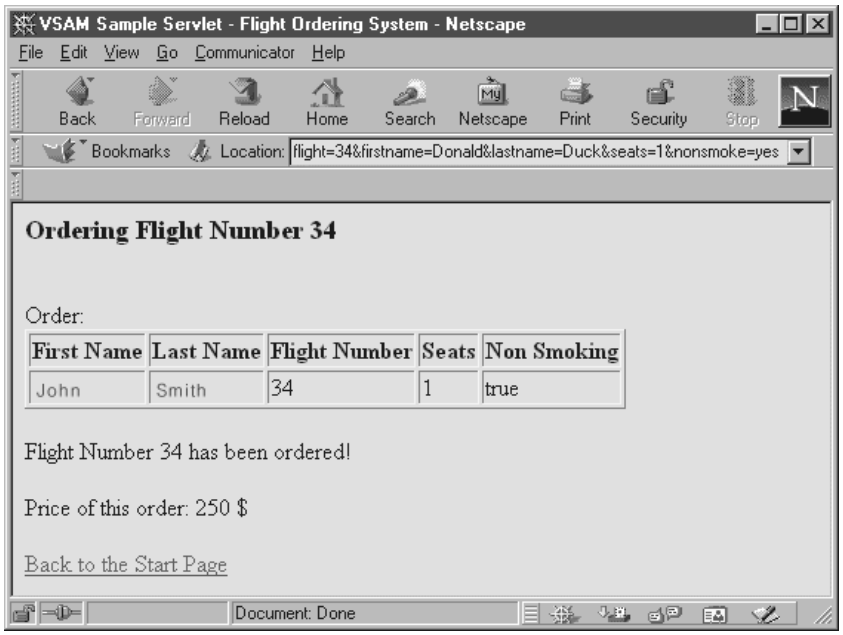


Figure 129. Flight Order Confirmation Window, Generated by the Sample Servlet

Chapter 21. Using Java Server Pages to Access Data

JSP Requests Java Server Pages (JSPs) are a way for developers who are familiar with HTML to easily create *servlets*, since JSPs are compiled into servlets. They are also useful for Java applications in which servlets, and other generators of dynamic HTML content, must be integrated with static HTML. The commonly-used naming conventions for JSPs is that the suffix ends with **.jsp**.

This chapter contains these main sections:

- "How JSPs Are Used in 3-Tier Environments"
- "Example of a Simple Java Server Page" on page 265

How JSPs Are Used in 3-Tier Environments

Figure 130 on page 264 describes how JSPs are used within the VSE/ESA *3-tier environment*. The number of each list item below describes a step shown in Figure 130 on page 264.

1. The client's Web browser sends the request for a JSP URL to the IBM HTTP Server on the middle-tier. Each JSP is stored in the IBM HTTP Server's normal document hierarchy.
2. The IBM HTTP Server retrieves the JSP and sends it to the WebSphere Application Server.
3. The action now performed by the WebSphere Application Server depends upon whether or not this JSP has previously been compiled to a servlet:
   - If the JSP has *not* previously been compiled to a servlet (or if the JSP has been modified since the last time it was used), the WebSphere Application Server uses its JSP-engine to do so.
   - If the JSP *has* previously been compiled to a servlet (and has not been modified since the last time it was used), the resulting servlet will already be stored in the servlet repository. A compilation of the JSP is therefore not required.

   The WebSphere Application Server then runs the servlet in its Java Virtual Machine (JVM). The servlet uses the VSE Java Beans class library (**VSEConnector.jar**) to build a connection to the VSE Connector Server.
4. The servlet sends the required HTML Web Page to the client's Web browser, via TCP/IP.
5. The client's Web browser displays the Web Page. The Web browser now sends a request for data to the servlet running in the WebSphere Application Server's Java Virtual Machine.
6. The servlet sends the request for data to the VSE Connector Server running on the VSE/ESA host (using the connection previously built using VSE Java Beans).

© Copyright IBM Corp. 2000, 2003

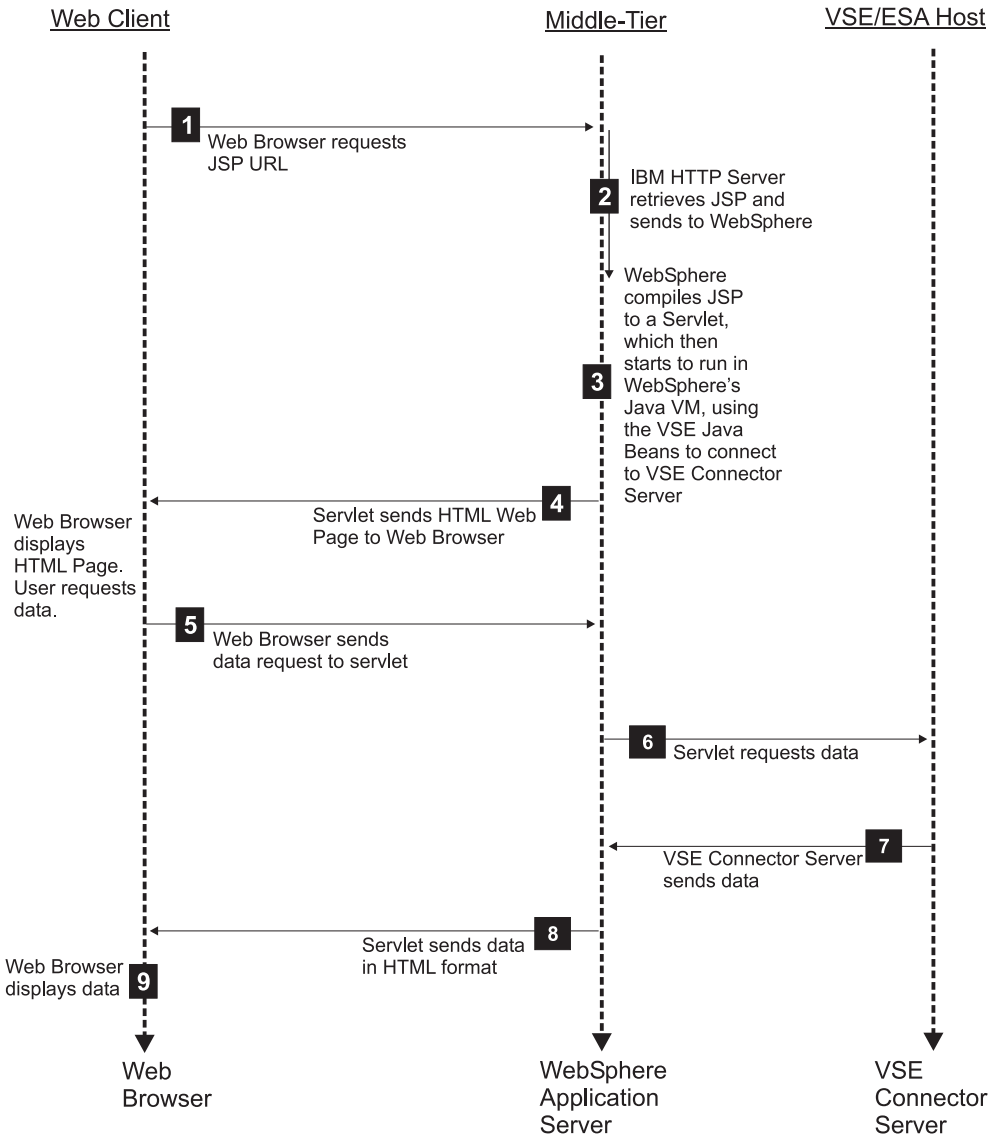


Figure 130. How JSPs Are Used in the VSE/ESA 3-Tier Environment

7 The VSE Connector Server obtains the data and sends it to the servlet.

Notes:

1. The VSE Connector Server could also be used for accessing POWER, VSE/ICCF, or Librarian, or sending console commands.
2. An alternate to using the *VSE Connector Server* is for accessing data is to use a DB2 Stored Procedure on the middle-tier, which communicates directly with the VSAM file system on the VSE/ESA host.
3. If the end-user had requested DB2 data, the data could be retrieved using a DB2 Connect router on the middle-tier and a DB2 Server for VSE on the VSE/ESA host. Alternatively, the DB2 data could be retrieved using a DB2 Connect router on the middle-tier and a DB2 Stored Procedure on the VSE/ESA host.
4. If the end-user had requested DL/I data, the data could be retrieved using a user-written application on the middle-tier which communicates directly with a DB2 Stored Procedure on VSE/ESA host, which can in turn accesses the DL/I databases of the VSE/ESA host.

5. If the end-user had requested CICS data, the data could be retrieved using an MQSeries Server router on the middle-tier and an MQSeries Server for VSE/ESA on the VSE/ESA host.

8 The servlet generates a dynamic Web Page consisting of HTML code, and sends this together with the requested data back to the client's Web browser.

9 The client's Web browser re-displays the Web Page together with the requested data.

HTTP Sessions are used between the Web Client and the middle-tier for sending and receiving data. Connect Sessions are used between the middle-tier and the VSE/ESA host for sending and receiving data.

The advantage in using JSPs rather than servlets, is that by using JSPs your HTML has a wider functionality. You can create HTML from servlets, but you require a lot of programming effort. Using JSPs, however, you can carry out a change and then simply let the WebSphere Application Server recompile and execute a servlet which contains your changes.

JSPs also have the advantage that they can be used by authors who do not possess much HTML knowledge. A JSP normally accesses an Enterprise Java Bean (EJB) that encapsulates database queries and business logic. The data returned by the EJB is sent to the Web client in HTML format and can then be dynamically included into the current Web page.

Example of a Simple Java Server Page

Figure 131 shows how a JSP can be used to display the current date. These special tags (<% and %>) are used to enclose the Java code. The JSP engine dynamically compiles the JSP into a servlet. The servlet is then executed, and displays the Web page.

```
<html>
<head>
<title>My first JSP</title>
</head>
<% import java.util.*; %>
<% Date date = new Date();
   response.println("The current date is " + date);
   ...
%>
</body>
</html>
```

Figure 131. Example of a Java Server Page /JSP)

Chapter 22. Using EJBs to Represent Data

EJBs are "write once, run anywhere" distributed beans that run in a web application server environment, such as IBM's WebSphere Application Server. They are intended to represent either:

- data in a database (entity beans).
- the connection to a remote data store (session beans).

In a VSE/ESA 3-tier environment, you can use EJBs to represent:

- data in a relational database (DB2).
- non-relational data, such as DL/I or VSAM. This requires that you have previously mapped such non-relational data to a *relational* structure (for details, refer to Chapter 15, "Mapping VSE/VSAM Data to a Relational Structure", on page 129).

EJBs are easy to implement. Furthermore, they:

- Are the most powerful feature of the WebSphere Application Server environment.
- Allow the application programmer to concentrate on developing the Java applications, without having to code the access to data.

This chapter contains these main sections:

- "Overview of the EJB Architecture"
- "Example of Using EJBs to Access VSAM Data" on page 273
- "Example of Implementing VSAM-Based EJBs" on page 274

Overview of the EJB Architecture

This information provides you with an overview of the EJB architecture. A practical example of how EJBs can be implemented, is provided in "Example of Implementing VSAM-Based EJBs" on page 274.

The WebSphere Application Server server must provide a number of services to EJBs in order to manage them properly. These services are provided by an entity called an *EJB Container* (described in "Overview of How EJB Containers are Used" on page 268).

There are two types of EJBs, *Entity beans* and *Session beans.* The table below shows the properties of Entity beans and Session beans.

Table 7. Properties of Session Beans and Entity Beans

Bean Type	Description	State and Persistence
Session	Short lived beans tied to a single client session	Two major forms of Session beans: **Stateless** Perform a simple operation. Holds no data. **Stateful** Maintains variables between client requests.

© Copyright IBM Corp. 2000, 2003

Table 7. Properties of Session Beans and Entity Beans (continued)

Bean Type	Description	State and Persistence
Entity	Long lived beans existing across many client sessions	Usually wrapped by Session beans. The EJB example provides two session beans that access one entity bean. All beans hold data, but there are two ways to manage persistence (synchronization with database): **Bean managed** Bean must get and put rows into the database. **Container managed** Bean gets and puts data automatically. The entity bean belonging to the EJB example implements *bean-managed* persistence.

Session beans perform the business logic in an application. These beans could be used to represent a shopping "cart" in an online ordering system, or to calculate sales tax on a purchase. A Session bean is a normal Java class tied to a single client session. They have several other restrictions. They cannot:

- Start new threads (since EJBs run inside the EJB Container and not within the Java Virtual Machine).
- Carry out any functions except representing one row of a database (VSAM record or DB2 row).
- Contain static read/write variables.
- Use **java.io** classes.

Entity beans have the following characteristic. They:

- must access data through JDBC or by some other means (since Entity beans directly represent data). The EJB example (included on page 274) uses the VSE Java Beans class library to access VSAM data on a remote VSE/ESA host.
- cannot be used without low-level support such as a JDBC interface or a connector to other data sources.
- can be used by several clients at the same time.
- have a lifetime equal to the length of time that the underlying data they represent, exists.

Overview of How EJB Containers are Used

EJB Containers act as an intermediaries between EJBs and clients, and also manage multiple EJB instances. After an EJB is written, it must be stored in a container which resides on the WebSphere Application Server (or another application server).

Figure 132 on page 269 shows how containers are used to manage EJBs.

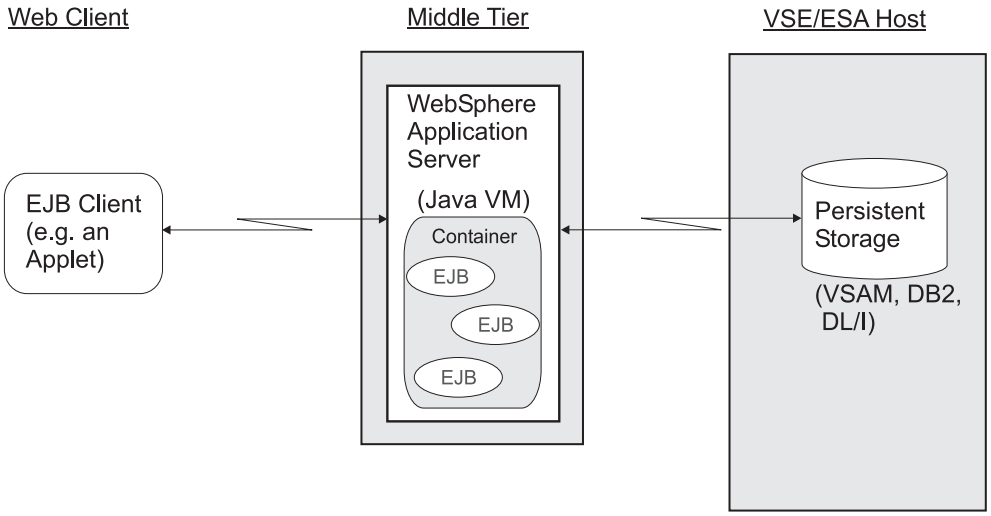


Figure 132. How Containers Are Used To Manage EJBs

Several tasks must be performed by the EJB container in order to fulfill its role as an EJB/client intermediary. These tasks include:

- Instance Passivation/Activation - Temporarily swapping EJBs in and out of storage.
- Instance Pooling - Sharing instances of EJBs among multiple clients.
- Database Connection Pooling - Allowing an EJB to use an open connection to the database from a pool of existing connections.
- Pre-cached Instances - Caching EJB state information to expedite the initial creation of EJBs.

Therefore, the container simplifies the EJB implementation process by managing all threading and client interaction with the EJBs. It also reduces the load on servers and databases by coordinating connection and instance pooling. As EJBs become standardized, vendors will increasingly provide EJB containers separately from application servers.

How EJBs Compare to JavaBeans / Java Servlets

The distinction between EJBs and JavaBeans lies primarily in their roles. JavaBeans are intended to be used as visual components in Graphical User Interfaces (GUIs), such as buttons and labels. On the other hand, EJBs are designed to either represent data in a database or perform actions on this data, and therefore have a restricted use.

Both EJBs and servlets reside in a multi-tier architecture. Servlets take in requests over HTTP, process the requests, and send responses over HTTP in the form of HTML code that can be viewed in a Web browser. Thus, servlets essentially provide User Interface functionality and business logic. When using EJBs, however, it is the *client's responsibility* to provide the User Interface.

Here is a summary of the limitations of using EJBs:

- They cannot access the local file system of the web server platform.
- There is no way to implement callback mechanisms, because EJBs are always single-thread programs. This is a disadvantage especially for long running actions, like all kind of requests that retrieve lists of objects.

- There is no way to stop a running action, nor to begin processing the result list before the complete action has finished.

Implementing Your Client Applications

Enterprise Java Beans allow you to separate the development of your client User-Interface from that of your business logic. Therefore, you can use various client platforms and multiple User Interfaces. EJBs can interact with many diverse client types including: other EJBs, Java applications, Java applets, Java servlets, and non-Java components by using CORBA connectivity.

Enterprise Java Beans provide easy connectivity to all Java and CORBA compatible components.

1. Using JNDI or another naming system, the client program locates the EJB home and uses the home to create a remote interface to access the bean.
2. This triggers the EJB container to create the bean.
3. After creating the remote interface, the bean appears to the client as any other class except that all passed objects must be serialized and sent over the network.

EJBs use RMI and 'stubs' to transmit information between the EJB container and the client computer. RMI requires that all objects passed across the network are serializable. Figure 133 shows the entities involved in a single EJB method call.

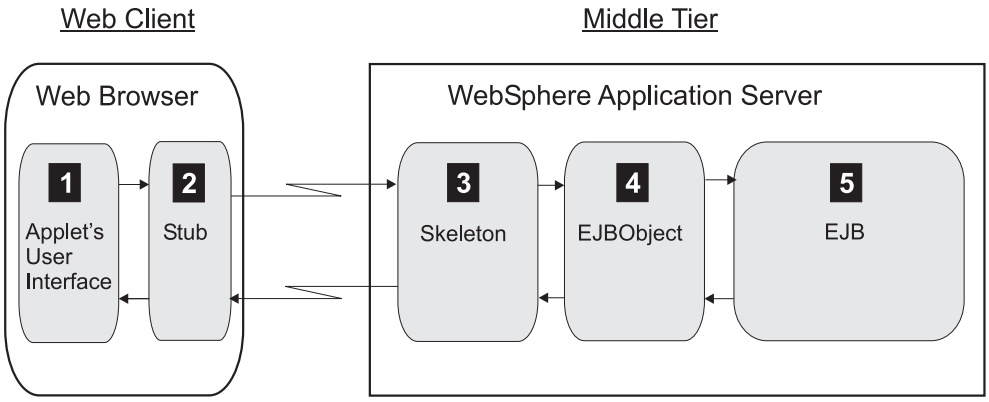


Figure 133. Overview of the Entities Involved in an EJB Method Call

The general processing during a method call is as follows:

1. The User Interface invokes the container generated stub, which serializes the arguments of the method call.
2. The generated stub sends the data over the network.
3. The skeleton stored on the EJB Server de-serializes the arguments, and passes them to the container-generated EJBObject (which is an implementation of the remote interface).
4. The EJBObject then interacts with the EJB.
5. When the EJB has completed its processing, it passes the result back to the EJBObject.

The reply is passed back to the User Interface, using the reverse path as described above. Therefore:

- The *server* manages calls to the EJBObject.

- The *container* manages calls to the EJB using the EJBObject.

EJBs connect to each other using the same method of calling, so it is very easy to implement a system of EJBs that communicate together. Even if the EJBs are in the same container, every method call must be made through the container to avoid multi-threading problems.

CORBA, Common Object Request Broker Architecture, provides a method for non-Java applications to connect to EJBs. CORBA works much the same way as RMI, but requires an application server that supports CORBA to EJB mapping and requires the use of different naming services.

How an EJB Client Accesses EJBs

Figure 134 shows how an EJB client accesses an Enterprise Java Bean (EJB). This information applies to both the example EJB and to EJBs in general.

The client never communicates directly with the EJB. Instead, it "talks to" the EJB via its *home interface* and its *remote interface*. The home interface and remote interface code are both created by the WebSphere Application Server during deployment of the EJBs.

The EJB server and EJB container are part of your Web application server (such as the WebSphere Application Server), and might be supplied by independent software vendors.

Therefore, this code must be implemented for an EJB:

- The *EJB client*, that communicates with the EJB. This is usually a servlet or an applet, but could also be any other Java program.
- The *EJB home interface*. The EJB developer simply specifies the interface. The EJB home interface is then created by the WebSphere Application Server when it deploys the EJB.
- The *EJB remote interface*. The EJB remote interface is also created by the WebSphere Application Server when it deploys the EJB.
- The *EJB class*. This class implements the EJB's business logic.

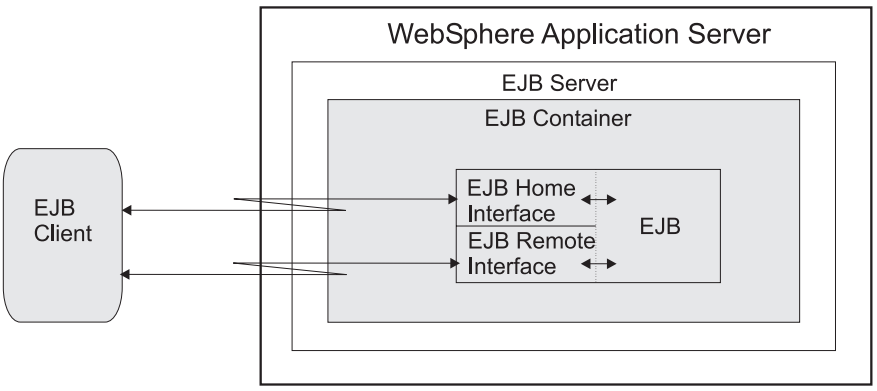


Figure 134. How an EJB Client Communicates with an EJB

In Figure 134, the *EJB client* can be:

Servlet or JSP
Runs in the same WebSphere Application Server as the EJB. Using servlets as EJB clients makes it possible to obtain access to the EJB data from any Web browser, via your company's Intranet or the Internet. The deployed **ejb-jar** file resides on the server that runs the WebSphere Application Server. For example, online shopping is often performed using a combination of static Web pages and servlets.

Java application
Runs on a different workstation and communicates with the EJB via a TCP/IP connection. You will normally use a Java application as an EJB client for test purposes in your development environment. A Java application is the most simple EJB client and can be used to implement a variety of test cases. Because the deployed **ejb-jar** file must be accessible by both the EJB client and the WebSphere Application Server, you must either copy the file to the workstation where your EJB client runs, or run the EJB client on the same machine as the WebSphere Application Server. You might also make the file accessible to both platforms over your company's network.

Applet
Runs in a Web browser on a different workstation and communicates with the EJB via a TCP/IP connection. If you require a more advanced user interface than is possible using HTML, you can use an applet as the EJB client. For example, online banking systems often use applets. In such systems, all classes that the applet requires, including the deployed **ejb-jar** file, are downloaded to the Web browser client when the applet runs. This can lead to performance problems, especially where the applet is accessed via the Internet.

Another EJB
Runs in the same or a different WebSphere Application Server as the EJB. For example, the employer bean and employee bean provided with the VSE Connector Client example, act as EJB clients when they access the record bean.

In other words, an EJB client is any Java program that accesses an EJB.

Example of Using EJBs to Access VSAM Data

Figure 135 provides an illustration of how EJBs are used together with an applet in the VSE/ESA 3-tier environment, to access VSAM data:

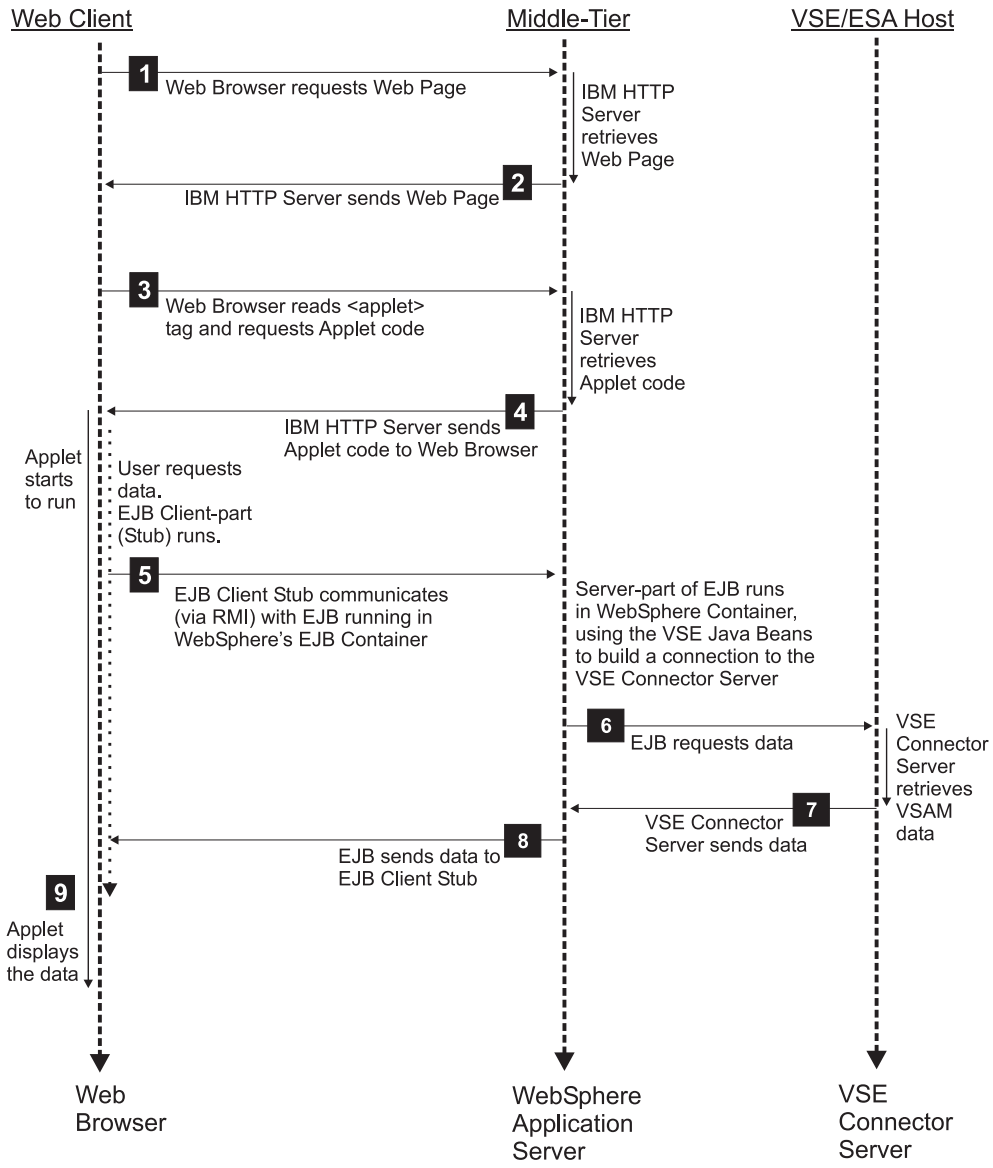


Figure 135. How EJBs Are Used Together with an Applet in the 3-Tier Environment

HTTP Sessions are used between the Web Client and the middle-tier for sending and receiving data. Socket connections are used between the middle-tier and the VSE/ESA host for sending and receiving data.

The number of each list item describes a step shown in Figure 135:

1. The client's Web browser sends a request for an HTML page to the IBM HTTP Server running on the middle-tier. This example assumes that the HTML file contains an applet tag which then causes the applet to be called. The applet acts as an EJB client to access the EJB data.
2. The IBM HTTP Server retrieves the Web Page, and sends it to the client's Web browser.

3. The client's Web browser reads an <applet> tag, and sends a request for the applet code to the IBM HTTP Server running on the middle-tier. The IBM HTTP Server sends a *JAR* file (containing the applet and the Java routines required to run it) to the client's Web browser.
4. The IBM HTTP Server retrieves the applet code, and sends it to the client's Web browser.
5. The client's Web browser runs the applet. The end-user now requests data that is stored on the VSE/ESA host, via the use of EJBs that are stored on the middle-tier. The applet uses EJB *stub classes* to communicate with the EJB server-part, via RMI. See Figure 132 on page 269 and Figure 133 on page 270 for further details. The server-part of the EJB uses the VSE Java Beans class library (**VSEConnector.jar**) to build a connection to the VSE Connector Server.
6. The server-part of the EJB requests the data from the VSE Connector Server running on the VSE/ESA host, using the connection previously built using the VSE Java Beans.
7. The VSE Connector Server obtains and then sends the requested VSAM data to the server-part of the EJB.
8. The server-part of the EJB sends the data to the EJB Stub running in the client's Web browser.
9. The applet running inside the client's Web browser displays the Web Page together with the requested data.

Example of Implementing VSAM-Based EJBs

EJBs are typically used so that one EJB instance encapsulates the data of one relational table row. The example provided here shows how to represent non-relational VSAM data, that has been mapped to a relational structure, by an EJB. The mapping of non-relational data to a relational structure is described in Chapter 15, "Mapping VSE/VSAM Data to a Relational Structure", on page 129.

This example consists of *three* EJBs:

- *RecordBean*, which is an entity bean, represents and contains a complete mapped VSAM record that belongs to a VSAM cluster containing data about employees.
- *EmployerBean* and *EmployeeBean*, which are session beans, provide the business logic to access the data represented by the *RecordBean*. They are used to implement two different views on this data, an employee's view and an employer's view.

In this example, utilities are also provided which are used to fill the VSAM cluster with sample data.

The execution of the sample EJB is described in these sections:

- "Step 1: Define the Sample's VSAM Cluster" on page 275
- "Step 2: Create the Record Layout for Employees" on page 275
- "Step 3: Specify the EJB's Home Interface" on page 276
- "Step 4: Specify the EJB's Remote Interface" on page 276
- "Step 5: Implement the RecordPK Class" on page 277
- "Step 6: Implement the EJB Code" on page 277
- "Step 7: Compile the Java Source Files" on page 282

- "Step 8: Deploy the EJBs" on page 282
- "Step 9: Access the EJBs from an EJB Client" on page 283

Step 1: Define the Sample's VSAM Cluster

In this first step, the VSAM cluster that is used by this example is created, and filled with sample data.

Note: Because the column placement of the label information in the IESVCLUP step is critical, you are recommended to use the VSE/ESA Interactive Interface dialogs to define this file!.

Use the following job to define the sample cluster EJB.VSAM.EXAMPLE:

```
* $$ JOB JNM=DEFINE,CLASS=0,DISP=D,NTFY=YES
// JOB  DEFINE EJB SAMPLE CLUSTER
// EXEC IDCAMS,SIZE=AUTO
 DEFINE CLUSTER ( -
        NAME (EJB.VSAM.EXAMPLE                          ) -
        CYLINDERS(2        2          ) -
        SHAREOPTIONS (2) -
        RECORDSIZE (80     80    ) -
        VOLUMES (DOSRES ) -
        NOREUSE -
        INDEXED -
        FREESPACE (15 7) -
        KEYS (4   0     ) -
        NOCOMPRESSED -
        TO (99366  )) -
        DATA (NAME (EJB.VSAM.EXAMPLE.@D@                       ) -
        CONTROLINTERVALSIZE (4096  )) -
        INDEX (NAME (EJB.VSAM.EXAMPLE.@I@                     )) -
        CATALOG (VSESP.USER.CATALOG                        )
  IF LASTCC NE 0 THEN CANCEL JOB
/*
// OPTION STDLABEL=ADD
// DLBL EJBSAMP,'EJB.VSAM.EXAMPLE',,VSAM,                            X
               CAT=VSESPUC
/*
// EXEC IESVCLUP,SIZE=AUTO
A EJB.VSAM.EXAMPLE                             EJBSAMP VSESPUC
/*
/&
* $$ EOJ
```

Step 2: Create the Record Layout for Employees

Create the record layout for employees, using this layout:

```
EMPNUM     Unsigned, offset=0,  length=4   (employee number = key)
PASSWORD   String,   offset=4,  length=10
NAME       String,   offset=14, length=25
DEPT       Unsigned, offset=39, length=4   ("department number")
HOURLY     Unsigned, offset=43, length=4   ("hourly wage")
PTD        Unsigned, offset=47, length=4   ("paid to day")
TNP        Unsigned, offset=51, length=4   ("time not paid")
```

- *To access the record of a specific employee*, you must specify the employee's number and a password.
- *To add new data to the cluster*, use the utility program *EJBPrepareData.java*. This program is supplied with the VSE Connector Client.
- *To create the map and its data fields*, use utility program *EJBPrepareData.java*.
- *To display the sample data*, use the Java program *EJBShowData.java*, which is supplied with the VSE Connector Client. You can also view the data using the

VSE Navigator application, which can be downloaded from the VSE Homepage. For details of how to access the VSE Homepage, see "Where to Find More Information" on page xix.

- *To run the pre-compiled utilities*, go to the **\vsecon\samples** directory and enter:

```
java com.ibm.vse.ejb.vsamexample.EJBPrepareData
java com.ibm.vse.ejb.vsamexample.EJBShowData
```

Step 3: Specify the EJB's Home Interface

In this step, the *EJB home interface* (described in Figure 134 on page 271) is specified for the *RecordBean*. For details of how to specify the home interface for other EJBs, refer to the VSE Connector Client online documentation (see "Using the Online Documentation Options" on page 28 for details).

The EJB's home interface inherits its methods from the interface *EJBHome*. It must define method signatures for:

- Any *create()* methods.
- The *findByPrimaryKey()* method.
- Other "finder" methods.

When an EJB is deployed, the WebSphere Application Server creates the home interface code, together with the remote interface and the service stub class, that are used to access the EJB.

The method signatures are implemented in the source file *RecordHome.java* as follows:

```
package com.ibm.vse.ejb.vsamexample;
import javax.ejb.*;
import java.rmi.*;

public interface RecordHome extends EJBHome
{
   public Record create(int empnum, String name, String password, int dept,
                        int hourly, int paytodate, int hoursnotpaid)
   throws java.rmi.RemoteException, javax.ejb.CreateException;

   public Record findByPrimaryKey(RecordPK pk)
   throws RemoteException, FinderException;
}
```

Step 4: Specify the EJB's Remote Interface

In this step, the *EJB remote interface* (described in Figure 134 on page 271) is specified for the *RecordBean*. For details of how to specify the remote interface for other EJBs, refer to the VSE Connector Client online documentation (see "Using the Online Documentation Options" on page 28 for details).

When an EJB is deployed, the WebSphere Application Server creates the remote interface code, together with the Home Interface and the service stub class, that are used to access the EJB.

The remote interface must provide:

- Inheritance from the interface *EJBObject*.
- Method signatures related to the EJB's business methods
  - The code for these methods is generated during the process of deploying the bean.
  - The method signatures are provided in the source file *Record.java*, as shown below.

```
package com.ibm.vse.ejb.vsamexample;
import javax.ejb.*;
import java.rmi.*;
import java.io.*;
import java.util.*;

public interface Record extends EJBObject
{
   public int getEMPNum() throws RemoteException;
   public String getPassword() throws RemoteException;
   public String getName() throws RemoteException;
   public int getDept() throws RemoteException;
   public int getHourly() throws RemoteException;
   public int getPayToDate() throws RemoteException;
   public int getTimeNotPaid() throws RemoteException;
   public void setDept(int dept) throws RemoteException;
   public void setEMPNum(int empnum) throws RemoteException;
   public void setName(String name) throws RemoteException;
   public void setHourly(int hourly) throws RemoteException;
   public void setPassword(String passwd) throws RemoteException;
   public void setPayToDate(int paytodate) throws RemoteException;
   public void setTimeNotPaid(int timenotpaid) throws RemoteException;
}
```

Step 5: Implement the RecordPK Class

The *RecordPK* class is a helper class, required by entity beans to provide objects that implement *java.io.Serializable*. Only serializable objects may be passed as arguments or as return values over RMI.

Serializable objects are returned, for example, by *ejbCreate()* or *ejbFindByPrimaryKey()*.

The code is implemented in source file **RecordPK.java**. You must specify the primary key class for entity beans, during the EJB deployment process (as described in "Step 8: Deploy the EJBs" on page 282).

```
package com.ibm.vse.ejb.vsamexample;

public class RecordPK implements java.io.Serializable
{
   public int empnum;
}
```

Step 6: Implement the EJB Code

The code provided here implements the sample EJB *RecordBean*. Methods are provided to:

- Access the *RecordBean*'s data.
- Connect to the remote VSE/ESA host.
- Obtain data from the database.
- Update the database.

This section includes only parts of the code. If you require the complete source code, refer to the file **RecordBean.java** provided with the VSE Connector Client (see "Using the Online Documentation Options" on page 28 for details).

```
...
public class RecordBean implements EntityBean
{
   /* the indexes of the fields in the backend map */
   private int EMPNUM_INDEX = 0;
   private int NAME_INDEX = 2;
```

```
private int PASSWORD_INDEX = 1;
private int DEPT_INDEX = 3;
private int HOURLY_INDEX = 4;
private int PAYTODATE_INDEX = 5;
private int TIMENOTPAID_INDEX = 6;

private transient VSESystem system;
private transient VSEVsamRecord record;

private transient EntityContext ctx;
private int empnum = 0;
...
```

Step 6.1: Implement the Methods of the EntityBean Interface

The methods described here implement the methods of interface *EntityBean*. These methods:

1. Access the remote database.
2. Fill the given EJB instance with one VSAM record.

The example uses the *ejbFindByprimaryKey()* method, because the EJB has been implemented using *bean-managed* persistence[1]. As a result, the EJB implements all the database calls that are required to send the object out to the database, and to read it back in again.

The logic for accessing the database is implemented in methods *ejbLoad()* and *ejbStore()*. The *ejbCreate()* method creates a new employee record with the parameters shown below, and returns its primary key.

```
public RecordPK ejbCreate(int empnum, String name, String password,
                          int dept, int hourly, int paytodate,
                          int hoursnotpaid)
throws CreateException
{
   RecordPK rpk= new RecordPK();
   rpk.empnum = empnum;
   try {
      rpk = this.ejbFindByPrimaryKey(rpk);
   }
   catch (Exception e)
   {
      try {
         system = connectVSE();
         record = new VSEVsamRecord(system, "VSESP.USER.CATALOG",
                                    "EJB.VSAM.EXAMPLE", "EJBMAP");
         record.setKeyField(EMPNUM_INDEX, new Integer(empnum));
         record.setField(NAME_INDEX, name);
         record.setField(PASSWORD_INDEX, password);
         record.setField(DEPT_INDEX, new Integer(dept));
         record.setField(HOURLY_INDEX, new Integer(hourly));
         record.setField(PAYTODATE_INDEX, new Integer(paytodate));
         record.setField(TIMENOTPAID_INDEX, new Integer(hoursnotpaid));
         record.add();
         rpk.empnum = empnum;
         this.empnum = empnum;
         return (rpk);
      }
      catch (IOException ioe)
      {
```

1. The opposite of bean-managed persistence is called *container-managed* persistence. Here, the EJB developer does not have to bother with synchronizing with the database. Instead, the entity bean's deployment descriptor specifies the fields that should be managed by the EJB container. At runtime, the container calls the *ejbLoad()* and *ejbStore()* methods when required, but the EJB developer must not provide any code for these methods.

```
            ...
         }
      }
   }
```

The following three methods must be implemented, because the EJB implements *bean-managed* persistence:

- ejbLoad()
- ejbStore()
- ejbFindByPrimaryKey()

```
/**
 * fill the EJB with new data from the remote database.
 */
public void ejbLoad() throws RemoteException
{
   RecordPK pk = (RecordPK) ctx.getPrimaryKey();
   system = connectVSE();
   try {
      record = findRecord(pk.empnum, system);
   }
   catch (FinderException e)
   {
      ...
   }
}

/**
 * make a change permanent in the remote database.
 */
public void ejbStore() throws RemoteException
{
   try {
      record.commit();
   }
   catch (IOException e)
   {
      ...
   }
}

/**
 * looks up the record and returns pk back to the caller.
 * If the record is not found, a FinderException is thrown
 * We have to provide this method, because we implement the EJB
 * using bean-managed persistence.
 */
public RecordPK ejbFindByPrimaryKey(RecordPK pk)
throws FinderException, RemoteException
{
   VSESystem system = connectVSE();
   VSEVsamRecord record = findRecord(pk.empnum, system);
   return (pk);
}
```

The methods described here are also contained in the interface *EntityBean*. There are a number of other interface methods that are not shown here: if you require the complete source code, refer to the VSE Connector Client online documentation (see "Using the Online Documentation Options" on page 28 for details).

Note: The *ejbRemove()* method only deletes the local instance of the VSAM record that is related to this EJB. To commit the change in the remote database, a call to *ejbStore()* is required.

```
public void ejbRemove() throws RemoteException
{
   try {
      record.delete();
   }
   catch (IOException e)
   {
      throw new RemoteException("" + e);
   }
}

public void ejbActivate () throws RemoteException
{
   system = connectVSE();
   try {
      findRecord(empnum, system);
   }
   catch (FinderException e)
   {
      throw new RemoteException("FinderException");
   }
}
```

Step 6.2: Access VSE/ESA Host and Get Records from the Database

In this section, the methods are implemented to:

- Access the VSE/ESA host.
- Get records from the database.

```
/**
  * Create connection specification. The connection spec holds
  * information about the physical host connection.
  */
 public VSESystem connectVSE() throws RemoteException
 {
    VSEConnectionSpec spec;
    VSESystem system;
    try
    {
       spec = new VSEConnectionSpec(InetAddress.getByName("9.164.155.95"),
                                       2893, "sysa","mypassw");

       // This is application server dependent
       Properties p = new Properties();
       p.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
             "com.ibm.websphere.naming.WsnInitialContextFactory");
       p.put(javax.naming.Context.PROVIDER_URL, "iiop:///");

       Context ctx = new InitialContext(p);
       spec.setJNDIContext(ctx);
       spec.setJNDIName("eis/VSEConnector");

       /* Stay logged on with this user for lifetime of this connection */
       spec.setLogonMode(true);

       /* Create VSE system instance with this connection */
       system = new VSESystem(spec);
    }
    catch (java.net.UnknownHostException e)
    {
       throw new RemoteException("Unknown host");
    }
 }

 /**
  * find and load the record with pk empnum from VSAM.
```

```
 */
protected VSEVsamRecord findRecord(int empnum, VSESystem system)
throws FinderException
{
   try {
      VSEVsamRecord rec = new VSEVsamRecord(system, "VSESP.USER.CATALOG",
                                 "EJB.VSAM.EXAMPLE", "EJBMAP");
      rec.setKeyField(EMPNUM_INDEX, new Integer(empnum));
      if (rec.isExistent())
      {
         rec.refresh();
         return rec;
      }
      else
      {
         throw new FinderException("Record not found.");
      }
   }
   catch (IOException e)
   {
      throw new FinderException("IOException");
   }
}
```

Step 6.3: Implement the Set & Get Methods to Access the Data Fields

In this section, the set and get methods are implemented in order to access the data fields (columns). These methods are the interface that EJB clients use, in order to access the encapsulated data of the EJB.

For each column you wish to access, you must provide a *get* method.

For all columns that are to be updated, you must provide a *set* method.

Notes:

1. These methods do not access the remote database. Instead, they simply return or modify the EJB's internal data.
2. If you want to *permanently* update changes, you must use the interface method *ejbStore()*.

```
/**
 * returns the employee number for the record.
 */
public int getEMPNum() throws RemoteException
{
   try {
      return ((Integer)record.getField(EMPNUM_INDEX)).intValue();
   }
   catch (IOException e)
   {
      throw new RemoteException("IOException");
   }
}


/**
 * set the employee number for the current record
 */
public void setEMPNum(int empnum) throws RemoteException
{
   try
   {
      record.setField(EMPNUM_INDEX, new Integer(empnum));
   }
   catch (IOException e)
```

```
        {
           throw new RemoteException("IOException");
        }
        return;
     }
     ...
}
```

Step 7: Compile the Java Source Files

You must compile the Java source code described in this section, in order to setup the EJB sample. Run the compile jobs from directory **\vsecon\samples**.

Note: You must include certain JAR files containing EJB-related classes, in your local classpath. For details, refer to the:

- VSE Connector Client online documentation (see "Using the Online Documentation Options" on page 28 for details).
- The documentation references provided within the section "Installing the WebSphere Application Server" on page 23.

The VSE Connector Client provides already-compiled class files, for all sample Java source files. However, the following source files belong to the EJB example.

1. Job to compile the utilities to create and display the sample data:

```
javac com\ibm\vse\samples\EJBPrepareData.java
javac com\ibm\vse\samples\EJBShowData.java
```

2. Job to compile the EJB source files:

```
javac com\ibm\vse\samples\Employer.java
javac com\ibm\vse\samples\EmployerHome.java
javac com\ibm\vse\samples\EmployerBean.java
javac com\ibm\vse\samples\Employee.java
javac com\ibm\vse\samples\EmployeeHome.java
javac com\ibm\vse\samples\EmployeeBean.java
javac com\ibm\vse\samples\Record.java
javac com\ibm\vse\samples\RecordHome.java
javac com\ibm\vse\samples\RecordBean.java
```

3. Job to compile the EJB clients:

```
javac com\ibm\vse\samples\EJBApplet.java
javac com\ibm\vse\samples\EJBClient.java
```

Step 8: Deploy the EJBs

The method you use to deploy your EJBs differs according to the version of the WebSphere Application Server you are using. The VSE Connector Client online documentation provides detailed instructions on how to deploy EJBs for various versions of the WebSphere Application Server (see "Using the Online Documentation Options" on page 28).

For details of how to deploy EJBs:

1. Proceed to the main window of the Online Documentation, as shown in Figure 5 on page 28.
2. Select **Programming Concepts**.
3. Select **EJBs**.
4. Select **An example to represent VSAM records**.
5. Select either:
   - **Deploy the EJBs on WebSphere 3.*x***
   - **Deploy the EJBs on WebSphere 4.*x***

Step 9: Access the EJBs from an EJB Client

Prerequisites for Accessing the EJBs from an EJB Client

Before you can begin accessing the EJBs you have created from an EJB client, these conditions must be met:

1. You have successfully created the VSAM cluster required for running the sample, as described in "Step 1: Define the Sample's VSAM Cluster" on page 275.
2. You have filled the above cluster with sample data, for example by using the utility *EJBPrepareData*, as described in "Step 2: Create the Record Layout for Employees" on page 275.
3. TCP/IP for VSE/ESA must be running on the VSE/ESA host (as described in "Configuring and Activating TCP/IP for VSE/ESA" on page 21).
4. The VSE Connector Server must be running on the VSE/ESA host (as described in "Starting the VSE Connector Server" on page 36).
5. You have specified the correct IP address, a VSE user ID, and password, in **RecordBean.java** (as described in "Using the Online Documentation Options" on page 28).
6. The IBM HTTP Server must be running on the middle-tier (as described in "Configuring and Activating the VSE HTTP Server" on page 21).
7. The WebSphere Application Server must be running on the middle-tier (refer to the documentation references provided in "Installing the WebSphere Application Server" on page 23).
8. The sample EJBs are running (as described in "Step 8: Deploy the EJBs" on page 282).

How EJBs Are Accessed from an EJB Client

An EJB client accesses the business logic contained in the EJBs, in this general way:

1. The EJB client uses a naming service to locate the EJB's home interface.
2. The naming service (usually the Java Naming and Directory Interface, JNDI) returns a reference to an object that implements the EJB's *home interface* (described in "How an EJB Client Accesses EJBs" on page 271).
3. The EJB client makes a call on the EJB's home interface, to gain access to the EJB's *remote interface* (described in "How an EJB Client Accesses EJBs" on page 271).
4. The EJB client make calls to the EJB's business methods against the remote interface.

However, the actual code you use to access an EJB from an EJB client will vary according to the type of Web Application Server you are using.

Figure 136 on page 284 shows how the *EJBClient* sample application accesses the EJBs via their home interfaces (shown as **H**), and remote interfaces (shown as **R**). If you wish to see the complete code for accessing the EJBs using the EJB Client, refer to the VSE Connector Client online documentation.

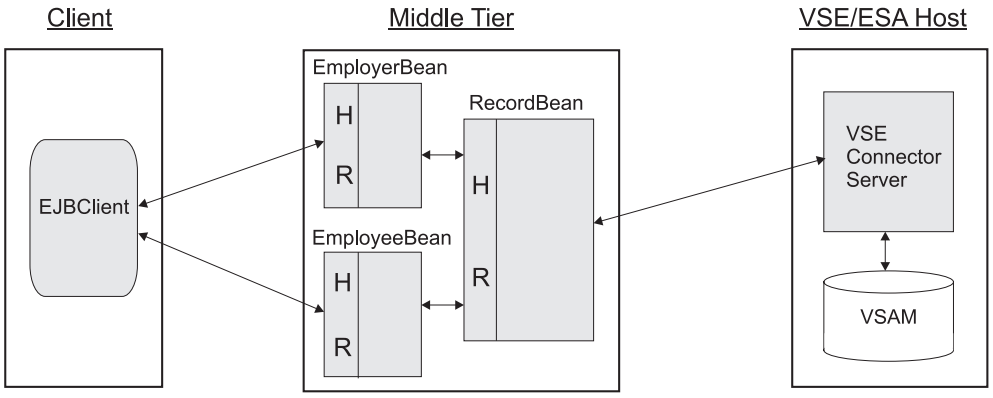


Figure 136. How the EJB Client Accesses EJBs in the Provided Example

The two session beans are in fact themselves EJB clients. When accessing the *RecordBean*, they perform the same processing of looking up the home interface and remote interface, as the EJB client does when it gets access to the session beans.

The session beans implement two different views on the same data, by including or not a particular column into a view. In addition, however, the session beans specify access rights for specific columns. For example, the *EmployeeBean* might only be able to read a given column, whereas the *EmployerBean* might also be able to update this column.

Sample EJB Client Source Code for Accessing EJBs from an EJB Client

This section shows the source code for the sample EJBClient application, that is part of the VSE Connector Client. The code is based upon the use of the *WebSphere Application Server* as the Web Application Server type.

```
public class EJBClient
{
   public static void main(String[] argv)
   {
      try
      {
         EmployerHome employerh;
         Properties p = new Properties();  1
         p.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY,
              "com.ibm.websphere.naming.WsnInitialContextFactory");
         p.put(javax.naming.Context.PROVIDER_URL, "iiop:///");
         InitialContext ic = new InitialContext(p);  2
         Object obj = (Object) ic.lookup("EmployerBean");  3
         if (obj instanceof org.omg.CORBA.Object)  4
         {
            employerh =
                (EmployerHome)(javax.rmi.PortableRemoteObject.narrow(
                         (org.omg.CORBA.Object)obj, EmployerHome.class));
         }
         else
         {
            System.out.println(
                   "Did not get an org.omg.CORBA.Object from lookup().");
            return;
         }
         Employer employer = employerh.create(1003, "newpass3");  5
         System.out.println("employer.calcPayToDate() = " +  6
                                              employer.calcPayToDate());
         Vector v = employer.getEMPInfo();
         System.out.println("employer.getEMPInfo() :");
         for (int i=0;i<v.size();i++)
            System.out.println("  " + v.elementAt(i).toString());
      }
      catch (Exception e)
      {
         ...
      }
   }
}
```

Figure 137. Example of EJB Client Code

The numbers below refer to the numbers in Figure 137:

1. Define properties. Specifying **iiop:///** as the PROVIDER_URL causes the EJB client to search for a name server on the local host listening on port number 900, which is the default for the WebSphere Application Server. In a real environment you would specify a *complete* URL, such as:

   ```
   iiop://bankserver.mybank.com:9019
   ```

2. The initial context object is instantiated.

3. The home interface of the EmployerBean is looked up. The string used here to identify the EJB, is stored in the EJB's deployment descriptor. You can also view the EJB's deployment properties by using the *WebSphere Administrative Console*.

4. Cast the returned object to the EJB's home interface class. After an object is returned by the lookup method, you must use the static method *PortableRemoteObject.narrow()* to obtain an EJB home object for the specified EJB.

5. Create a remote interface object to obtain access to the EJB's business logic methods. This call to the remote interface *create()* method, invokes the *ejbCreate()* method of the EJB.

6. Access the EJB's business logic methods. For example, get some properties of employer that has the number 1003. Please be aware however, that the EJB is always accessed via the *remote* object (in this example, "Employer.class"). An EJB client cannot call methods that belong to the EJB directly (in this example, the **EmployerBean.class**). The above example employee number and password are taken from the EJB sample provided with the VSE Connector Client, which also provides utilities to fill the sample VSAM cluster with some sample data.

Chapter 23. Extending the Java-Based Connector

This chapter describes how you can extend the Java-based connector in 2-tier and 3-tier environments, by writing your own "plugins". It makes much use of the "Date and Time plugin" example, which obtains and displays a VSE/ESA host's current date and time.

By writing your own plugins, you can provide additional access to:
- *resources* (such as accessing VSAM files that are currently open in CICS).
- *applications* (such as starting CICS transactions, or starting Vendor applications).

You can write your plugin to consist of:
- A *client plugin* only. You do so by extending the VSE Java Beans class library. In this case, your plugin is a JavaBean which itself uses the (existing) JavaBeans to access data or provide services. **Note:** This JavaBean does *not* implement *VSEPlugin* (see below for details).
- A *client plugin* and a *server plugin*. In this case, your server plugin extends the capabilities of the VSE Connector Server by providing additional functionality that is required by your client plugin.

Your server plugin consists of a VSE PHASE that is a plugin to the VSE Connector Server. This PHASE (which you write using LE/VSE-C) is loaded during startup of the VSE Connector Server. The PHASE must also provide a well-defined interface to enable the VSE Connector Server to call the implemented functions.

Your client plugin consists of one or more JavaBeans written by yourself. These JavaBeans implement the Java class *VSEPlugin*, which is contained in the package **com.ibm.vse.connector**. For further details of how you can write your own JavaBeans, refer to the online documentation.

This chapter contains these main sections:

- "Implementing a Server Plugin" on page 288
- "Implementing a Client Plugin" on page 303
- "General Considerations When Designing Your Plugin" on page 305

© Copyright IBM Corp. 2000, 2003

Implementing a Server Plugin

A server plugin consists of a set of *callback functions* that are called by the VSE Connector Server in a given sequence. Each server plugin consists of a PHASE that you write using LE/VSE-C, which is loaded during VSE Connector Server startup. Your plugin must provide a well-defined interface to allow the VSE Connector Server to call the functions implemented in your server plugin. These callback functions (PluginMainEntryPoint, SetupPlugin, and so on) are described later in this section.

An overview of how the functions are called by the VSE Connector Server, is provided in Figure 138:

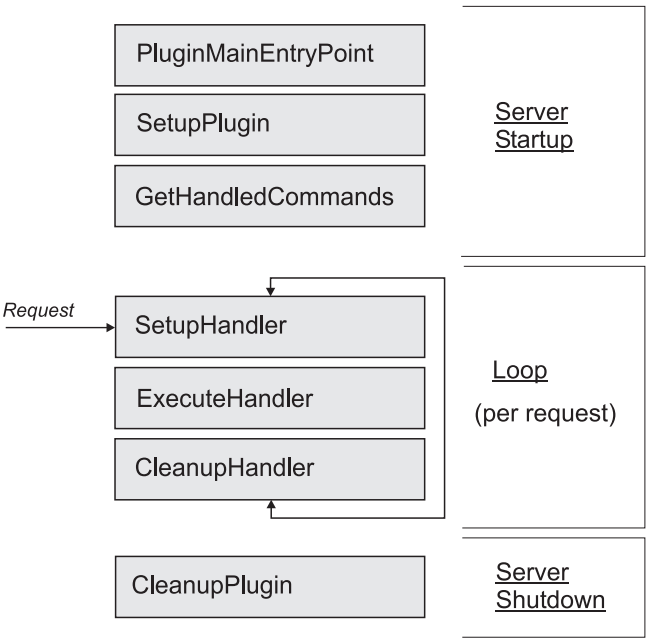


Figure 138. Overview of How a Server Plugin's Functions Are Called

A *typical sequence* of how functions are called by the VSE Connector Server, is provided in Figure 139 on page 289, Figure 140 on page 290, and Figure 141 on page 291:

- Figure 139 describes the server plugin's functions that are called during the startup of the VSE Connector Server.
- Figure 140 and Figure 141 describe the server plugin's functions that are called when the VSE Connector Server receives a request from a client, and after the request processing is finished, during shutdown of the VSE Connector Server.

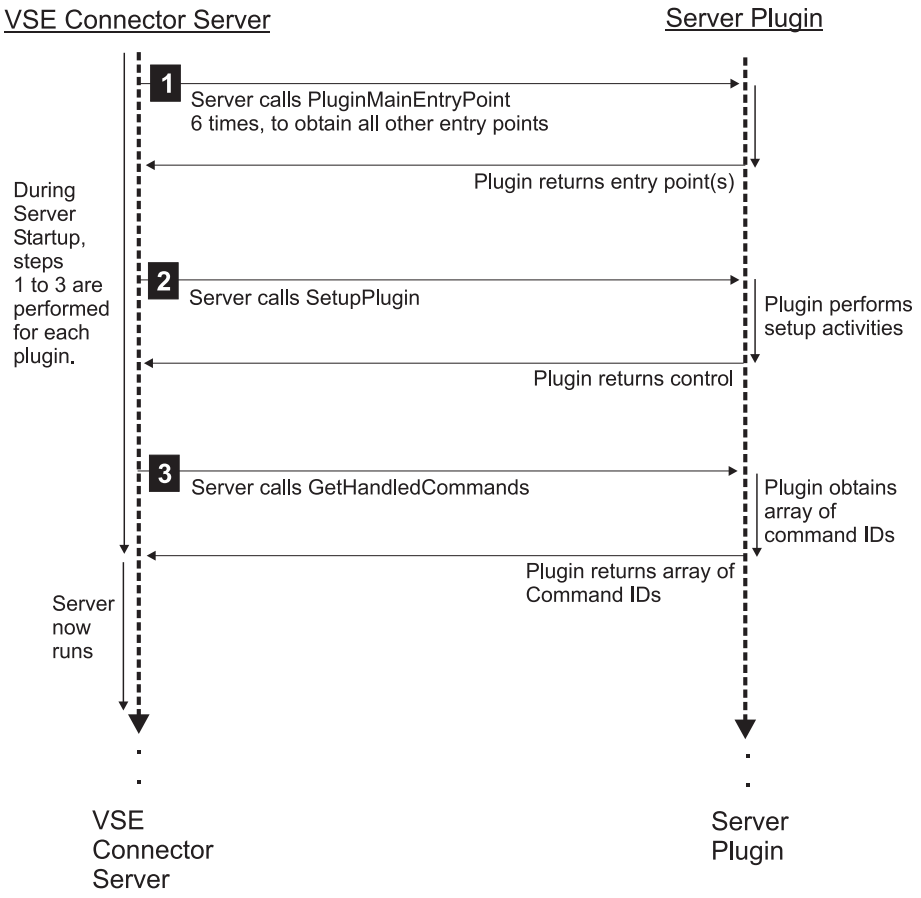


Figure 139. How a Plugin's Functions Are Called During Startup

1 By loading the server plugin phase (CDLOAD), the VSE Connector Server obtains the entry point of the server plugin (*PluginMainEntryPoint*). This entry point is then called six times to obtain the entry points of the other server plugin functions. Each call returns the entry point of a function specified by the parameter *iFuncID*. When this step is completed, the server has a list of the entry points in the plugin phase.

2 The *SetupPlugin* function is the first function that the VSE Connector Server calls, after the server plugin has been loaded. This function is used to initialize all request-independent resources (allocate storage, open database connections, and so on).

3 The VSE Connector Server calls the *GetHandledCommands* function immediately after calling the *SetupPlugin* function. *GetHandledCommands* returns a list of IDs that are accepted by the server plugin. The VSE Connector Server later uses this information in order to direct a request to a particular server plugin. Therefore, command IDs must be unique across all server plugins. The VSE Connector Server accepts a predefined range of command IDs. Refer to the header file IESPLGIN.H file for details.

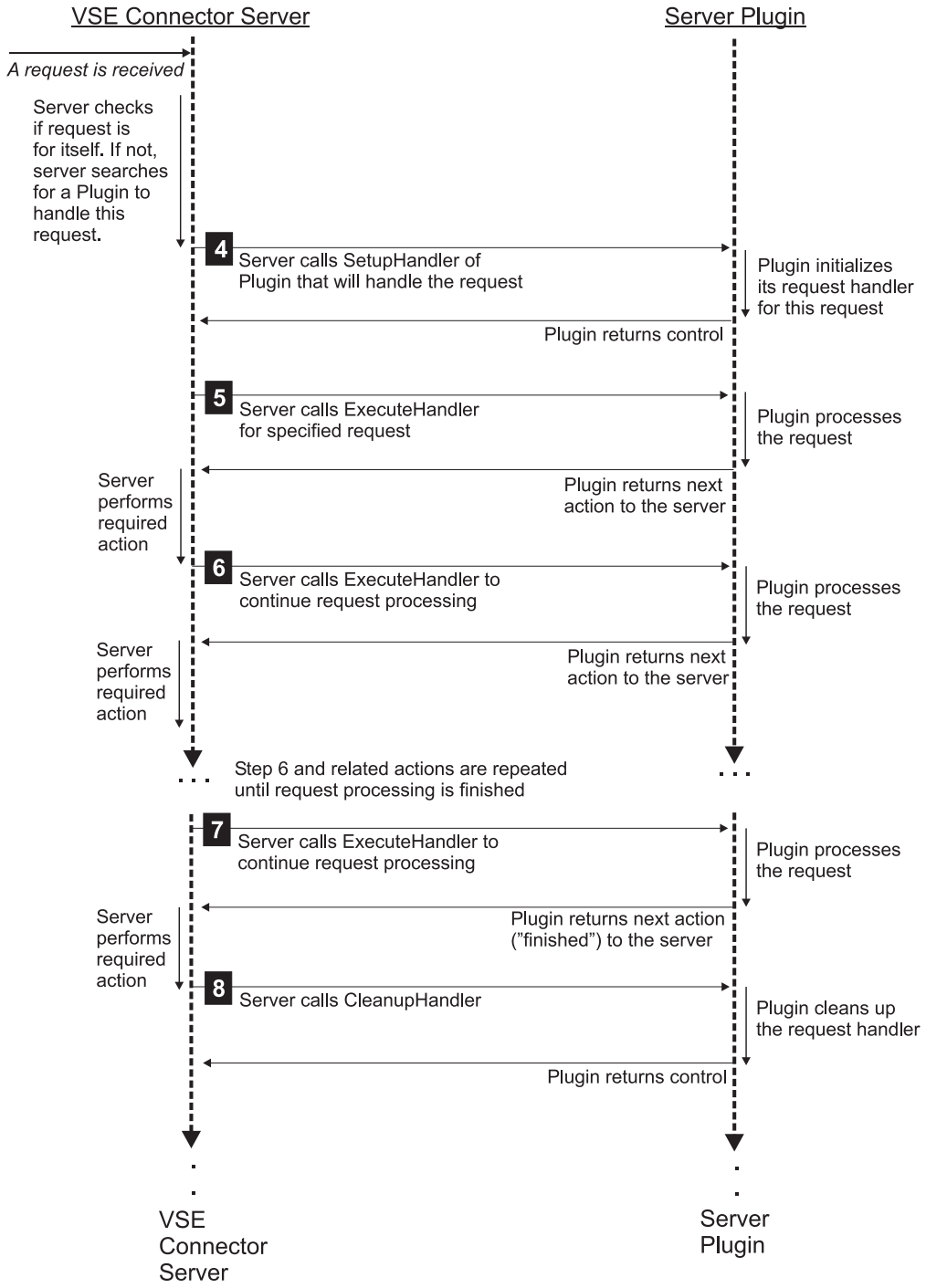


Figure 140. How a Plugin's Functions Are Called When a Request Is Received

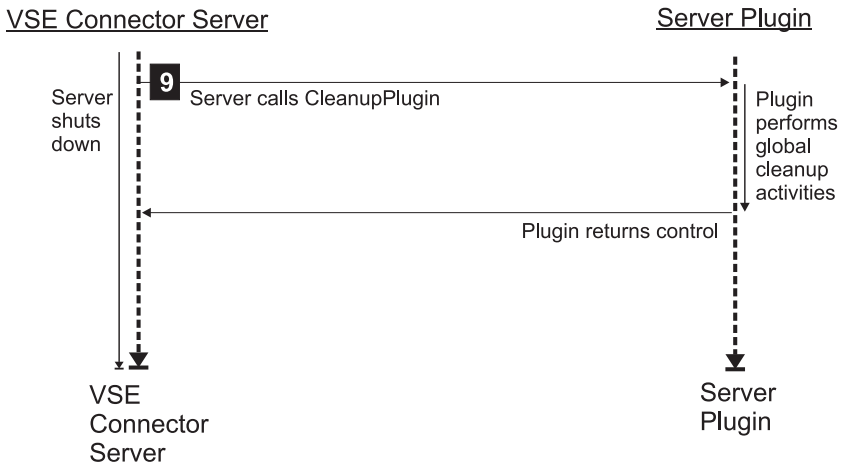


Figure 141. Overview of How a Plugin's Functions Are Called During Server Shutdown

4 After a request has been received from a client, the VSE Connector Server first determines which server plugin should handle the request, and calls the *SetupHandler* function of the selected server plugin. The server plugin allocates a *request handler* for this particular request, consisting of a control block used to process the request. The control block is allocated by the *SetupHandler* function and is specific to a request.

5 To process the specific request, the VSE Connector Server calls the *ExecuteHandler* function of the server plugin. The server plugin normally splits this code into small chunks, each of which is called by the VSE Connector Server during the next calls to the *ExecuteHandler* function (steps '6' and '7' below). The control block allocated by the *SetupHandler* function is passed for each call to the *ExecuteHandler* function. This control block can also be used to store local variables. The VSE Connector Server then processes the action returned by the *ExecuteHandler* function, which can be sending or receiving data, or waiting for timers or ECBs.

6 The VSE Connector Server again calls the *ExecuteHandler* function of the server plugin, to continue processing the request. The control block allocated by the *SetupHandler* function is again passed to the *ExecuteHandler* function. The VSE Connector Server then processes the action returned by the *ExecuteHandler* function.

7 The VSE Connector Server calls the *ExecuteHandler* function of the server plugin, to continue processing the request. The VSE Connector Server then processes the action returned by the *ExecuteHandler* function, which indicates that the server plugin has finished all processing (all actions are completed).

8 The VSE Connector Server calls the *CleanupHandler* function to allow the server plugin to cleanup all resources allocated for this specific request. If an unexpected error occurs during request processing (for example a network connection is broken), the *CleanupHandler* function should be able to cleanup all resources that were used up to this point of time.

9 The VSE Connector Server is shut down, and it calls the *CleanupPlugin* function of all the loaded server plugins.

Implementing a PluginMainEntryPoint Function

The VSE Connector Server calls this function to establish the entry point for a plugin. The *PluginMainEntryPoint* function fetches all other entry points of the plugin's functions. The VSE Connector Server calls *PluginMainEntryPoint* several times. Each call then returns the entry point of the plugin function specified using the *iFuncID* parameter.

Notes:

1. You should not rename the function *PluginMainEntryPoint*. This is because the C-Header file IESPLGIN.H contains a `#pragma linkage` statement which defines this function as `fetchable`.
2. If you do decide to rename this function, you must provide your *own* `#pragama linkage` statement:

   ```
   #pragma linkage(MyEntryPointFunction,fetchable)
   ```

   which forces *MyEntryPointFunction* to be used as entry point of your PHASE.

Here is an example of how the *PluginMainEntryPoint* function can be coded:

```
/*********************************************************************
* Function:    PluginMainEntryPoint Main entry point of the Plugin     *
*                           PHASE.                                     *
* Parameters:  iFuncID      Function ID to get the entrypoint for     *
* Return:      Entrypoint of the requested Function or NULL           *
*********************************************************************/
FUNC_PTR PluginMainEntryPoint(int iFuncID)
{
  switch(iFuncID)
  {
    case PLGFUNC_SETUPPLUGIN:
      return((FUNC_PTR)fetchep((FETCH_PTR)SetupPlugin));
      break;
    case PLGFUNC_CLENUPPLUGIN:
      return((FUNC_PTR)fetchep((FETCH_PTR)CleanupPlugin));
      break;
    case PLGFUNC_GETHANDLEDCMDS:
      return((FUNC_PTR)fetchep((FETCH_PTR)GetHandledCommands));
      break;
    case PLGFUNC_SETUPHANLDER:
      return((FUNC_PTR)fetchep((FETCH_PTR)SetupHandler));
      break;
    case PLGFUNC_EXECUTEHANLDER:
      return((FUNC_PTR)fetchep((FETCH_PTR)ExecuteHandler));
      break;
    case PLGFUNC_CLEANUPHANLDER:
      return((FUNC_PTR)fetchep((FETCH_PTR)CleanupHandler));
      break;

    default:
      return(NULL);
      break;
  };
  return(NULL);
};
```

Figure 142. Sample Code for Implementing PluginMainEntryPoint Function

Implementing a SetupPlugin Function

The VSE Connector Server calls this function *after* a server plugin has been loaded. The *SetupPlugin* function:

- Performs all the initialization steps required for the plugin.
- Gets a pointer to a PLUGIN_INFO block, which contains information about:
  - The VSE/ESA system parameters that are specified in the plugin configuration member.
  - Entry points of utility functions that the VSE Connector Server provides.

You can also allocate your own plugin private-control block, and pass the pointer of this block back to the VSE Connector Server. The VSE Connector Server passes this pointer to each function belonging to your plugin, that is called from this point-of-time onwards.

Notes:

1. If you define the same plugin more than once, the plugin will be loaded more than once.
2. The *SetupPlugin* is called separately for each plugin "instance".
3. All occurrences of plugin private-control blocks that you allocate, are separated from each other.

Here is an example of how the *SetupPlugin* function can be coded:

```
/*********************************************************************
* Function:    SetupPlugin   Sets up the Plugin. The Plugin may alloc  *
*                            resources used for command processing.    *
*                            The Plugin can allocate a PluginPrivate   *
*                            Data Area, that is passed to each function*
* Parameters:  lplpPluginPrivate Pointer to a Pointer to the Plugins   *
*                            private Data area. This pointer should be *
*                            set by the Plugin.                        *
*              lpPluginInfo  Pointer to a struct PLUGIN_INFO containing*
*                            several information about the server      *
* Return:      Error values see PLGERR_xxx values                      *
*********************************************************************/
int SetupPlugin(void** lplpPluginPrivate,PLUGIN_INFO* lpPluginInfo)
{
  SAMPLE_PLUGIN*  lpSamplePlugin;

  /* Parameter checking ... */
  if(lplpPluginPrivate==NULL || lpPluginInfo==NULL)
    return(PLGERR_INVALID_PARAM);

  /* Allocate Plugin-Private Data area */
  *lplpPluginPrivate = malloc(sizeof(SAMPLE_PLUGIN));
  if(*lplpPluginPrivate==NULL)
    return(PLGERR_NULL_POINTER);

  /* Fill the Plugin-Info */
  lpPluginInfo->dwPluginVersion = PLUGIN_VERSION;
  strcpy(lpPluginInfo->szDescription,PLUGIN_DESCRIPTION);

  /* Initialize the Plugin ... */

  lpSamplePlugin = (SAMPLE_PLUGIN*)*lplpPluginPrivate;
```

Figure 143. Sample Code for Implementing the SetupPlugin Function (Part 1 of 2)

```
  /* Open SYSLOG as trace output */
  lpSamplePlugin->lpOutput = fopen("DD:SYSLOG","w");
  if(lpSamplePlugin->lpOutput==NULL)
    return(PLGERR_NULL_POINTER);

  fprintf(lpSamplePlugin->lpOutput,"SetupPlugin Called\n");

  return(PLGERR_NO_ERROR);
};
```

Figure 143. Sample Code for Implementing the SetupPlugin Function (Part 2 of 2)

Implementing a CleanupPlugin Function

The VSE Connector Server calls this function *before* a server plugin has been unloaded. It is the counterpart to the *SetupPlugin* function.

The *SetupPlugin* function performs all required cleanup steps. The plugin now de-allocates plugin private-control blocks that were previously allocated.

Here is an example of how the *CleanupPlugin* function can be coded:

```
/*************************************************************************
* Function:    CleanupPlugin Cleans up the Plugin.                      *
* Parameters:  lpPluginPrivate Pointer to the Plugins private data area*
* Return:      Error values see PLGERR_xxx values                       *
*************************************************************************/
int CleanupPlugin(void* lpPluginPrivate)
{
  SAMPLE_PLUGIN*  lpSamplePlugin;

  /* Parameter checking... */
  if(lpPluginPrivate==NULL)
    return(PLGERR_INVALID_PARAM);

  /* Cleanup the Plugin ... */

  lpSamplePlugin = (SAMPLE_PLUGIN*)lpPluginPrivate;

  fprintf(lpSamplePlugin->lpOutput,"CleanupPlugin Called\n");

  /* Close trace output */
  fclose(lpSamplePlugin->lpOutput);

  /* Free the Plugin Private Data */
  free(lpPluginPrivate);

  return(PLGERR_NO_ERROR);
};
```

Figure 144. Sample Code for Implementing the CleanupPlugin Function

Implementing a GetHandledCommands Function

The VSE Connector Server calls this function immediately after *SetupPlugin* has completed. It is used to determine which commands (requests), and how many commands (requests), are currently being handled by a server plugin.

GetHandledCommands passes an array of command-Ids back to the VSE Connector Server.

Here is an example of how the *GetHandledCommands* function can be coded:

```
/*********************************************************************
* Function:    GetHandledCommands Returns a list of CommandIDs that are*
*                           handled by this Plugin.                  *
* Parameters:  lpPluginPrivate Pointer to the Plugins private data area*
*              lpiNumCommands Pointer to a int that should be set by   *
*                           Plugin to the number of Commands          *
*              lpCommandIDs  Pointer to a array of ints. Each element  *
*                           defines one CommandID                     *
* Return:      Error values see PLGERR_xxx values                      *
*********************************************************************/
int GetHandledCommands(void* lpPluginPrivate,
                       int* lpiNumCommands,
                       int** lpCommandIDs)
{
  SAMPLE_PLUGIN*  lpSamplePlugin;

  /* Parameter Checking ... */
  if(lpPluginPrivate==NULL ||
     lpiNumCommands==NULL ||
     lpCommandIDs==NULL)
    return(PLGERR_INVALID_PARAM);

  lpSamplePlugin = (SAMPLE_PLUGIN*)lpPluginPrivate;

  fprintf(lpSamplePlugin->lpOutput,"GetHandledCommands Called\n");

  /* There are 2 Command sHandled by the Sample */
  *lpiNumCommands = 2;

  /* Return a Pointer to the List of Command-IDs */
  *lpCommandIDs = &HandledCommandIDs[0];

  return(PLGERR_NO_ERROR);
};
```

Figure 145. Sample Code for Implementing the GetHandledCommands Function

The `HandledCommandIDs` array is defined as follows:

```
int   HandledCommandIDs[2] = { CMD_SAMPLE_TIME,
                               CMD_SAMPLE_DATE };
```

Implementing a SetupHandler Function

The VSE Connector Server calls this function each time a request is received that must be handled by a server plugin:

1. This function initializes the request handler for the a request.
2. The VSE Connector Server passes a pointer to a CMD_INFO control block to this function.
3. The CMD_INFO control block contains information about the command to handle (for example the command ID of the request). You can allocate a handler private-control block and pass a pointer to it back to the server. The server engine will pass this pointer to each function call belonging to the same request.

Note: It is possible that the same request is executed multiple times. For each 'instance' of request the function *SetupHandler* is called separately. That is, the handler private-control blocks you allocate are separated from each other (you must account for this in your coding).

You must also ensure that your handler is implemented as *reentrant*.

Here is an example of how the *SetupHandler* function can be coded:

```
/*********************************************************************
* Function:    SetupHandler  Sets up a Command Handler.               *
* Parameters:  lpPluginPrivate Pointer to the Plugins private data area*
*              lplpCommandPrivate Pointer to a Pointer to the Handlers *
*                           private data area                         *
*              lpCmdInfo     pointer to a struct CMD_INFO defining the *
*                           actual command                            *
* Return:      Error values see PLGERR_xxx values                     *
*********************************************************************/
int SetupHandler(void* lpPluginPrivate,
                 void** lplpCommandPrivate,
                 CMD_INFO* lpCmdInfo)
{
  SAMPLE_PLUGIN*  lpSamplePlugin;

  /* parameter Checking ... */
  if(lpPluginPrivate==NULL ||
     lplpCommandPrivate==NULL ||
     lpCmdInfo==NULL)
    return(PLGERR_INVALID_PARAM);

  lpSamplePlugin = (SAMPLE_PLUGIN*)lpPluginPrivate;

  /* Initialize the Handler... */
  fprintf(lpSamplePlugin->lpOutput,"SetupHandler Called\n");

  /* check which Command we should handle */
  switch(lpCmdInfo->Command.dwCommand)
```

Figure 146. Sample Code for Implementing the SetupHandler Function (Part 1 of 2)

```
  {
    case CMD_SAMPLE_TIME:  /* We are Handling Time-Command */
      /* allocate Command Private datat area */
      ...
      break;

    case CMD_SAMPLE_DATE:   /* We are Handling Date Command */
      /* allocate Command Private datat area */
      ...
      break;
  };
  return(PLGERR_NO_ERROR);
};
```

Figure 146. Sample Code for Implementing the SetupHandler Function (Part 2 of 2)

Implementing an ExecuteHandler Function

The VSE Connector Server calls this function to execute the processing of the handler. The VSE Connector Server passes the plugin's private-control block and the handler's private-control block to this function. A pointer is also passed to the CMD_INFO block, which contains information about the command to handle. This block is also used to tell the VSE Connector Server what action it should take after the *ExecuteHandler* function has returned control to it.

The VSE Connector Server runs in a single task only. Therefore, your server plugin is also executed in the one (main) task. As a result, you should ensure that the *ExecuteHandler* function (and all other functions) return control to the VSE Connector Server as quickly as possible. To ensure this, you should split the request processing into several small chunks, where each chunk is executed in a minimum amount of time. For an example of how to split the request processing into several chunks, refer to the "Date and Time Sample plugin" (described in "Using the IBM-Supplied Server Plugin Example" on page 302).

Your request handler can set up a action command in the CMD_INFO block and return control to the VSE Connector Server, if it must wait:

- for an ECB (Event Control Block)
- for a timer
- to receive data from the client
- to send data to the client

The VSE Connector Server will wait for the event you specified and will pass control to your plugin, and will pass control to your plugin, after the event has occurred. Therefore, your plugin should not itself wait, since this would block all parallel-executed tasks being performed by the VSE Connector Server.

You also can set up an action command if you want to send or receive data over the network. Your plugin does not have access to network services directly. Instead, the VSE Connector Server handles the sending or receiving of data over the network, on behalf of your plugin.

Here is an example of how the *ExecuteHandler* function can be coded:

```
...
/* Check which Command to handle */
switch(lpCmdInfo->Command.dwCommand)
{
  case CMD_SAMPLE_TIME:  /* Time Command */
    ...
    break;

  case CMD_SAMPLE_DATE:  /* Date Command */
    ...
    break;

  ...
};
```

Figure 147. Sample Code for Implementing the ExecuteHandler Function

The code fragment of Figure 148 shows how to distinguish between multiple requests within one plugin. To split the request processing into several small chunks, the CMD_INFO block contains a field *iState* which can be used to store the actual state of the handler:

```
/* Check in which state we are (0 for first call) */
switch(lpCmdInfo->iState)
{
  case 0:  /* Initial State */
    /* Verify that the Command is correctly sent */
    if(lpCmdInfo->Command.dwDataSize!=0)
      return(PLGERR_PROTOCOL_ERROR);

    /* OK, start processing */
    ...
    /* return from handler without any special action */
     lpCmdInfo->iAction = PLGACT_NOTHING;
     lpCmdInfo->iState = 1;
     break;

   case 1: /* Send the Rresponse */
     /* Send the Response Command */
     /* Setup the COMMAND-Struct for Response */
     memset(&lpCmdInfo->Response,0,sizeof(COMMAND));
     lpCmdInfo->Response.dwCommand = RES_SAMPLE_TIME;
     lpCmdInfo->Response.dwDataSize = sizeof(SAMPLE_TIME);

     /* Setup Action and State */
     lpCmdInfo->iAction = PLGACT_SEND_RESP;
     lpCmdInfo->iState = 2; /* State after Send-Command */
     break;

   case 2: /* state after send reponse */
     ...

}
```

Figure 148. Sample Code for Distinguishing Between Multiple Requests Within a Plugin

When a request has been received, the VSE Connector Server sets the field *iState* to zero. The *ExecuteHandler* function can set this field to a value representing its actual state. The *iState* field will be passed to the *ExecuteHandler* function unchanged at the next call. The VSE Connector Server continues to call the *ExecuteHandler* function each time the client gets dispatched, until the action PLGACT_FINISH is returned.

Implementing a CleanupHandler Function

The VSE Connector Server calls this function after a request has been executed. It is the counterpart of the *SetupHandler* function. This function should cleanup the handler and may free the handler private-control block (if allocated during the *SetupHandler* function).

The *SetupHandler* function is normally called after a request processing has been completed (that is, the *ExecuteHandler* function has set the action to PLGACT_FINISH). If a networking error occurs (for example the connection is broken), the VSE Connector Server calls the *CleanupHandler* function. The *CleanupHandler* function should always be able to cleanup the handler. It should check the handler's state, and cleanup all resources that have been allocated by the handler.

Creating Your Own Plugin Callback Functions

The plugin callback functions described previously are those that your server plugin must implement, and which are called by the VSE Connector Server. However, you may decide to split your code into additional plugin callback functions. You also can decide to split the code into several modules.

As previously stated, a VSE server plugin is designed to be implemented in LE/VSE-C. If you do not want to implement it in C, you may either:

- Implement the complete plugin in a different programming language (such as PL1 or assembler). However, you must ensure that your code is LE/VSE compliant, and supports the calling conventions used by LE/VSE.
- Use the skeleton as a small stub code, which calls functions implemented in a different programming language. This is the recommended method.

Figure 149 is a chunk of code that illustrates the second method (using the skeleton as a small stub code). In this example, a function called *MyASMExecuteFunction* is called (via the symbol MYASMEXE). The calling convention's Operating System passes the required parameters.

```
/* define the ASM function */
#pragma map(MyASMExecuteFunction,"MYASMEXE")
#pragma linkage(MyASMExecuteFunction,OS)

int ExecuteHandler(void* lpPluginPrivate,
                   void* lpCommandPrivate,
                   CMD_INFO* lpCmdInfo)
{
  int rc;

  rc = MyASMExecuteFunction(lpCmdInfo);
  /* TODO errro checking ... */

  return(PLGERR_NO_ERROR);
};
```

Figure 149. Example of Calling a Stub Code Written in a Language Other Than C

Action Codes Supported by the VSE Connector Server

On return from the *ExecuteHandler* function you have to setup the action code in the field *iAction* of the CMD_INFO block. This tells the VSE Connector Server what to do in the next iteration. Possible action codes are:

PLGACT_NOTHING
No special action, the handler requests to be called again as soon as possible.

PLGACT_SEND
The handler wants to send data to the client. The handler has to setup the following fields in the CMD_INFO block:
lpData
Pointer to buffer
dwDataSize
Size of data to send in bytes

Note: the buffer must be located in the handler's private-control block. It can not be a local variable, because a local variable is only valid within the function. The handler will get control after all data has been sent.

PLGACT_RECEIVE
The handler wants to receive data from the client. The handler has to setup the following fields in the CMD_INFO block:
lpData
Pointer to buffer
dwDataSize
Size of data to receive in bytes

Note: the buffer must be located in the handler's private-control block. It can not be a local variable, because a local variable is only valid within the function. The handler gets control after all data has been received.

PLGACT_SEND_RESP
The handler wants to send the response header to the client. The handler has to setup the field Response in the CMD_INFO block. The handler gets control after the response header has been sent.

PLGACT_FINISH
The handler tells the VSE Connector Server that the request processing is completed. The VSE Connector Server calls the *CleanupHandler* function as a result of this action, and will never call the ExecuteHandler function for this request.

PLGACT_CHECKCANCEL
This action is like the PLGACT_NOTHING, but the VSE Connector Server will check if the cancel byte is available to receive. A request is cancelable if the flag FLG_CANCANCEL is specified in the command header. The VSE Connector Server will set the field bWasCanceled to TRUE if the cancel byte has been received.

PLGACT_WAIT
The handler wants to wait for a ECB (Event Control Block). The handler has to setup the field *lpECB* to a pointer to the ECB. The handler gets control after the ECB has been posted.

PLGACT_WAITRECV
This action is like the PLGACT_WAIT but it also waits for the availability

of data to receive. That is the handler gets control after the ECB has been posted or any data is available to receive. Note: No data is received in this action.

PLGACT_WAITTIMER
The handler wants to wait for a specified amount of time. The handler has to set up the field *iTimer* in the CMD_INFO block to the number of seconds to wait. The handler gets control after the specified amount of time has expired.

Utility Functions Supported by the VSE Connector Server

The VSE Connector Server supports several *utility functions* that can be called from within a server plugin. The plugin:

- Obtains the entry points of the functions in the PLUGIN_INFO control block, which is passed to the *SetupPlugin* function.
- Stores the entry points of the functions it needs in its plugin private-control block.

These are the utility functions that are supported:

StrToAscii
Converts an EBCDIC string to ASCII using the codepages configured in the VSE Connector Server configuration members

StrToEbcdic
Converts an ASCII string to EBCDIC using the codepages configured in the VSE Connector Server configuration members

StrToUpcase
Converts a string to its uppercase equivalent

StrTrim
Removes heading and trailing blanks from a string

StrReplace
Replaces any occurrence of a character in a string with an other character

MatchWildCards
Checks if a string matches a wildcard filter (with '*' and '?')

EncryptData
Encrypts a buffer of data using the random XOR algorithm

Decryptdata
Decrypts a buffer of data using the random XOR algorithm

CheckLibrSecurity
Checks if access to a Libr resource is guaranteed

CheckPOWERSecurity
Checks if access to a POWER resource is guaranteed

CheckVSAMSecurity
Checks if access to a VSAM resource is guaranteed

CheckConsoleSecurity
Checks if access to a Console resource is guaranteed

CheckICCFSecurity
Checks if access to a ICCF resource is guaranteed

Using the IBM-Supplied Server Plugin Example

IESPLGIN.H and IESPLGSK.C are members supplied with the VSE Connector Client, that were used for writing the *server plugin* of the "Date and Time plugin" example. When writing your own server plugins, you can also use the contents of these members:

- IESPLGIN.H is a C-header file used for defining the programming interface of a plugin.
- IESPLGSK.C is a skeleton C-program which is the basis of a plugin. This skeleton contains all the functions you require for writing your own server plugins. It also contains comments which provide you with guidance when writing your own plugins.

Registering and Compiling Your Server Plugin

1. You must register your server plugin in the VSE Connector Server's plugin configuration member (normally IESPLGIN.L). This member consist of one or more textual lines, where each line defines one plugin. The syntax of each line is as follows:

   ```
   PLUGIN=<phase name>,PARM=<any kind of parameters>
   ```

   where you use:

   - Keyword PLUGIN to enter the name of your server plugin.
   - Keyword PARM to pass a parameter list to your server plugin. You can use this parameter list to configure your server plugin. **Note:** You must use the keyword PARM even if you do not pass any parameters to the plugin.
2. You must enter your *phase name* in the LIBDEF chain of the VSE Connector Server startup job (see "Job SKVCSSTJ – Startup Job" on page 30 for details).
3. To compile your server plugin, you should use a job like the one below. Your plugins must be compiled using the re-entrant option (RENT).

   ```
   * $$ JOB JNM=COMPILE,CLASS=4,DISP=L
   * $$ LST DISP=D,CLASS=A,RBS=100
   // JOB COMPILE AND LINK VSE CONNECTOR SEVRER PLUGIN
   // DLBL SYSMSGS,'CVSE.COMP.MSGS',0,VSAM,RECSIZE=3000,RECORDS=35,       X
                 CAT=VSESPUC
   // LIBDEF *,SEARCH=(YOURLIB.YOURSLIB,PRD2.SCEEBASE,PRD2.PROD,          X
                 PRD2.DBASE)
   // LIBDEF PHASE,CATALOG=YOURLIB.YOURSLIB
   // OPTION ERRS,SXREF,SYM,NODECK,CATAL,LISTX
      PHASE YOURNAME,*,SVA
   // EXEC EDCCOMP,SIZE=EDCCOMP,PARM='LONGNAME RENT SS SOURCE             X
                 INFILE(DD:YOURLIB.YOURSLIB(YOURNAME.C))'
   /*
   // EXEC EDCPRLK,SIZE=EDCPRLK,PARM='UPCASE MAP'
   /*
   // EXEC LNKEDT,SIZE=512K
   /&
   * $$ EOJ
   ```
4. Now you can restart the VSE Connector Server, which will write this message to the system console:

   ```
   Fn nnnn LOADING PLUGIN:  <your plugin>
   ```

Implementing a Client Plugin

For each *server plugin* that you implement (described in "Implementing a Server Plugin" on page 288), you must also implement a *client plugin*. The client plugin consists of one JavaBean for each request that your server plugin supports.

These are the requirements for developing your client plugin. You must:

- Have a JDK (Java Development Kit) installed (Version 1.1.6 or later)
- Have the VSE Connector Client installed. As a minimum, the JAR file **VSEConnector.jar** must be entered in your classpath.
- Use the JavaDoc of the VSE Java Beans as a API reference.

The JavaBeans you implement use the communication services provided by the VSE Java Beans. Therefore, your JavaBeans do not have to establish a connection to the VSE Connector Server, since each of the VSE Java Beans that your plugin uses is a subclass of the *VSEResource* class. Your JavaBeans use an instance of the class *VSESystem* to identify the VSE/ESA host.

The setup of the *VSESystem* is the same as for all other VSE Java Beans (for example a *VSELibrarian* Bean). That is, you:

1. Create a *VSEConnectionSpec* and set the necessary properties.
2. Create a *VSESystem* and use the *VSEConnectionSpec* to identify the target VSE/ESA host.
3. Create an instance of your Plugin Bean (when you have a *VSESystem* object).

Here is the outline code for setting up the VSESystem:

```
VSEConnectionSpec spec =  new VSEConnectionSpec(...);
...
VSESystem system = new VSESystem(spec);
...
MyPluginBean myPlugin = new MyPluginBean(system);
```

Using the VSEPlugin class

Any client plugin JavaBean that you write must be a subclass of *VSEPlugin*. *VSEPlugin* is a abstract class, and is itself a subclass of *VSEResource*. As you can also see in the JavaDoc of the VSE Java Beans, *VSEResource* implements the following methods:

setVSESystem
Sets the *VSESystem* to use

getVSESystem
Gets the currently used *VSESystem*

addVSEResourceListener
Adds a *VSEResourceListener* to the bean

removeVSEREsourceListener
Removes a *VSEResourceListener*

notifyListStarted
Notifies all registered *VSEResourceListeners*

notifyListAdded
Notifies all registered *VSEResourceListeners*

notifyListEnded
Notifies all registered *VSEResourceListeners*

setCancel
Cancels the current request

getCancel
Gets the current cancel state

isExistent
Checks if the resource is existent

toString
Returns a string representation

In addition you may also decide to implement some of your own methods, to add properties to your JavaBean.

To activate the request of the client plugin JavaBean, call the method *execute()*. This method is declared as protected, since it should only be called from inside the class. For example you may override the method *refresh* and call *execute* to request a refresh of the JavaBeans properties. The method *execute* will trigger the execution of the request and will return control, after the request has been processed. During request processing, these methods are called:

- *getRequestID*
- *getRequestSubcommand*
- *getRequestFlags*
- *getRequestError*
- *getRequestDataSize*
- *sendRequestdata*
- *getResponseID*
- *getResponseDataSize*
- *receiveResponseData*

For the server plugin and client plugin to be able to work together:

- The Command Ids returned by the methods *getRequestID* and *getResponseID* must match the CommandIDs used in the server plugin.
- The data format that is transferred for a request or response, must match the data format that is supported by the server plugin.
- The plugin JavaBean must use the protocol used for communicating between the plugin server and the plugin client.

The methods *getRequestDataSize* and *getResponseDataSize* should return the size (in bytes) of the data that is transferred:

- If no data is transferred, this method should return zero.
- If the request or response use a DATAEX format, the method should return the constant SIZE_UNKNOWN. This flags the command as a DATAEX command.
- The methods *sendRequestData* and *receiveResponseData* should also return the size of the data that has been sent or received.

In addition:

- For a DATAEX command, the method should retain control until the command has finished. After completion of the data transfer the method should return control to indicate the end of the command.
- For a non-DATAEX command, the return value of the *sendRequestData* and *receiveResponseData* methods should be equal to the value returned by *getRequestDataSize* or *getResponseDataSize*.

Your client plugin can call these methods:
- *getResponseSubCommand*
- *getResponseFlags*
- *getResponseError*

in the method *receiveResponseData()* or after return of control from method *execute()*. By calling these methods, your client plugin can obtain the sub command, flags and error values that were sent in the response.

Your client plugin can then use these values to perform error checking and handling.

General Considerations When Designing Your Plugin

You should consider these areas when designing your plugin:
- How should I define the protocol between the VSE Connector Server and my server plugin?.
- How can I access the resource or application on VSE/ESA?.
- What kind of data should be accessed?.
- Which requests or functions should be allowed?.
- How is the data format to be transferred over the network?.
- How should I structure the view of the data for the client plugin, when defining the JavaBean interface?.

Specifying the Protocol Between VSE Connector Server and Plugin

The Java-based connector uses its *own protocol* for communication purposes. However, you can extend this protocol with your own commands. The characteristics of the protocol used by the Java-based connector is described in this section.

The protocol used by the Java-based connector is based on a TCP/IP connection: it uses a *connection-orientated* stream to transfer data. The (endless) stream is separated into smaller parts, called *stream commands*. A stream command consist of:
- a command header
- a data part (optional)

The various stream commands are identified by a unique 4-byte command-Id, which is stored in the command header. The command header can contain these optional fields:

Field	Description
Flags	Bit-flags (4 bytes). For details, see predefined Flag values
Error	Error code (4 bytes). For details, see predefined error codes
SubCommand	SubCommand ID (4 bytes). For future use only.

The command header is directly followed by the data belonging to the stream command. If the stream command contains no data, the command header for the next stream command follows.

The data belonging to a stream command can be transferred in two ways:

- Fixed-length data
- Variable-length data (DATAEX)

When data is transferred as *fixed length*, the number of data bytes is defined in the command header. After this number of bytes, the next command header starts.

When data is transferred as *variable length* (the "DATAEX" method), the length of the data is not directly specified, but is instead specified implicitly within the data. When developing your plugin, you can therefore define any kind of indicator within the transferred data, that marks the end of the data. The sender and the receiver of the data must both be able to recognize this indicator. After the end of the variable length data has been reached, the next command header follows.

When developing your plugins, you can extend the existing set of requests with your own requests. To do so, you must define:

- Command Ids for your requests and responses.
- The format of the data that belongs to the commands.

Choosing the Access Method to the Data / Application

Your must first choose the access method on VSE/ESA to be used for accessing the data or application for which you require a plugin. Since the VSE Connector Server runs in a *batch* partition (static or dynamic), your plugin will also be executed in the same batch environment. As a result, your access method must be able to access the data or application from within a *batch* program.

Your plugin must be written within the LE/VSE environment. A VSE Connector Server plugin should be implemented in LE/VSE C, since the VSE Connector Server is itself implemented in LE/VSE C. You can, however, call Assembler or PL/1 modules from within your plugin.

You also can use VSE macros (such as GETVIS) when writing your plugin, but you must program the use of the LE/VSE register yourself.

Considerations for ASCII / EBCDIC and Big / Little Endian

You can provide access to any kind of data. If you decide to access textual data, you must use the utility functions to convert between ASCII and EBCDIC, before transferring the data over the network. Your plugin is responsible for carrying out the required conversions.

You can perform the conversion between ASCII and EBCDIC on the client side, but you are recommended to perform this conversion on the server side. The utility function to convert form ASCII to EBCDIC and vice versa uses the configured code pages to do the translation. If you decide to transfer binary data like integer values, you do not usually have to be concerned about big or little endian formats.

VSE/ESA (that is the S/390 platform) uses the big endian format for integer values. Java also uses the big endian format even if it is running on an Intel (little endian) platform. Therefore you do not have to convert integer values before transferring them.

Deciding Which Requests / Functions Should Be Supported

You have to decide which kind of access requests or functions your plugin should support. A VSE Connector Server plugin can support one ore more different requests.

You can decide to use one request for each kind of access (read, write, update, ...), or you can decide to use only one request which is able to execute different kinds of access. In the latter case, you might define a parameter which informs the plugin when a write or read operation is requested.

Transferring Data Over the Network

You must also define the interface between the server-part and the client-part of the plugin. That is, you must define the *protocol* that is used to transfer the data. A well-defined protocol is used for communicating between the VSE Connector Client and VSE Connector Server. Therefore, your plugin's protocol must also support this protocol.

The protocol used for communicating between the VSE Connector Client and VSE Connector Server consists of *requests*. Each request is:

- Independent from other requests.
- Transferred as a block over the network.
- Identified by a 4-byte number, the *command-Id*.

Your plugin can use its own command-Ids that must be within a specific range. Although your plugin must use the predefined protocol to transfer its data, the format of the data itself can be defined by yourself.

Structuring the Client Plugin's View

You must also consider how the client-part view of the data or application, should be structured. The Java-based connector allows you to implement your own JavaBeans plugins which use the communication methods provided by the Java-based connector.

For each request you have to implement a corresponding JavaBean on the client side. This JavaBean is responsible for sending and receiving the data belonging to the request. Therefore, this JavaBean must be familiar with the format of the transferred data.

You can design the external interfaces of the plugin JavaBeans that you develop.

Chapter 24. Using the DB2-Based Connector to Access Data

This chapter describes how you use the DB2-based connector to access:
- VSAM datasets
- DL/I databases

The DB2-based connector uses these products:

- On the VSE/ESA host:
  - DB2 Server for VSE
  - The DB2 Stored Procedures facility (available with Version 6 or later of the DB2 Server for VSE).
  - The VSAMSQL CLI
  - DL/I VSE
- On the middle-tier:
  - DB2 Connect
  - ODBC, JDBC, or CLI

For a general introduction to the DB2-based connector, see "Overview of the DB2-Based Connector" on page 6.

This chapter contains these main sections:

- "How You Use DB2 Stored Procedures" on page 310 (which includes a description of Stored Procedure Servers)
- "Using DB2 Stored Procedures to Access VSAM Data" on page 312
- "Using DB2 Stored Procedures to Access DL/I Data" on page 320

© Copyright IBM Corp. 2000, 2003

How You Use DB2 Stored Procedures

A DB2 Stored Procedure is a *program* that you write yourself, and then compile and catalog into a library, and then define to DB2. Your DB2 Stored Procedures can then be called from:

- A Web Client
- The middle-tier of a 3-tier VSE/ESA environment
- A local batch program.

You can write a DB2 Stored Procedure in any LE (Language Environment)-compliant language. The API (Application Programming Interface) within a DB2 Stored Procedure differs depending upon whether VSAM or DL/I data is being accessed on the VSE/ESA host.

These are the main advantages of using DB2 Stored Procedures:

- When the database manager is running in multiple user mode, local applications or remote DRDA applications can invoke a DB2 Stored Procedure. Since the SQL statements issued by a DB2 Stored Procedure are local to the VSE/ESA host, they do not incur the high network costs of distributed statements. Instead, single network *send* and *receive* operations can be used for processing all the statements contained in a DB2 Stored Procedure.
- You can use a DB2 Stored Procedure to hide the details of the database design from application programs running on the Web Client or middle-tier.
- If a database is modified, only the DB2 Stored Procedure (and not the application programs) needs to be modified.
- You can use a DB2 Stored Procedure to hide sensitive data from specific application programs.
- You can encapsulate business logic at the VSE/ESA host, instead of having to include this business logic in numerous application programs.
- It is easier to maintain an environment in which DB2 Stored Procedure applications are maintained at the VSE/ESA host, instead of being distributed across a number of Web Clients or middle-tiers.

Grouping Stored Procedure Servers

The *Stored Procedure Server* is contained in a separate static or dynamic partition, and must be dedicated to a single DB2 Server for VSE. The DB2 Stored Procedures described in the previous section run under the control of a Stored Procedure Server, which is LE-compliant.

By grouping your Stored Procedure Servers, you can distribute the database workload over multiple partitions. This might be useful if certain DB2 Stored Procedures must always have a DB2 Server for VSE available. In this case, a Stored Procedure Server group could be dedicated to the DB2 Stored Procedure. Other DB2 Stored Procedures could then share other Stored Procedure Server groups:

- Some DB2 Stored Procedures might have special requirements (for example, they require unusually large amounts of virtual storage).
- Other DB2 Stored Procedures might have to access resources that are not required by the other DB2 Stored Procedures.

The ability to group Stored Procedure Servers provides the database administrator with flexibility when defining the environment, and is also useful for system-tuning purposes.

Programming Requirements When Using DB2 Stored Procedures

On the Web Client or middle-tier: you require JDBC or ODBC/CLI (which do not require a SQL precompiler). If however you use another programming language that contains embedded SQL to call a DB2 Stored Procedure, you *do require* an SQL precompiler.

On the VSE/ESA host: providing the DB2 Stored Procedure does not contain SQL-specific calls, you do not require an SQL precompiler.

The VSAMSQL CLI consists of a small object module that you must link to each of your DB2 Stored Procedures that will use the VSAMSQL CLI. This object module is a "stub" that allows your DB2 Stored Procedures to use the CLI (Call Level Interface) to access VSE/VSAM data *as if the data were relational*.

A DB2 Stored Procedure must be LE-compliant. Language Environment is the prerequisite run-time environment for applications generated using compilers that run with VSE/ESA.

These are the most important interfaces that you can use from your application programs:

- ODBC (Open DataBase Connectivity) on your middle-tier platform
- CLI (Call Level Interface) in your DB2 Stored Procedure running on the VSE/ESA host

Using DB2 Stored Procedures to Access VSAM Data

This section describes how you can use DB2 Stored Procedures to access VSE/VSAM data. Although Figure 150 on page 313 shows the access to VSAM data only, you can access DB2 and DL/I data using the same DB2 Stored Procedure.

To access the mapped VSAM data, your application programs use a VSAMSQL Call Level Interface (CLI), which is based upon the IBM DB2 Call Level interface. Using the VSAMSQL CLI, your application programs can issue SQL-like calls to VSAM data from within a DB2 Stored Procedure, as described in the following sections:

- "Overview: Accessing VSAM Data via DB2 Stored Procedures" on page 313 provides an overview of how DB2 Stored Procedures are used to display mapped VSAM data via the VSAMSQL CLI.
- "Using the Call Level Interface: Activities on the Requestor" on page 314 describes the activities you must perform *on the client* in order to use a DB2 Stored Procedure to display mapped VSAM data via the VSAMSQL CLI.
- "Using Call Level Interface: Activities on the VSE/ESA host" on page 315 describes the activities you must perform *on the VSE/ESA host* in order to use a DB2 Stored Procedure to display mapped VSAM data via the VSAMSQL CLI.
- "Program Flow When Using the VSAMSQL Call Level Interface" on page 317 describes the typical program flow when a DB2 Stored Procedure performs a VSAM-CLI update on mapped VSAM data.
- "SQL Statements Supported by VSAMSQL Call Level Interface" on page 318 lists the SQL statements that you can use when a DB2 Stored Procedure performs a VSAM-CLI update on mapped VSAM data.

Note: For a practical example of how an application program uses the VSAMSQL CLI, see "Step 2. Initialize the VSAMSQL CLI Environment" on page 226.

Overview: Accessing VSAM Data via DB2 Stored Procedures

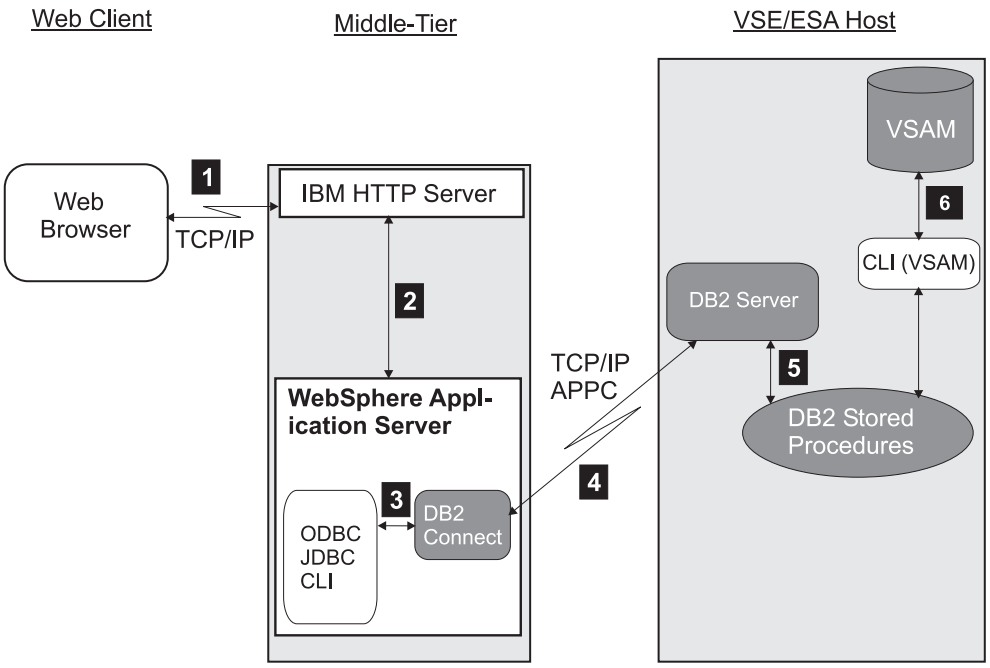


Figure 150. How You Use DB2 Stored Procedures To Access VSAM Data

The number of each list item below describes a step shown in Figure 150:

1. The clients's Web browser requests an HTML page from the IBM HTTP Server running on the middle-tier.
2. The IBM HTTP Server calls the WebSphere Application Server for requests contained in the HTML page (for example, requests for an applet or servlet).
3. The interfaces (ODBC, JDBC, or CLI) communicate with the DB2 Server for VSE, via DB2 Connect.
4. DB2 Connect communicates with the DB2 Server for VSE, via DRDA (Distributed Relational Database Architecture). The underlying protocol used here can be either APPC or TCP/IP.
5. DB2 Server for VSE manages the execution of a DB2 Stored Procedure, using the Stored Procedure Server.
6. The DB2 Stored Procedure can now access the VSAM data stored on the VSE/ESA host, via VSAMSQL CLI. The DB2 Stored Procedure contains calls to the VSAM *CLI functions* (described in Table 8 on page 315), which are used to perform the access.

The process is now the reverse of steps 1 to 6. The DB2 Server for VSE passes the results back to the requester.

As shown in Figure 150, to use a DB2 Stored Procedure to access VSAM data stored on the VSE/ESA host, you require:

- *On the middle-tier:*
  - A Web Server (such as the IBM HTTP Server)
  - The WebSphere Application Server
  - DB2 Connect

  - ODBC or CLI
- *On the VSE/ESA host:*
  - The DB2 Server for VSE
  - The VSAMSQL CLI

These are the general steps you should follow to develop application programs that access mapped VSAM data via DB2 Stored Procedures:

1. Establish DB2 connection between the requestor and VSE/ESA host.
2. Design your application program and the VSAM cluster to be used for storing your maps.
3. Define the maps and views for the VSAM cluster (see Chapter 15, "Mapping VSE/VSAM Data to a Relational Structure", on page 129).
4. Write your DB2 Stored Procedure that includes the logic and requests to VSAM and DB2 (refer to the IBM publications *DB2 Server for VSE, Database Administration*, SC092-3890 and *DB2 Server for VSE, Application Programming*, SC092-3930).
5. Create an entry in DB2 for your DB2 Stored Procedure.
6. Write the requestor calls to the DB2 Stored Procedure.
7. Test and run your application program.

Using the Call Level Interface: Activities on the Requestor

On the requestor, you can invoke DB2 Stored Procedures via Stored-Procedure calls that are implemented in relational database interfaces such as JDBC or ODBC/CLI. Here is a summary of the stored-procedure interfaces for these relational databases:

JDBC

```
String sql="Call <proc_name> (?,?,?,?,?,?)";
statement = con.prepareCall (sql);
statement.execute ();
```

ODBC/CLI

```
CALL procedure_name (?,?,?,?,...)
SQLExecDirect() or
SQLPrepare() followed by SQLExecute()
```

Embedded SQL

```
CALL proc_name [(parm1[:parmind1],...,parmn[:parmindn])] or
CALL proc_name USING DESCRIPTOR :sqlda
```

For a detailed description of the above interfaces, refer to these documents that describe how stored procedures are called:

- Microsoft ODBC SDK Programmer's Reference
- IBM DB2 Universal Database Call Level Interface Guide and Reference (S10J-8159)
- IBM Embedded SQL Programming Guide (S10J-8158)
- The JDBC Data Access API, at these Web sites:

  http://java.sun.com/products/jdbc/index.html
  http://java.sun.com/products/jdk/1.1/docs/guide/jdbc/index.html

Using Call Level Interface: Activities on the VSE/ESA host

On the VSE/ESA host, a VSAMSQL Call Level Interface (CLI) provides you with C-program functions for accessing mapped VSAM data using a DB2 Stored Procedure. This interface accesses a VSAM file in the same way as a table is used in a relational database. Therefore, you must define a *relational* view for each VSAM cluster whose mapped data you wish to access using a DB2 Stored Procedure via the VSAMSQL interface.

All VSAMSQL CLI C-program functions have a prefix VSAMSQL, and have the same syntax and functionality as DB2 CLI functions (where DB2 CLI functions have a prefix SQL).

Note: To use the CLI functions, you have to include the C header file *IESVSQL.h* in your program source. The object file *IESVSQLO.Obj* must be linked to your application. Both *IESVSQL.h* and *IESVSQLO.Obj* are located in VSE library PRD1.BASE.

These CLI functions are supported:

Table 8. CLI Functions You Can Use for Accessing Mapped VSAM Data

CLI Function	Description
VSAMSQLAllocConnect *	Allocate Connection Handle
VSAMSQLAllocEnv *	Allocate Environment Handle
VSAMSQLAllocHandle	Allocates Environment, Connection, Statement, or Descriptor Handles
VSAMSQLAllocStmt *	Allocate a Statement Handle
VSAMSQLBindCol	Bind a Column to an Application Variable
VSAMSQLBindParameter	Bind A Parameter Marker to a Buffer
VSAMSQLCloseTable	Close a specified table (Cluster)
VSAMSQLColAttribute *	Return a Column Attribute
VSAMSQLColAttributes	Get Column Attributes
VSAMSQLColumns	Get Column Information for a Table
VSAMSQLDescribeCol	Return a Set of Attributes for a Column
VSAMSQLError	Retrieve Error Information
VSAMSQLExecDirect	Execute a Statement Directly
VSAMSQLExecute	Execute a Statement
VSAMSQLFetch	Fetch Next Row
VSAMSQLFreeConnect *	Free Connection Handle
VSAMSQLFreeEnv *	Free Environment Handle
VSAMSQLFreeHandle	Free Handle Resources
VSAMSQLFreeStmt *	Free (or Reset) a Statement Handle
VSAMSQLGetDiagRec	Get multiple fields settings of Diagnostic Record
VSAMSQLNumParams	Get Number of Parameters in A SQL Statement
VSAMSQLNumResultCols	Get Number of Result Columns
VSAMSQLPrepare	Prepare a Statement
VSAMSQLPrimaryKeys	Get Primary Key Columns of A Table
VSAMSQLRowCount	Get Row Count

Table 8. CLI Functions You Can Use for Accessing Mapped VSAM Data (continued)

CLI Function	Description
VSAMSQLSetParam *	Bind A Parameter Marker to a Buffer
VSAMSQLTables	Get Table Information

Note: The functions marked with an asterisk (*) have been included in more recent functions. For details, refer to the IBM publication *DB2 Universal Database Call Level Interface, Guide and Reference*, S10J-1859.

Example of the Syntax of a CLI Function – VSAMSQLCloseTable

Here is an example of how the CLI function *VSAMSQLCloseTable* is described in the online documentation.

VSAMSQLCloseTable - Close a specified table (Cluster)

Purpose

VSAMSQLCloseTable() closes the specified VSAM clusters.

Syntax

```
VSAMSQLRETURN   VSAMSQLCloseTable (VSAMSQLHENV       henv,
                                   VSAMSQLCHAR*      szTableName,
                                   VSAMSQLSMALLINT   cbTableName);
```

Function Arguments

VSAMSQLCloseTable Arguments

Data Type	Argument	Use	Description
VSAMSQLHENV	henv	input	Environment handle
VSAMSQLCHAR*	szTableName	input	Table name to close
VSAMSQLSMALLINT	cbTableName	input	Length of the table name or VSAMSQL_NTS

Return Codes

```
VSAMSQL_SUCCESS
VSAMSQL_ERROR
VSAMSQL_INVALID_HANDLE
```

Diagnostics

VSAMSQLCloseTable SQLSTATEs

SQLSTATE	Description	Explanation
HY009	Invalid argument value	The argument was invalid

Program Flow When Using the VSAMSQL Call Level Interface

This is the typical flow of program statements, when the VSAMSQL CLI is used for mapped VSAM data:

```
VSAMSQLRETURN  rc;         // Return Code
VSAMSQLHENV    hEnv;       // Environment Handle
VSAMSQLHDBC    hDBC;       // Connection Handle
VSAMSQLHSTMT   hStmt;      // statement Handle

//allocate Environment
rc = VSAMSQLAllocHandle(VSAMSQL_HANDLE_ENV,VSAMSQL_NULL_HANDLE,&hEnv);
//allocate Connection
rc = VSAMSQLAllocHandle(VSAMSQL_HANDLE_DBC,hEnv,&hDBC);
//allocate Statement
rc = VSAMSQLAllocHandle(VSAMSQL_HANDLE_STMT,hDBC,&hStmt);

// Prepare a Statement
rc = VSAMSQLPrepare(hStmt,
     "UPDATE VSESP.USER.CATALOG/VSAM.DISPLAY.DEMO.CLUSTER/MAP1 SET EMAIL=?, AGE=?"
     "WHERE NAME=?",VSAMSQL_NTS);

// Query the number of Parameters
rc = VSAMSQLNumParams(hStmt,&Num);

// Bind local Variables/Values to the Statement
rc = VSAMSQLBindParameter(hStmt,1,VSAMSQL_PARAM_INPUT,
     VSAMSQL_C_CHAR,VSAMSQL_VARCHAR,5,0,"Hugo",5,NULL);
 ...

// Execute the Statement
rc = VSAMSQLExecute(hStmt);
 ...
rc = VSAMSQLFreeHandle(VSAMSQL_HANDLE_STMT,hStmt);
rc = VSAMSQLFreeHandle(VSAMSQL_HANDLE_DBC,hDBC);
rc = VSAMSQLFreeHandle(VSAMSQL_HANDLE_ENV,hEnv);
```

Figure 151. Typical Program Flow When Performing a VSAMSQL CLI Update

SQL Statements Supported by VSAMSQL Call Level Interface

These SQL statements are supported when you use the VSAMSQL CLI on mapped VSAM data:

```
INSERT INTO table
       (col1, col2, ...)
       VALUES (val1, val2, ...),
              (val3, val4, ...),
              ...

UPDATE table
       SET col1=val2,
           col2=val2,
           ...
       WHERE col3=val3 AND
             col4=val4 AND
             ...

DELETE FROM table
       WHERE col3=val3 AND
             col4=val4 AND ...

SELECT col1, col2, ...
       FROM table
       WHERE col3=val3 AND
             col4=val4 AND
             ...
```

Figure 152. SQL Statements Supported by VSAMSQL Call Level Interface (CLI)

The **WHERE** statement supports these compare operations:

```
= (equal to)
< (less than)
> (greater than)
<= (less or equal)
>= (greater or equal)
<> (not equal)
```

Notes:

1. You can combine multiple filters using **AND**.
2. **OR** is not supported.

You specify the table using:
- VSAM Catalog
- Cluster File-ID
- map name
- view name (optional)

Here is the syntax of this statement:

```
 MY.USER.CATALOG/MY.VSAM.CLUSTER/MYMAP1
or
 MY.USER.CATALOG/MY.VSAM.CLUSTER/MYMAP1/MYVIEW1
```

The SQL statement can contain placeholders ('**?**') for parameters. The parameters must be bound before execution of the statement. The parameters are numbered from the beginning of the statement, starting at one.

You can use placeholders for these parts:
- table name
- column names

- values

To get the result set of a select statement, you can bind local variables to the result set columns. The columns of the result set are either:

- Those which have been specified in the select statement:

  ```
  ("SELECT col1,col2,... FROM...")
  ```

- All columns of the map/view:

  ```
  ("SELECT * FROM ...")
  ```

For RRDS Clusters, a special column named **RELRECNO** is used to specify the relative record number of the record (=key), which is not part of the record itself. A **SELECT * FROM** statement automatically adds the **RELRECNO** column as the last column to the result set. You can specify the **RELRECNO** column as a filter in all supported SQL statements **("UPDATE table SET col1=val1 WHERE RELRECNO=5 ...")**.

Using DB2 Stored Procedures to Access DL/I Data

This section describes how you can use DB2 Stored Procedures to access DL/I data. Although Figure 153 shows the access to DL/I data only, you can access VSE/VSAM and DB2 data using the same DB2 Stored Procedure.

To access DL/I data from within a DB2 Stored Procedure, your program can use the AIBTDLI interface, which accesses DL/I data *directly* using DL/I methods. You do *not* map DL/I data (as you do for VSE/VSAM data).

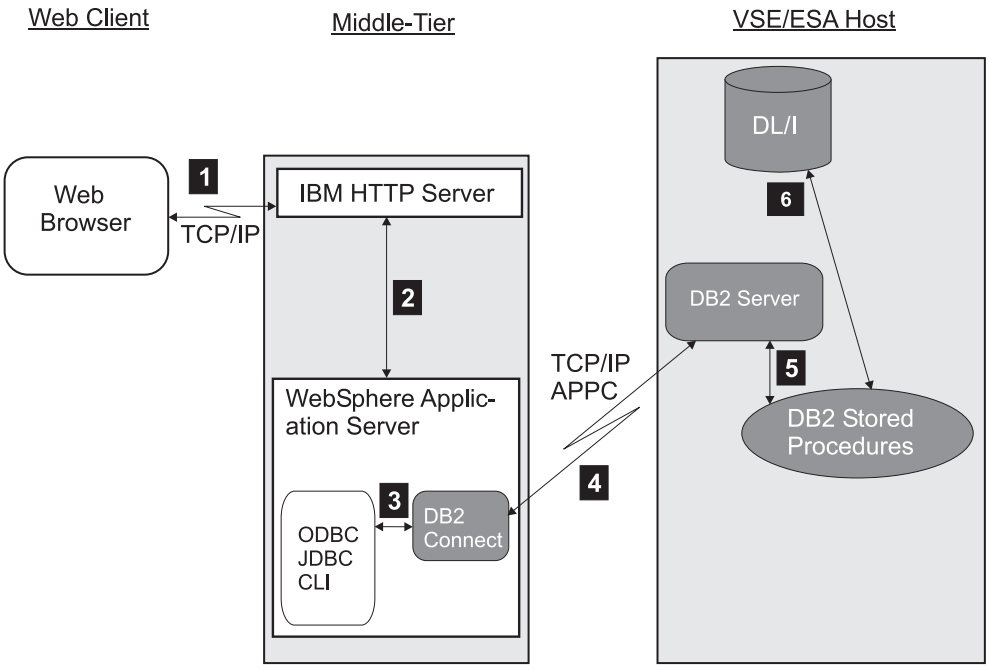


Figure 153. How You Use DB2 Stored Procedures To Access DL/I Data

The number of each list item below describes a step shown in Figure 153:

1. The clients's Web browser requests an HTML page from the IBM HTTP Server running on the middle-tier.
2. The IBM HTTP Server calls the WebSphere Application Server for requests contained in the HTML page (for example, requests for an applet or servlet).
3. The interfaces (ODBC, JDBC, or CLI) communicate with the DB2 Server for VSE, via DB2 Connect.
4. DB2 Connect communicates with the DB2 Server for VSE, via DRDA (Distributed Relational Database Architecture). The underlying protocol used here, can be either APPC or TCP/IP.
5. DB2 Server for VSE manages the execution of a DB2 Stored Procedure, using the Stored Procedure Server.
6. The DB2 Stored Procedure can now access the DL/I data stored on the VSE/ESA host, via the interface AIBTDLI (see "Overview of the AIBTDLI Interface" on page 321 for details).

The process is now the reverse of steps 1 to 6. The DB2 Server for VSE passes the results back to the requester.

As shown in Figure 153 on page 320, to use a DB2 Stored Procedure to access DL/I data stored on the VSE/ESA host, you require:

- *On the middle-tier:*
  - A Web Server (such as the IBM HTTP Server)
  - The WebSphere Application Server
  - DB2 Connect
  - ODBC, JDBC, or CLI
- *On the VSE/ESA host:*
  - The DB2 Server for VSE
  - DL/I VSE

These are the general steps you should follow to develop application programs that access DL/I data via DB2 Stored Procedures:

1. Establish DB2 connection between the requestor and VSE/ESA host.
2. Design your application program.
3. Write your DB2 Stored Procedure that includes the logic and requests to DL/I and VSAM or DB2 (refer to the IBM publications *DB2 Server for VSE, Database Administration*, SC092-3890 and *DB2 Server for VSE, Application Programming*, SC092-3930).
4. Create an entry in DB2 for your DB2 Stored Procedure.
5. Write the requestor calls to the DB2 Stored Procedure.
6. Test and run your application program.

Overview of the AIBTDLI Interface

The *AIBTDLI* Interface allows VSE batch programs, such as DB2 Stored Procedures, to issue DL/I calls without a DL/I batch environment having been established using DLZRRC00 or DLZMPI00.

Note: For a list of the installation requirements for using the AIBTDLI interface, see "Step 8: Customize the DB2-Based Connector for DL/I Data Access" on page 83.

AIBTDLI passes the DL/I calls to *DLZMPX00*, which connects to a running CICS/DLI online system in the form of an MPS batch task. DLZMPX00 associates each VSE batch task that uses AIBTDLI, with a DLZBPC00 mirror task on the CICS/DLI online-side, which runs on its behalf in the known MPS scheme.

The databases reside (are "opened") at the CICS/DLI online side. All DL/I calls passed to, and results and feedback information returned from, the CICS/DLI online system are routed through DLZMPX00 via a unique XPCC connection between the VSE batch and the CICS/DLI BPC mirror task. Database access contention, resource logging, and recovery are performed on the CICS/DLI online system using existing CICS/DLI functions.

Figure 154 on page 322 illustrates the partition layout and processing flow of the three existing DL/I environments (batch, MPS batch and CICS/DLI online), and compares them to an environment where AIBTDLI is used:

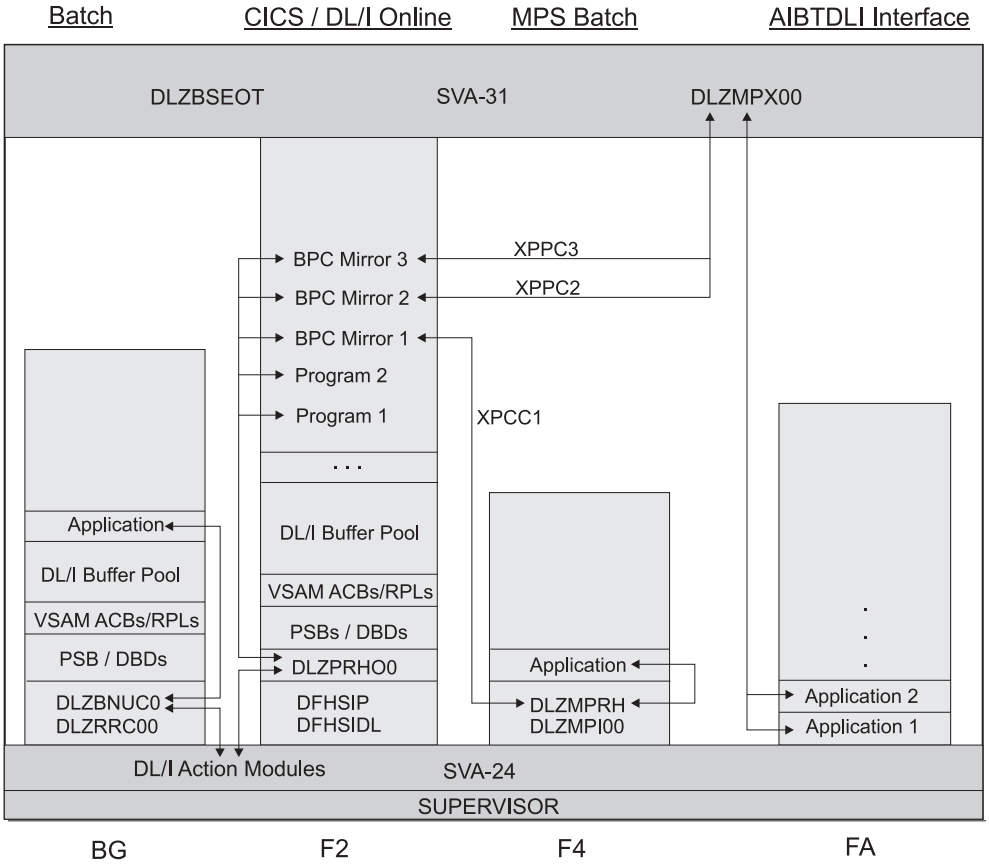


Figure 154. DL/I Partition Layout for Batch, MPS Batch, CICS/DLI Online, and AIBTDLI Interface

BG A DL/I batch environment is initialized by calling the DL/I batch root phase *DLZRRC00*. The application is loaded by DLZRRC00 and communicates with DL/I via the DL/I batch program request handler *DLZBNUC0*. All DL/I resources reside in the DL/I batch partition.

F2 A CICS/DLI online environment is established by calling the CICS root phase *DFHSIP*, which calls DL/I module *DFHSIDL* for the DL/I initialization. The online programs communicate with DL/I via the DL/I online program request handler *DLZPRHO0*, which resides in the DL/I online nucleus *DLZNUCxx*. All DL/I resources reside in the CICS/DLI partition.

F4 An MPS batch environment is initialized by calling the DL/I MPS batch root phase *DLZMPI00*. The application is loaded by DLZMPI00 and communicates with DL/I via the MPS batch program request handler *DLZMPRH*. All DL/I resources reside in the CICS/DLI online partition.

Using an XPCC connection, DLZMPRH passes DL/I calls to, and receives data from, a CICS/DLI BPC mirror task which runs on its behalf in the CICS/DLI online system.

FA In an environment where the *AIBTDLI* interface is used, a DL/I batch application can be started as the main program, which communicates with DL/I via *DLZMPX00*. All DL/I resources reside in the CICS/DLI online partition.

Using an XPCC connection, DLZMPX00 passes DL/I calls to, and receives data from, a CICS/DLI BPC mirror task which runs on its behalf in the

CICS/DLI online system. The processing flow is similar to the MPS batch environment. The difference is that the initialization of a DL/I batch system is *not* required, and multiple batch programs (tasks) can use DLZMPX00 at the same time.

Creating Programs That Use AIBTDLI

The AIBTDLI interface is used by VSE batch programs. Your programs can be started as the main task, or attached as a subtask. Up to 128 (sub)tasks can use AIBTDLI at the same time.

Programs using AIBTDLI can be written in COBOL for VSE, PL/I for VSE, or Assembler. They can take any amode/rmode characteristics. All types of existing DL/I call functions are supported:

- PCB
- GET-type
- ISRT
- REPL
- DLET
- CHKP
- TERM

In addition, two new calls are provided which allow database changes to be backed out, providing they have not been committed by a DL/I CHKP or TERM call, or implicitly by task termination:

ROLB The Roll Back call is used to dynamically back out database changes up to the last syncpoint, and return control to the program.

ROLL The Roll call is used to dynamically back out database changes up to the last syncpoint, and abnormally terminate (CANCEL) the program.

The ROLB and ROLL calls are coded like a DL/I TERM call. No further parameters are required.

For a description of the general call format of the AIBTDLI interface, refer to "Invoking the AIBTDLI Interface" on page 324.

From a DL/I view, applications using AIBTDLI are implemented in the same way as CICS/DLI online programs that use the CALL interface. To access a DL/I database, your applications:

- Must start with a DL/I scheduling call, which builds up a connection between the user program (task) and the CICS/DLI MPS system and schedules the PSB. You can serially schedule more than one PSB. Before scheduling a new PSB, you must release the previous PSB using a DL/I termination call.
- Can issue any type of DL/I database calls.
- Should end with a DL/I termination call, which ends the connection between the user program (task) and the CICS/DLI MPS system, releases the PSB, frees all DL/I resources owned by this task, and provides for commit and syncpoint processing.

In order to check DL/I response information as returned in the AIB (described in "Return and Status Codes" on page 326) you must include the copybook DLIAIB.

You should be aware of the following differences to normal DL/I batch applications:

COBOL and PL/I programs

- The ENTRY (COBOL) or PROCEDURE (PL/I) statement no longer requires a reference to the PCB addresses.
- The pointers to the PCBs are instead obtained through a scheduling call in the same way as in a CICS/DLI online program.

PL/I programs

- The AIBTDLI interface has to be declared as an assembler language entry point:

  ```
  DCL AIBTDLI ENTRY OPTIONS(ASM);
  ```

- PSBs should not be generated with LANG=PLI.

The sample programs DLZHLA80, DLZAIC50 and DLZAIP50 show how you can write a program that uses the AIBTDLI interface, when the Assembler, COBOL or PL/I language is used, respectively.

Invoking the AIBTDLI Interface

You can use the AIBTDLI interface in COBOL, PL/I, and assembler programs. The format and parameters of this interface are equivalent to the respective CBLTDLI, PLITDLI, or ASMTDLI interface, as described in *DL/I CALL and RQDLI Interfaces*. The following description shows the general format of the types of DL/I calls, and explains the scheduling call's new parameters.

Note: For assembler programs:
- You must provide an 18-fullword register save area in register 13.
- Register 1 must point to the parameter list.

Format of the Scheduling Call

The general format of the scheduling call is for:

COBOL: CALL 'AIBTDLI' USING [parm-count,] 'PCB ',psbname, **aibparm[,destination]**.

PL/I: CALL AIBTDLI (parm-count,'PCB ',psbname, **aibparm[,destination])**;

ASSEMBLER: CALL AIBTDLI

Here is a description of the new parameters. Existing parameters retain their previous functionality.

aibparm

is the name of a fullword to which DL/I returns the address of the *Application Interface Block (AIB)*. Use of this parameter is mandatory. The AIB is a new control block used to pass to the user the address of the PCB list and the maximum length of the I/O area following a scheduling call, the return code after each DL/I call, and pointers to a message area and a partition list in the event of an error. Details on the format and usage of the AIB are shown in "Format of the AIB – User Section" on page 326. The AIB is equivalent to the existing online UIB control block, as documented in *DL/I CALL and RQDLI Interfaces*.

After a successful scheduling call, the field *AIBPCBAL* contains the address of the PCB list.

destination

denotes the target system, where the scheduling (and all subsequent) calls should be processed. As described above, DL/I calls entered via AIBTDLI are routed to an active CICS/DLI MPS system. When MPS has been activated in more than one CICS/DLI partition at the same time, the destination specification allows the selection of the MPS subsystem to which the DL/I calls should be directed.

The destination specification can be entered using one or both of the following two parameters:

PARTID=xx

... xx denotes the MPS partition, where the DL/I calls should be processed

APPLID=yyyyyyyy

... yyyyyyyy denotes the CICS (generic) applid, where the DL/I calls should be processed

A destination specification is not required, if only one MPS system is running. For more information on how to select the correct target system, refer to "Scheduling with Single and Multiple MPS Systems" on page 328.

Format of the Database Call

The general format of the database call is for:

COBOL: CALL 'AIBTDLI' USING [parm-count,]call-function, db-pcb-name,i/o-area[,ssa...].

PL/I: CALL AIBTDLI (parm-count,call-function,db-pcb-name, i/o-area[,ssa...]);

ASSEMBLER: CALL AIBTDLI

Existing parameters retain their previous functionality.

Format of the Termination Call

The general format of the termination call is for:

COBOL: CALL 'AIBTDLI' USING [parm-count,]'TERM'.

PL/I: CALL AIBTDLI (parm-count,'TERM');

ASSEMBLER: CALL AIBTDLI

The function code 'TERM' can be abbreviated to 'T '.

Format of the Roll Back Call

The general format of the Roll Back call is for:

COBOL: CALL 'AIBTDLI' USING [parm-count,]'ROLx'.

PL/I: CALL AIBTDLI (parm-count,'ROLx');

ASSEMBLER: CALL AIBTDLI

Compiling and Link-Editing Your Programs

The compile and link requirements for DL/I programs implemented in COBOL, PL/I, and Assembler, are described in the *DL/I 1.11 Release Guide*. However, for PL/I programs, there are certain differences between the information provided in the *DL/I 1.11 Release Guide* and what you must specify in order to use the AIBTDLI interface:

- Your programs should *not* be compiled with * PROCESS SYSTEM(DLI);
- You should exclude these two linkage-editor statements:

```
INCLUDE IBMRPJRA
ENTRY PLICALLB
```

The sample programs DLZHLA80, DLZAIC50 and DLZAIP50 illustrate how you can compile and linkedit a DL/I program using the AIBTDLI interface, in each of the languages COBOL, PL/I, and Assembler.

Return and Status Codes

For all types of calls, DL/I returns response and error information in the new *Application Interface Block (DLIAIB)*, which you use in the same way as the previous online User Interface Block (DLIUIB). For DL/I database calls, status codes are passed back using the PCB (as was done previously).

After each DL/I call, your checking of the information returned should start with a check of the AIB. If the AIB return code does not suggest that an error has occurred, you can then examine the PCB status code.

Format of the AIB – User Section

```
DLZAIB   DSECT
AIB      DS    0F          START OF DSECT
AIBPCBAL DS    A           PCB ADDRESS LIST
AIBRCODE DS    0XL2        DL/I RETURN CODES
AIBFCTR  DS    X           RETURN CODE
AIBDLTR  DS    X           ADDITIONAL INFORMATION
         DS    2X          RESERVED
AIBMSGPT DS    A           POINTER TO ERROR MSG AREA
AIBPLPT  DS    A           POINTER TO LIST OF PARTIDS
AIBIOLM  DS    F           MAX. LENGTH OF IOAREA
         DS    2F          RESERVED
AIBLEN   EQU   *-AIB       LENGTH OF USER AIB
```

The AIB return code is passed back in the two-byte field *AIBRCODE*, which can take the same values as UIBRCODE in a CICS/DLI online program. For a list of all possible return codes of UIBRCODE (=AIBRCODE), refer to *DL/I Messages and Codes* and the *DL/I 1.11 Release Guide*.

In addition to the existing return codes mentioned above, the new codes *X'080A'* and *X'FF00'* have been introduced. For details about these codes, see "AIBTDLI DL/I Messages and Return Codes", on page 365. For code *X'FF00'*, also see "How Return Code X'FF00' Is Used" on page 327 (below).

After having inspected the feedback information in the AIB, the DL/I PCB status code should be checked. A list of all PCB status codes can be found in *DL/I Messages and Codes*. No new PCB status codes have been introduced.

The user section of the AIB will be delivered as member DLIAIB in an Assembler, COBOL and PL/I version.

How Return Code X'FF00' Is Used

When X'FF00' is returned in AIBRCODE, the AIBTDLI interface has detected an unrecoverable error. At such an event it ends the connection between the user task and the CICS/DLI MPS system, releases the PSB and frees all DL/I resources acquired by this task. Error handling for an X'FF00' situation internally triggers a DL/I termination call, which provides backout and syncpoint processing for the PSB which has been scheduled.

The reason for the X'FF00' situation is explained in a message, which DL/I writes to the console. The address field *AIBMSGPT* points to a storage area containing the message, which is formatted as follows:

- A 2-byte LL field. LL is the length of the message without the length of the LLBB field.
- A 2-byte BB field, set to binary zero.
- A variable length field containing the text of the message.

Errors That Do Not Produce a Return Code

In the following situations, DL/I is unable to return to the caller and pass back response information through the AIB:

- Load for DLZMPX00 has failed (message DLZ150I).
- Non-compatible environment (message DLZ151I).
- DL/I exit routine DLZBSEOT is not in the SVA (message DLZ152I).
- GETVIS for the AIB could not be obtained (message DLZ153I).
- DL/I subsystem registration has failed (message DLZ134I).
- AIB parameter missing or invalid (message DLZ154I).

In these cases the according error message is written to the console and the task abnormally terminated (canceled).

Scheduling with Single and Multiple MPS Systems

When only one CICS/DLI MPS system is running, the AIBTDLI interface routes a scheduling request to this MPS system. In this case, you are not required to provide a destination specification.

If MPS has been started in more than one CICS/DLI partition at the same time, you must use a destination specification as described in "Format of the Scheduling Call" on page 324. The destination specification allows the AIBTDLI interface to distinguish between the different MPS systems.

After a successful scheduling call, DL/I returns X'0000' in AIBRCODE.

When a scheduling error has occurred, AIBRCODE contains either:

- One of the existing return codes, as documented in *DL/I Messages and Codes* and the *DL/I 1.11 Release Guide*.
- Return code X'FF00' and a pointer to an error message in AIBMSGPT. The message provides an explanation of why the scheduling has failed.

In the second case (return code X'FF00' and a pointer to an error message in AIBMSGPT), the explanation may be that DL/I was not able to find a suitable CICS/DLI MPS target system. You should then consider these situations:

- Message DLZ089I ('... MPC NOT ACTIVE OR ENDING') is returned in AIBMSGPT. This means that either an MPS system has not been started, or the destination specification passed as 'PARTID=' and/or 'APPLID=' parameter does not match with one of the MPS systems that are currently running.
- Message DLZ145I ('MORE THAN ONE MPS ACTIVE ...') is returned in AIBMSGPT. When MPS is active in more than one CICS/DLI partition, DL/I might not be able to determine the target system. This occurs when either a destination specification is missing in the scheduling call, or 'APPLID=' has been entered, but more than one CICS/DLI MPS system is running with the same CICS (generic) applid.

  If DLZ145I is returned, *AIBPLPT* points to a list of partition ids. The partition ids denote the partitions where an active CICS/DLI MPS system has been found, for the destination specification passed in the scheduling call:
  - If no destination has been specified, all active MPS systems are shown.
  - If 'APPLID=' has been specified, only those MPS systems are shown which are running under the searched-for CICS applid.

  Up to 10 partition ids are returned. They are represented through a list of 2-byte character fields, separated by a comma. A missing comma marks the end of the list.

  The partition ids returned via AIBPLPT can then be used to define/adapt the destination specification for a retry of the scheduling call.

Task Termination and Abend Handling

VSE task termination calls the DL/I task termination exit DLZBSEOT for tasks, which have used the AIBTDLI interface to perform final DL/I cleanup processing. This implies an internal DL/I termination call, if a PSB is still scheduled, because:

- You have not coded a regular DL/I termination call, before terminating your program.
- A termination call could not be given, when a program abend has occurred.

The internal DL/I termination call ends the connection between the task and the CICS/DLI MPS system, releases the PSB, and frees all DL/I resources acquired by this task. For a VSE normal task termination, commit and syncpoint processing is performed. For a VSE abnormal task termination, backout and syncpoint processing is performed.

The AIBTDLI interface does not handle any program errors or abend conditions. Your programs must set up and contain their own exits for dealing with, for example, AB or PC type of abends. Assembler programs may use STXIT linkage, COBOL, and PL/I programs may run with the LE TRAP runtime option.

Note: VSE considers program failures that are handled by exit routines, as "normal" processing conditions. When a program ends with a "normal" processing condition, VSE normal task termination takes place. Only unhandled program failures will lead to a VSE abnormal task termination, and backout processing.

Messages and Return Codes

The AIBTDLI interface reuses a subset of the messages available for the existing MPS batch function. Following each DL/I call, the AIBTDLI interface receives a return code from the CICS/DLI online system. These messages and return codes are described in *DL/I Messages and Codes* and the *DL/I 1.11 Release Guide*.

New messages and return codes are described in "AIBTDLI DL/I Messages and Return Codes", on page 365.

All messages are written to the console. Error messages are also passed to the user program via the field AIBMSGPT, return codes are stored in AIBRCODE. See "Return and Status Codes" on page 326 for details.

Chapter 25. Using SOAP for Inter-Program Communication

This chapter describes how you use the Simple Object Access Protocol (abbreviated to SOAP) to send and receive information between CICS programs and other modules, over the Internet. It contains these main sections:

- "Overview of the SOAP Syntax"
- "How the VSE/ESA Host Can Act As the SOAP Server" on page 332
- "How the VSE/ESA Host Can Act As the SOAP Client" on page 334
- "How the IBM-Supplied SOAP Control Blocks Are Used" on page 335
- "Description of the IBM-Supplied SOAP Service (getquote.c)" on page 339
- "Description of the IBM-Supplied SOAP Client (soapclnt.c)" on page 341
- "Using a Java SOAP Client" on page 343
- "Running the IBM-Supplied SOAP Sample" on page 344
- "Writing Your Own SOAP Programs" on page 348

For a general introduction to SOAP, see "Overview of VSE/ESA Support for Web Services and SOAP" on page 8. For detailed information about SOAP, you might go to the Apache SOAP documentation website, whose URL is:

```
http://xml.apache.org/soap/docs/index.html
```

Note!

The implementation of SOAP for VSE/ESA:

1. is for use with the CICS Transaction Server for VSE/ESA only.
2. does not require the use of either:
   - UDDI (Universal Description, Discovery, and Integration),
   - WSDL (Web Services Description Language).

Overview of the SOAP Syntax

You do not usually need to concern yourself with the tagging described here, since it is *automatically* generated by either the:

- SOAP client, and converted to native data by the SOAP server.
- SOAP server, and converted to native data by the SOAP client-processor.

However, for debugging purposes you might require a knowledge of the SOAP tagging. In this case for detailed information, you should refer to the Apache Web site whose URL is given in the previous section. The information below provides you with an overview only.

A SOAP message is a standard XML document containing these main parts:

- A SOAP envelope, that defines the content of the message.
- A SOAP header (optional), that contains header information.
- A SOAP body, that contains call and reply information.

These are the types of element used for the SOAP message:

- The <Envelope> element is the root element of a SOAP message, and defines the XML document to be a SOAP message.

© Copyright IBM Corp. 2000, 2003

- The <Header> element can be used to include additional, application-specific information about the SOAP message. The information here is user-defined. For example it might be used to define the language used for the message.
- The <Body> element is used to define the message itself.
- The <Fault> element can be optionally used within the <Body> element, and is used to supply information about any errors that might have occurred, when the SOAP message was processed.

Here is an example of the SOAP syntax:

```
<soap:Envelope>
  <soap:Body>
    <GetStock>
        <Company>IBM</Company>
    </GetStock>
  </soap:Body>
</soap:Envelope>
```

The above example shows a SOAP XML document used for requesting the IBM share price. It is, of course, much simplified.

How the VSE/ESA Host Can Act As the SOAP Server

Figure 155 on page 333 shows how SOAP can be used in a CICS environment, when the VSE/ESA host acts as the SOAP server that provides *SOAP services* (in the VSE/ESA environment, CICS User Transactions).

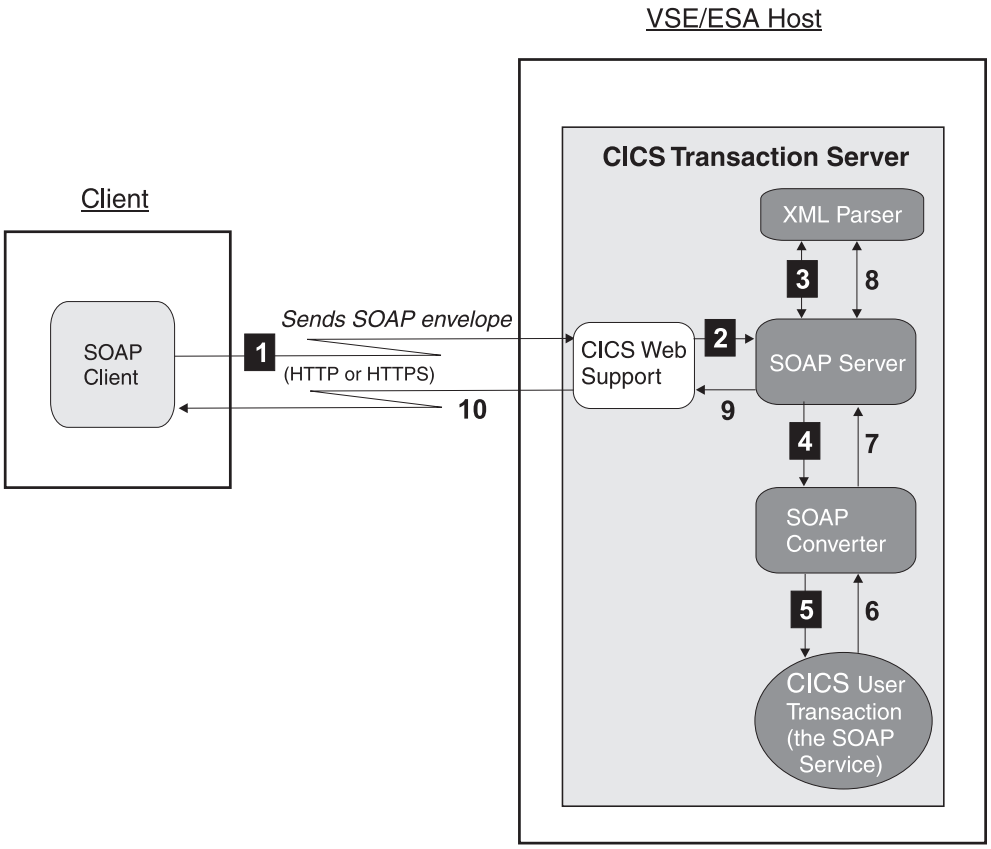


Figure 155. How SOAP Is Used When the VSE/ESA Host Acts As SOAP Server

1. The SOAP client (for example a platform using Microsoft .NET, IBM Websphere, Apache SOAP, or AXIS) sends a SOAP envelope (in XML format) to the SOAP server running under CICS. The SOAP envelope is sent via the CICS Web Support (CWS) component of the CICS Transaction Server.
2. CWS forwards the SOAP envelope (in XML format) to the SOAP server running under CICS.
3. The SOAP server forwards the SOAP envelope to the XML parser, also running under CICS. The XML parser then parses the SOAP envelope from textual XML format into a tree-representation of the data. For example, if the data is to be processed by a C program, the SOAP envelope would be converted to a C program structure (with pointers) so that a C program running on the VSE/ESA host could process the data, and returns this parsed XML tree to the SOAP server.
4. The SOAP server forwards the parsed XML tree to the SOAP converter running under CICS. The SOAP converter de-serializes (decodes) the parameter sub-tree contained in the parsed XML tree, and converts the parameter sub-tree into a binary representation. IBM-supplied SOAP decoder is named IESSOAPD.
5. The SOAP converter forwards the binary representation of the parameter sub-tree to the CICS User Transaction running on the VSE/ESA host (the SOAP service), via the communication area (COMMAREA) of the CICS User Transaction. The CICS User Transaction then processes the data.

The reply is then sent from the CICS User Transaction (the SOAP service) back to the SOAP client, using the reverse of the above steps (that is, steps 6 to 10). The reply is sent via the Communication area back to the SOAP converter, which serializes the parameters and returns them to the SOAP server. The SOAP server uses the XML parser to convert it from a tree-representation of the data to textual XML. The SOAP server then creates a SOAP envelope, which is then sent back (via HTTP or HTTPS) to the SOAP client. The SOAP client can then convert the SOAP envelope to its own native data format, and process the reply.

How the VSE/ESA Host Can Act As the SOAP Client

Figure 156 shows how SOAP can be used in a CICS environment, when the VSE/ESA host acts as the SOAP client:

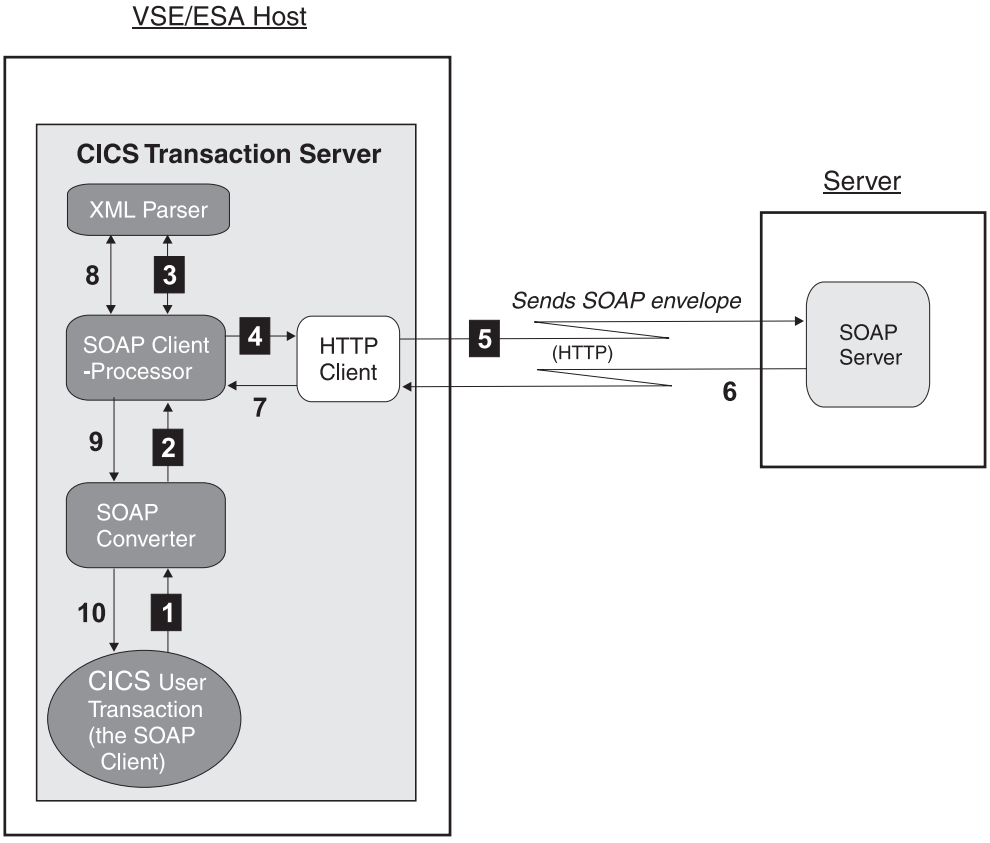


Figure 156. How SOAP Is Used When the VSE/ESA Host Acts As SOAP Client

1 The CICS User Transaction running on the VSE/ESA host (the SOAP client) sends the binary representation of the parameters to the SOAP converter. This is done via the communication area (COMMAREA) of the SOAP converter.

2 The encoder-part of the SOAP converter (IESSOAPE) serializes (encodes) the binary representation of the parameters, into an XML tree. The SOAP converter forwards the XML tree to the SOAP client-processor running under CICS.

3. The SOAP client-processor generates the SOAP envelope, and forwards it to the XML parser running under CICS. The XML parser then converts the XML tree into a textual XML format of the SOAP envelope. The XML parser returns the textual XML format to the SOAP client-processor.
4. The SOAP client-processor forwards the textual XML format to the HTTP client running under CICS.
5. The HTTP client sends the SOAP envelope (in textual XML format) to a SOAP server (for example a platform using Microsoft .NET, IBM Websphere, Apache SOAP, or AXIS) via HTTP. This can also be routed via a SOCKS or Proxy server.

The reply is then sent from the SOAP server back to the CICS User Transaction (the SOAP client), using the reverse of the above steps (that is, steps 6 to 10). The SOAP server sends the reply back to the HTTP client, which forwards it to the SOAP client-processor. The SOAP client-processor calls the XML parser to parse the textual XML format into a tree representation. The tree is passed to the SOAP converter, which de-serializes the parameters into a binary representation, and forwards them to the CICS User Transaction. The CICS User Transaction then processes the reply.

How the IBM-Supplied SOAP Control Blocks Are Used

This section describes the IBM-supplied SOAP control blocks, that are defined in the C-language header file IESSOAPH.H. You can find IESSOAPH.H in directory **...\vsecon\samples\soap\vseSoapClient**.

File IESSOAPH.H is used by all SOAP programs that run on the VSE/ESA host:
- The SOAP converter, SOAP server, and CICS User Transaction of Figure 155 on page 333.
- The SOAP converter, SOAP client-processor, and CICS User Transaction of Figure 156 on page 334.

Other control blocks, not described here, are used if you want to write your own SOAP converter (when the IBM-supplied converter does not meet your requirements). If you wish to write your own SOAP converter, refer to the header file documents for details of such control blocks.

How the SOAP_PARAM_HDR Control Block Is Used

The SOAP_PARAM_HDR control block is used to provide each parameter's data, consisting of a:
- name
- value (the data itself)
- length of the value
- type of value

as shown below.

```
char         name[16];      // parameter name
char         typename[16];  // data type name
unsigned int length;        // length of block (inc. header)
unsigned int type;          // type (see SOAP_TYPE_xxx)
```

Figure 157. Contents of the SOAP Parameter

1. A parameter's data is either:
   - passed to the CICS User Transaction,
   - generated by the CICS User Transaction.
2. The data is then converted by the SOAP converter either:
   - from native data (shown in Figure 157) to XML,
   - from XML to native data.

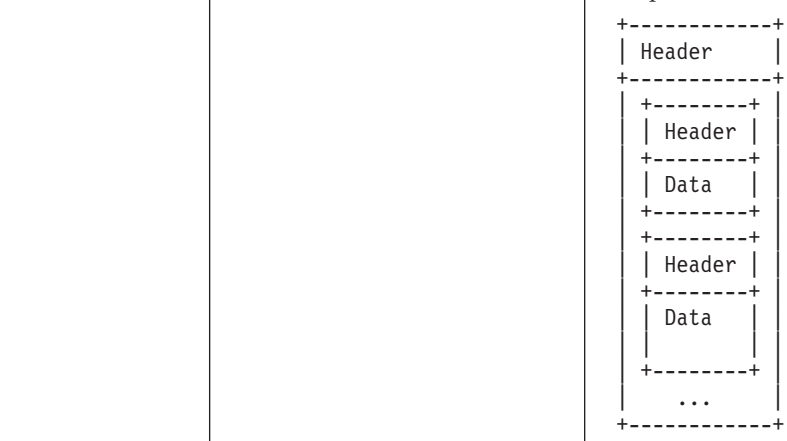
Here is a list of all possible values for the `type` of value field of Figure 157:

```
// Values for type field in SOAP_PARAM_HDR
#define SOAP_TYPE_UNSPECIFIED   0     // unknown/unspecified type
#define SOAP_TYPE_PRIVATE       1     // private type
#define SOAP_TYPE_STRUCT        2     // hirarchical structure
#define SOAP_TYPE_STRING        10    // String
#define SOAP_TYPE_INTEGER       11    // Integer (4 bytes)
#define SOAP_TYPE_SHORT         12    // Short (2 bytes)
#define SOAP_TYPE_BYTE          13    // Byte (1 byte)
#define SOAP_TYPE_BOOLEAN       14    // Boolean (1 byte)
#define SOAP_TYPE_BINARY        15    // Binary (XML Base64)
```

Figure 158. Possible Values for Type of Value Field

The entries in Figure 158 are self-explanatory, except for the three fields described below.

If type is ...	**Then the SOAP converter ...**
SOAP_TYPE_PRIVATE	found an unknown type which had a namespace URL. In this case, the data is not converted and is given as plain text. Your program must then check if it knows the namespace URL/type pair and convert the data itself. The namespace URL is contained in the SOAP_PROG_PARAM control block.
SOAP_TYPE_UNSPECIFIED	found an unknown type *without* a given namespace URL. In this case the data is not converted and is given as plain text. Your program must therefore check if it recognizes the type (even without the namespace), and must then convert the data itself.

If type is ...	Then the SOAP converter ...
SOAP_TYPE_STRUCT	received a hierarchical structure. This might be a type of array, or even an array that is a member of an enclosing array (the depth of the hierarchy is unlimited). An example of a hierarchical structure might be:  The "outer" header contains the length of this header's data. The type is SOAP_TYPE_STRUCT. In this example, the data of the "outer" header contains two parameters (each consisting of a header and data). Each parameter can be of any length. The program can recognize if there are further parameters by checking the length of the outer header's data block against the current parameter length.

How the SOAP_PROG_PARAM Control Block Is Used

The SOAP_PROG_PARAM control block is passed to a CICS User Transaction when the transaction is called as a *SOAP service* (for example in Figure 155 on page 333 the CICS User Transaction running on the host would receive SOAP_PROG_PARAM at Step 5).

The CICS User Transaction is called by the converter program (for example IBM-supplied IESSOAPE). It contains the requested SOAP method name, the names of the input and output queue to use and maybe a namespace URL for the contained parameter types (see SOAP_PARAM_HDR above). If you want to use a private type (SOAP_TYPE_PRIVATE) as a parameter to return, you can specify your own namespace URL here. The converter will build the XML using your namespace/type pair.

Here is a list of fields contained in the SOAP_PROG_PARAM control block:

```
char  method[16];        // (in)     method name
char  inqueue[8];        // (in)     input params
char  outqueue[8];       // (in)     output params
char  namespaceurl[128]; // (in/out) private namespace url
```

Figure 159. Fields Contained in SOAP_PROG_PARAM Control Block

How the SOAP_DEC_PARAM Control Block Is Used

The SOAP_DEC_PARAM control block must be used by a CICS User Transaction acting as a SOAP client, to call the SOAP converter. The SOAP converter then calls the SOAP client-processor, which performs the SOAP call to the SOAP server. For details, see Figure 156 on page 334.

If the field proxytype is not HTTP_TYPE_DIRECT (0), you have to fill in the required proxy fields for this type.

```
char  url[128];         // (in) the servers url
char  method[16];       // (in) method name
char  urn[128];         // (in) the urn
char  inqueue[8];       // (in) input queue name
char  outqueue[8];      // (in) output queue name
char  namespaceurl[128]; // (in/out) namespace url
// proxy
int   proxytype;        // (in) proxy type (HTTP_TYPE_xxx)
char  proxy[128];       // (in) proxy server
int   proxyport;        // (in) port number
char  userid[16];       // (in) userid for socks
char  password[16];     // (in) password for socks 5
```

Figure 160. Fields Contained in SOAP_DEC_PARAM Control Block

These are the defined proxy types you can use if no direct connection to the SOAP server is available:

```
// Proxy types
#define HTTP_TYPE_DIRECT      0 // direct connection
#define HTTP_TYPE_PROXY       1 // connection through a proxy
#define HTTP_TYPE_SOCKS4      2 // connection through Socks V4
#define HTTP_TYPE_SOCKS5      3 // connection through Socks V5
```

Figure 161. Proxy Types That Can Be Used With SOAP_DEC_PARAM Control Block

Description of the IBM-Supplied SOAP Service (getquote.c)

The source code for the IBM-supplied SOAP service is `getquote.c`. You can find this C for CICS program in directory **...\vsecon\samples\soap\vseSoapService**. Also in this directory you will find the jobs to compile and link `getquote.c` (jobs `compile.job` and `link.job`).

1. SOAP Server Calls SOAP Service
The program `getquote` is invoked by the SOAP server (via the SOAP converter), as shown in Step 5 of Figure 155 on page 333. Each request contains the:
- TargetObjectURI, which defines which object is to be called for this request
- method name of the method to call in this object.

The URI (Uniform Resource Identifier) used for `getquote` is `urn:iessoapd:getquote`. It always starts with `urn`, followed by the name of the:
1. SOAP converter (the IBM-supplied `iessoapd`)
2. SOAP service (the IBM-supplied C for CICS program `getquote`).

You could use your own SOAP converter instead of `iessoapd`. The method name for the IBM-supplied sample is `getQuote`, which is the name given to the SOAP service. The SOAP service uses the method name to decide which operation is do be done. Therefore each SOAP service (or CICS program) can handle several methods.

2. Map COMMAREA to SOAP_PROG_PARAM
When the SOAP service is started, it must retrieve the pointer to the provided CICS COMMAREA. The COMMAREA must be mapped to the C structure SOAP_PROG_PARAM, as shown in Figure 162.

```
SOAP_PROG_PARAM* call;
EXEC CICS ADDRESS COMMAREA(call);
```

Figure 162. Mapping COMMAREA to SOAP_PROG_PARAM Control Block

3. Check Which Method is Requested
The SOAP service can now check for the method that has been requested, as shown in Figure 163. The IBM-supplied SOAP service knows of only one method, `getQuote`: all other method names will produce an error code of 1. If the correct method name is requested, the SOAP service calls a sub-function, and provides two CICS queues that were built by the SOAP server to:
1. pass the given SOAP parameters to the SOAP service (via `inqueue`)
2. receive the output parameter from the SOAP service (via `outqueue`).

```
// check if the SOAP method name is 'getQuote'
if(strncmp(call->method, "getQuote", 8) == 0 )
    rc = ProcessGetQuote(call->inqueue, call->outqueue);
else
    rc = 1;
```

Figure 163. Checking Which SOAP Method Has Been Requested

4. Get Input Parameters
The SOAP service gets the input parameters sequentially by reading them from the CICS `inqueue`. The SOAP service then maps the value that was read to the SOAP_PARAM_HEADER structure, as shown in Figure 164 on page 340. `pName` is

the element name of the parameter, pPtr is a pointer to the data, and pLen is the length of the data.

```
SOAP_PARAM_HDR* param;
unsigned short length;
EXEC CICS READQ TS QUEUE(inqueue)
    SET(param) LENGTH(len) NEXT
    RESP(resp) RESP2(resp2);
if(param->type != SOAP_TYPE_STRING)
    return 5; // invalid type (for this service)
*pName = (char*)&param>name;
*pPtr  = (char*)&param[1];
*pLen  = param->length - sizeof(SOAP_PARAM_HDR);
```

Figure 164. Get Input Parameters from CICS Queue

5. Put Value into CICS Output Queue
To put a parameter into the CICS outqueue the SOAP service then copies the parameter into a SOAP_PARAM_HEADER structure, and writes this structure into the CICS outqueue , as shown in Figure 165.

```
int SetNextOutParameter(char*outqueue,
        char *typename, unsigned int type,
        char* name, char* ptr,int len)
{
  SOAP_PARAM_HDR* param;
  int resp,resp2;
  param = (SOAP_PARAM_HDR*)malloc(sizeof(SOAP_PARAM_HDR)+len);
  if(param==NULL)
      return(-1);
  memset(param->name,' ',sizeof(param->name));
  memcpy(param->name,name,strlen(name));
  memset(param->typename,' ',sizeof(param->typename));
  memcpy(param->typename,typename,strlen(typename));
  param->type = type;
  param->length = len + sizeof(SOAP_PARAM_HDR);
  memcpy(&param[1],ptr,len);
  EXEC CICS WRITEQ TS QUEUE(outqueue)
            FROM(param) LENGTH(len+sizeof(SOAP_PARAM_HDR))
            RESP(resp) RESP2(resp2);
  free(param);
  return(0);
};
```

Figure 165. Put Parameter Into the CICS Output Queue

Remaining Processing
The remaining processing is fairly uncomplicated:

1. The SOAP service reads the first parameter from the CICS inqueue, and checks if the:
   - parameters name is symbol
   - type is SOAP_TYPE_STRING.

   If not, the SOAP service returns an error code to the SOAP server.
2. For illustration purposes, the SOAP service uses a hard-coded symbol value, and puts this value (as a parameter) into the CICS outqueue together with the name data and SOAP_TYPE_STRING.
3. The SOAP service should now exit, returning an error code of zero to the SOAP server (via the SOAP converter). The SOAP server creates the XML

representation of the returned value in the CICS outqueue and sends it back to the SOAP client (see Steps 9 and 10 of Figure 155 on page 333).

Description of the IBM-Supplied SOAP Client (soapclnt.c)

The source code for the IBM-supplied SOAP client is soapclnt.c. You can find this C for CICS program in directory **...\vsecon\samples\soap\vseSoapClient**. Also in this directory you will find jobs to compile and link soapclnt.c (jobs compile.job and link.job).

1. Preparing to Call the SOAP Service
Program soapclnt.c calls the SOAP service residing on the VSE/ESA host, as shown in Steps 1 to 5 of Figure 156 on page 334.

Each call contains the:
- URL (Uniform Resource Locator) of the SOAP server
- name of the program that is to be called by the SOAP server (the URI)
- method to be requested from the called program (METHOD).

An example is shown in Figure 166.

```
char *URL = "http://9.164.155.95:1080/cics/CWBA/IESSOAPS";
char *URN = "urn:iessoapd:getquote";
char *METHOD = "getQuote";
```

Figure 166. Preparing the SOAP Client's Call Parameter

2. Prepare the SOAP_DEC_PARAM Structure
The SOAP client prepares a structure of type SOAP_DEC_PARAM by copying the values required for the call into the SOAP_DEC_PARAM structure, as shown in Figure 167.

```
// prepare the call
strncpy(call.url, URL, strlen(URL));
strncpy(call.method, METHOD, strlen(METHOD));
strncpy(call.urn, URN, strlen(URN));
strncpy(call.inqueue, "INPUT   ",8);
strncpy(call.outqueue,"OUTPUT  ",8);
call.proxytype = 0;
```

Figure 167. SOAP Client Prepares the SOAP_DEC_PARAM Structure

3. Insert Parameters into CICS Input Queue of the SOAP Server
The SOAP client uses the CICS inqueue and outqueue (as does the SOAP service) to transmit parameters. In the IBM-supplied example, the SOAP client requests the stock quote for symbol IBM. In response, a parameter called symbol, with value IBM, and type SOAP_TYPE_STRING, is inserted into the CICS inqueue as shown in Figure 168.

```
SetNextOutParameter("INPUT   ",
    "string", SOAP_TYPE_STRING,
    "symbol", "IBM", 3);
```

Figure 168. SOAP Client Inserts Values into the SOAP Server's Input Queue

4. Call SOAP Converter to Handle Requests
Now that the preparation of the call is complete, the SOAP client can call the SOAP converter (IESSOAPE) to handle the SOAP client requests, as shown in Figure 169. The SOAP converter in turn calls the SOAP client-processor (as shown in Step 2 of Figure 156 on page 334).

```
EXEC CICS LINK PROGRAM("IESSOAPE")
    COMMAREA(&call) LENGTH(sizeof(call))
    RESP(rc) RESP2(rc2);
```

Figure 169. SOAP Client Calls SOAP Converter (IESSOAPE) to Handle Requests

5. Obtain Results of the SOAP Call
After the SOAP client has called the SOAP converter (IESSOAPE), it can then obtain the result of this call, as shown in Figure 170. The variable *call.namespaceurl* contains the namespace URL that was used by the SOAP service that was called, to encode the content of the reply. If *namespace url* contains a value, then the URL was *not* know to the decoder. The SOAP client must therefore deserialize the value of the parameter itself.

```
rc = GetNextInParameter("OUTPUT  ",&name,&val,&len);
if(rc!=0)
    break;
cicsprintf(  "name/type = %.32s", name );
cicsprintf(  "len       = %d", len);
sprintf(temp,"val       = %s%ds", "%.", len);
cicsprintf(temp,val);
```

Figure 170. SOAP Client Obtains Results of the SOAP Call

6. Delete CICS Queues That Were Created
After the SOAP call is completed, the SOAP client must now delete the temporary CICS queues that were automatically created during the preparation of the call. This ensures that all memory is given back to CICS. This is shown in Figure 171.

```
EXEC CICS DELETEQ TS QUEUE("INPUT   ")
    RESP(rc) RESP2(rc2);
EXEC CICS DELETEQ TS QUEUE("OUTPUT  ")
    RESP(rc) RESP2(rc2);
```

Figure 171. SOAP Client Deletes CICS Queues

Using a Java SOAP Client

Instead of using the IBM-supplied SOAP client written in C for CICS (`soapclnt.c`), you can use a Java SOAP client.

The SOAP client `GetQuote.java` is supplied with the Apache SOAP distribution. This example is stored in the Apache SOAP directory **soap-2_3_1\samples\stockquote\GetQuote.java**, contained in the Apache SOAP package (see "Step 1: Download and Install the Java SOAP Client Packages on the Client" on page 344 for details).

You can also find an IBM-modified version of `GetQuote.java` in the directory **...\vsecon\samples\soap\javasample**, together with batch files to compile and run the program. The sample has been modified so that it calls the IBM-supplied `getQuote` SOAP service, as shown by the highlighted statement of Figure 172.

```
...
// Build the call.
Call call = new Call ();
call.setTargetObjectURI ("urn:iessoapd:getquote");
call.setMethodName ("getQuote");
call.setEncodingStyleURI(encodingStyleURI);
Vector params = new Vector ();
params.addElement (new Parameter("symbol", String.class, symbol, null));
call.setParams (params);
...
```

Figure 172.

As you can see, the setup of the call is similar to the setup when a SOAP client runs on the VSE/ESA host (as shown in Figure 167 on page 341). The Java client sample performs the same processing as the SOAP client sample (`soapclnt`) that runs on the VSE/ESA host.

You must change the `run.bat` file (contained in the directory **...\vsecon\samples\soap\javasample**) so that the URL of the SOAP server and the requested symbol, are given to the program on the command line. For details, refer to "Running the IBM-Supplied SOAP Sample" on page 344.

Running the IBM-Supplied SOAP Sample

The IBM-supplied SOAP sample consists of three programs:

- GetQuote.java (which is a SOAP client running on a Java-enabled platform)
- getquote.c (which implements a SOAP service on the VSE/ESA host)
- soapclnt.c (which is used when the VSE/ESA host acts as a SOAP client)

The program getquote.c can be called as a SOAP service from either GetQuote.java or soapclnt.c . The IBM-supplied SOAP sample illustrates both of these scenarios, where the VSE/ESA host acts as:

- a SOAP client (when soapclnt.c is the SOAP client)
- a SOAP server (when GetQuote.java is the SOAP client).

To run the IBM-supplied SOAP sample, you must follow the steps described in this section. However, before you can run this SOAP sample you must have:

- JDK 1.3 or later installed.
- The IBM C-compiler for VSE/ESA installed.

Step 1: Download and Install the Java SOAP Client Packages on the Client

You must download various packages, if you do not already have the required files installed. These are the packages you require:

- Apache SOAP package, which you can obtain from URL `http://xml.apache.org/soap/`. At this Web site, go to the directory containing the latest version, and download the soap-bin package (for example,**soap-bin-2.3.1.zip**).
- Apache xerces XML Parser, which you can obtain from URL `http://xml.apache.org/xerces-j/index.html`. At this Web site, download the latest Xerces-J-bin package (for example **Xerces-J-bin.1.4.4.zip**).
- Sun J2EE, which you can obtain from URL `http://java.sun.com/j2ee/download.html`. At this Web site, download the latest J2EE SDK package.
- Sun Java Mail API, which you can obtain from URL `http://java.sun.com/products/javamail/`.
- Sun JavaBeans Activation FrameWork (JAF), which you can obtain from URL `http://java.sun.com/products/javabeans/glasgow/jaf.html`.

Step 2: Extract and Install the Required Java Programs

Step 2.1: Create a New Directory

In this step, you create a new directory. To simplify the CLASSPATH definition, all .JAR files required for running the SOAP sample will be saved into this directory. In the steps below, this directory is called **work**.

Step 2.2: Install the SUN J2EE Package

1. Install the Sun J2EE package. For example, start the `j2sdkee-1_3_1-win.exe`, and then follow the instructions provided.
2. Change to the directory **j2sdkee1.3.1/lib** (assuming you installed J2EE into **j2sdkee1.3.1**).
3. Copy the file `j2ee.jar` into the directory **work**.

Step 2.3: Extract JAR Files and Place in Your Directory

In this step you extract the required **.JAR** files from the downloaded ZIP files, and put the extracted JAR files into your directory **work**:

For the ...	You should extract the file ...
Apache SOAP package	`soap.jar` from the `soap-bin-2.3.1.zip` file.
Apache xerces XML Parser	`xerces.jar` from the `Xerces-J-bin.1.4.4.zip` file.
Sun Java Mail API	`mail.jar` from the `javamail-1_2.zip` file
Sun JavaBeans Activation FrameWork (JAF)	`activation.jar` from the `jaf1_0_1.zip` file.

Step 3: Compile /Link the Sample C Programs, and Define Them to CICS

Before you can start to use the IBM-supplied C-for-CICS programs (the SOAP client and the SOAP service), you must compile and link the source for these programs, and then define the programs to CICS. You should therefore:

1. change the URL in the SOAP client (soapclnt.c) to match your VSE/ESA system. The sample URL is as follows:

   ```
   http://9.164.123.23:1080/cics/CWBA/IESSOAPS
   ```

   (Usually, you simply need to change the IP-address).
2. upload the source for:
   - soapclnt.c from directory **...\vsecon\samples\soap\vseSoapClient**
   - getquote.c from directory **...\vsecon\samples\soap\vseSoapService**

   to a VSE/ESA library. Use either the TCP/IP **ftp** utility, or the File Transfer function provided with a 3270 Emulator.
3. compile source programs soapclnt.c and getquote.c using the IBM-supplied jobs `compile.job` and `link.job`. which you can find in the same directories as the relevant source programs (soapclnt.c or getquote.c). You might need to adapt these jobs to your own environment (source and destination library names, and so on). The compile Job performs the following processing:
   a. Invokes the CICS pre-compiler.
   b. Invokes the C compiler.
   c. Catalogs the object deck into the VSE/ESA library that you specify in your compile job.

   The link Job performs the following processing:
   a. Invokes the LE pre-linker.
   b. Invokes the linkage editor.
   c. Catalogs the phase into the VSE/ESA library that you specify in your link job.
4. define the programs SOAPCLNT and GETQUOTE to CICS, using the CEDA transaction. You must also ensure that the phase you created above, can be located by CICS (using the LIBDEF statement of CICS). Figure 173 on page 346 shows how you use CEDA for the GETQUOTE program.

```
CEDA DEFINE PROGRAM
OVERTYPE TO MODIFY
 CEDA  DEFine PROGram(          )
  PROGram      ==> GETQUOTE
  Group        ==> VSESPG
  DEscription  ==> SAMPLE SOAP SERVICE GETQUOTE
  Language     ==> C                  COBol | Assembler | C | Pli
  RELoad       ==> No                 No | Yes
  RESident     ==> No                 No | Yes
  USAge        ==> Normal             Normal | Transient
  USEsvacopy   ==> No                 No | Yes
  Status       ==> Enabled            Enabled | Disabled
  RSl            : 00                 0-24 | Public
  Cedf         ==> Yes                Yes | No
  DAtalocation ==> Any                Below | Any
  EXECKey      ==> User               User | Cics
 REMOTE ATTRIBUTES
  REMOTESystem ==>
  REMOTEName   ==>
  Transid      ==>
```

Figure 173. Using CEDA to Define Sample SOAP Service to CICS

Step 4: Define the SOAP Server to CICS

To activate the IBM-supplied SOAP server, you must define a TCP/IP service for CICS. All SOAP clients (whether written in C-for-CICS or Java) require that you perform this step. Perform the CEDA DEFINE shown below.

```
CEDA   DEFine TCpipservice( SOAP     )
 TCpipservice   : SOAP
 Group          : VSESPG
 Description  ==> TCP/IP SERVICE FOR SOAP SEVRER
 Urm          ==> DFHWBADX
 Portnumber   ==> 01080              1-65535
 Certificate  ==>
 STatus       ==> Open               Open | Closed
 SSl          ==> No                 Yes | No | Clientauth
 Attachsec    ==> Local              Local | Verify
 TRansaction  ==> CWXN
 Backlog      ==> 00001              0-32767
 TSqprefix    ==>
 Ipaddress    ==>
 SOcketclose  ==> No                 No | 0-240000
```

Figure 174. Using CEDA to Define SOAP Server to CICS

Step 5: Activate the ASCII to EBCDIC Converter

An ASCII / EBCDIC converter (DFHCNV) is provided in VSE/ICCF library 59. It is used by CICS Web Support (and therefore by the SOAP server) to convert:

- Incoming XML data from ASCII to EBCDIC.
- Outgoing XML data from EBCDIC to ASCII.

To activate the ASCII / EBCDIC converter, submit skeleton DFHCNV to the VSE/POWER reader queue.

Step 6: Compile the Java Sample

1. Copy these files from the directory **...\vsecon\samples\soap\javasample** to your directory **work**:
   - compile.bat

- run.bat
- GetQuote.java

2. Run the file `compile.bat`.

Step 7: Run the Java SOAP Client Sample

1. Open the file `run.bat` using a text editor, and change the URL to match your own VSE/ESA system. The sample URL is as follows:

   ```
   http://9.164.123.23:1080/cics/CWBA/IESSOAPS
   ```

   (Usually, you simply need to change the IP-address).
2. Start the `run.bat` file. A SOAP call is then sent from the SOAP client to the SOAP server on the VSE/ESA host, requesting the price for the stock symbol of IBM. The sample SOAP service running on the VSE/ESA host then returns a hard-coded symbol value to the Java program.
3. The Java SOAP client outputs the value of the stock (which is "hard-coded" in the program!) to the screen.

Step 8: Run the C-Program SOAP Client Sample

To run the C program soapclnt.c, you must:

1. define (again using CEDA) a CICS transaction that will call the CICS program SOAPCLNT that you defined in "Step 3: Compile /Link the Sample C Programs, and Define Them to CICS" on page 345. **Note:** ensure that you have changed the URL in the SOAP client (soapclnt.c) to match your VSE/ESA system!.
2. start the CICS transaction defined above. A SOAP call is then sent from the SOAP client to the SOAP server on the VSE/ESA host, requesting the price for the stock symbol of IBM. The sample SOAP service running on the VSE/ESA host then returns a hard-coded symbol value to the C program.

Writing Your Own SOAP Programs

The IBM-supplied sample which you can use as a template for writing your own programs, consists of:

- a SOAP C language header file `IESSOAPH.H` , which is described in "How the IBM-Supplied SOAP Control Blocks Are Used" on page 335. The COMMAREA and memory mappings used by the VSE/ESA SOAP implementation are defined in `IESSOAPH.H`. If you want to write SOAP services or clients in another language (such as COBOL) you must adopt these definitions to this language by yourself. A mapping for the assembler language is included in `IESSOAPH.H` as comments.
- a SOAP service `getquote.c` written in C for CICS, which is described in "Description of the IBM-Supplied SOAP Service (getquote.c)" on page 339
- a SOAP client `soapclnt.c` written in C for CICS, which is described in "Description of the IBM-Supplied SOAP Client (soapclnt.c)" on page 341.
- a SOAP client `GetQuote.java` written in Java, which is described in "Using a Java SOAP Client" on page 343.

If the SOAP converters IESSOAPD and/or IESSOAPE do not meet your requirements, you can replace them with your own programs. You specify the name of the SOAP converter for the VSE/ESA inbound direction in the URI of the SOAP request. For details, see "Description of the IBM-Supplied SOAP Service (getquote.c)" on page 339.

Chapter 26. Using the VSE Script Connector for Non-Java Access

This chapter describes how you use the *VSE Script connector* (which consists of the VSE Script Client and the VSE Script Server) to access VSE/ESA host data from Java or *non-Java* platforms. Since access from non-Java platforms is the main advantage of using the VSE Script Connector, this has been included in the title of this chapter.

For each VSE Script Client that you write, you will most probably also need to write a corresponding VSE Script (using the VSE Script Language), to meet your own specific requirements. However, new VSE Script Clients can use existing VSE Scripts to access VSE/ESA host data.

This chapter contains these main sections:

- "How the VSE Script Connector Is Used"
- "Overview of the Protocol Used Between Client and Server" on page 350
- "Writing VSE Scripts Using the VSE Script Language" on page 351
- "Sample Files You Can Use for Writing VSE Script Clients" on page 355
- "Example of Writing a VSE Script Client (and Its VSE Script)" on page 356

Related Sections:

- "Overview of the VSE Script Connector" on page 7
- Chapter 7, "Installing the VSE Script Connector", on page 43

How the VSE Script Connector Is Used

Figure 175 shows how a VSE Script can be called by a VSE Script Client, to obtain data from the VSE/ESA host.

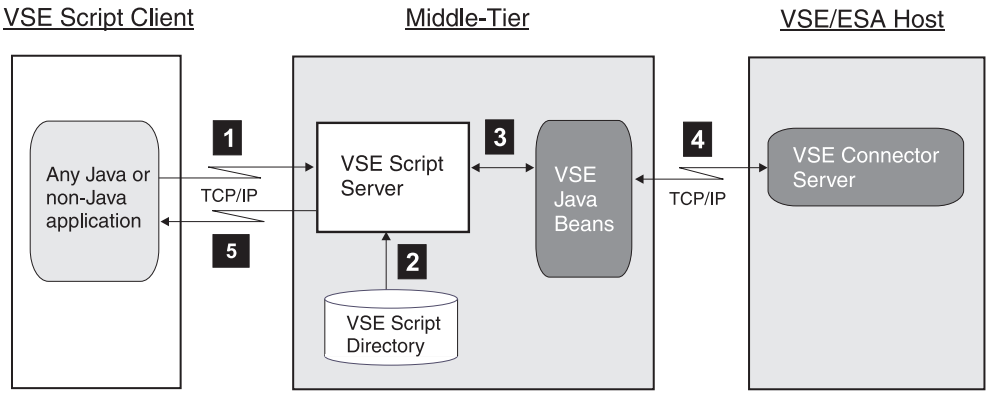


Figure 175. How the VSE Script Connector is Used

1 The VSE Script Client establishes a TCP/IP connection to the VSE Script Server on the middle-tier. The VSE Script Client then calls a VSE Script by sending:

- the file name of the VSE Script.
- the parameters belonging to the VSE Script

© Copyright IBM Corp. 2000, 2003

to the VSE Script Server running on a middle-tier and Java-enabled platform.

2 The VSE Script Server reads the VSE Script file from the VSE Script directory, and starts to interpret and translate the VSE Script file statements into VSE Java Beans requests. Each VSE Script file is written using the *VSE Script Language* (described in "Writing VSE Scripts Using the VSE Script Language" on page 351).

3 The VSE Java Beans requests are executed by the VSE Java Beans.

4 The VSE Java Beans communicate with the VSE Connector Server running on the VSE/ESA host, which performs the request for functions and data.

5 The data that was obtained by the VSE Java Beans is then converted to a format that the VSE Script Client can use, and is returned to the VSE Script Client.

Overview of the Protocol Used Between Client and Server

This section describes the protocol used for the data flow between a VSE Script Client and the VSE Script Server on the middle-tier. The general flow of actions is as follows:

1. The VSE Script Client opens a connection to the VSE Script Server.
2. The VSE Script Client then sends to the VSE Script Server:
   a. the name of a VSE Script, followed by a `CR LF (X'0D' X'0A')`
   b. each parameter value, followed by `CR LF`
3. After the last parameter has been sent, an empty line is sent (only `CR LF`). This indicates that all parameters have been transmitted.
4. The VSE Script Server starts to execute the VSE Script.
5. The VSE Script Server sends the output from the VSE Script back to the VSE Script Client, line-by-line. Each line is terminated by `CR LF`.
6. After it has sent the last output line to the VSE Script Client, the VSE Script Server then closes the connection.

You can easily implement the protocol described here into any kind of programming language. The programming language is only required to support calls to TCP/IP socket functions.

You can use a Telnet application to easily test the VSE Script Connector. To do so, you must:

1. Create a connection between the telnet client and the VSE Script Server.
2. Type the name of your VSE Script and press Enter to terminate the line with `CR LF`.
3. Type the name of each parameter and press Enter to terminate the line with `CR LF`.

   Note: You will not see what you type, since the VSE Script Server does not echo the data it receives.
4. After you have entered the last parameter, press Enter to start executing your VSE Script. The output from the VSE Script will now be displayed by the telnet client.

The example below shows a sequence where a VSE Script called `test.src` which contains three parameters, is executed. The parameters are `Hello`, `583`, and `get`. The

VSE Script Server sends the output from the VSE Script (`Script test.src has been executed`) back to the VSE Script Client.

1. VSE Script Client to the VSE Script Server:

```
test.src<CR><LF>
Hello<CR><LF>
583<CR><LF>
get<CR><LF>
<CR><LF>
```

2. VSE Script Server to the VSE Script Client:

```
Script test.src has been executed<CR><LF>
```

Writing VSE Scripts Using the VSE Script Language

The VSE Script connector includes a special programming language, called the *VSE Script Language*, that you can use to write your own VSE Scripts. These VSE Scripts are required for accessing VSE functions and data from your VSE Script Clients. As with other programming languages, the VSE Script Language consists of:

- statements (if, while, for, break, continue, sub, gosub, return)
- operators (logical, concatenation, arithmetic)
- variables
- built-in functions.

The VSE Script Language is described in detail in the *Language Reference* online documentation, which is installed on your workstation during the installation of the VSE Script Server (as described in Chapter 7, "Installing the VSE Script Connector", on page 43.

This section provides an introduction to the VSE Script Language in these sub-sections:

- "General Rules That Apply to the VSE Script Language"
- "VSE Script Language Built-In General Functions" on page 352
- "VSE Script Language Built-In String Functions" on page 353
- "VSE Script Language Built-In Console Functions" on page 353
- "VSE Script Language Built-In POWER Functions" on page 353
- "VSE Script Language Built-In VSAM Functions" on page 354

General Rules That Apply to the VSE Script Language

Statements:

1. A statement can be either a:
   - keyword (IF, WHILE, ...)
   - function call
   - variable declaration
   - variable assignment.
2. Each statement must end with a semicolon (;), and any number of statements can be written on a single line.
3. A command cannot be split over two lines (statements on one line are considered as one command, including the `do` statement).
4. Blank lines are allowed.
5. The VSE Script Language is not case-sensitive.
6. Unrecoverable errors that occur when a VSE Script is interpreted are described as *ScriptErrors*. A ScriptError generates this information:
   - the line number and statement number where the error occurred.
   - the (unique) error number.

- a descriptive text for the error number.

Comments:

1. A comment begins with two forward slashes "//", and ends at the end of the line.
2. A comment can appear behind a `command`, or at the beginning of a line.

Variables:

1. An unlimited number of variables can be declared.
2. Each variable can be declared only once in a VSE Script
3. After they have been declared, all variables are global in the VSE Script.
4. A variable must be declared before use, and can be defined at any time/place.
5. Variable names must start with a alphabetic character. Afterwards, all alphabetic characters, digits and the underscore '_' are allowed.
6. There are three data types: int, string, and boolean.
7. Each variable can be accessed as an array of values.
8. `var[0]` is the same as the variable `var` itself, and therefore `var[0]=1` is the same as `var=1`.
9. The variable ARGV is always defined in a VSE Script, and a variable with this name cannot be declared.
10. The variable is of type STRING, the values of the variable are the parameters that are passed to the VSE Script during startup.

VSE Script Language Built-In General Functions

Function	Description
print / println	prints a comma separated list of values or expressions (int, string, boolean) to standard out
exit	exits the VSE Script with a specified return code
sleep	sends the VSE Script into sleep mode for a specified time interval
resetVar	deletes all values of a given variable, especially useful with arrays
arraySize	gets the actual size of a given array variable
getLastErrorMsg	gets the last internal error message. This message is set by many VSE/ESA access functions to give a detailed description of an error
readFile	reads a specified file line by line into a string array
saveFile	saves the lines in a string array to a file
getCallersIP	gets the IP address of the host that called the VSE Script. This could be used from inside a VSE Script to reject access to the VSE Script if other IP's than the expected try to call it.
call	calls an external program on the machine where the VSE Script Server runs.
callNoWait	calls an external program on the machine where the VSE Script Server runs. The function will return to the VSE Script immediately, it will not wait for the called program to end. Hence the output of the program can't be captured and no exit code is available.

Function	Description
VSEConnect	tries to connect to the specified host. You don't need to call this function explicitly, each function that access the host will automatically connect to VSE if needed. This function can be used to check if a connection is possible before any other operations start. Once established, the connection is pooled for later use.

VSE Script Language Built-In String Functions

Function	Description
formatNumber	formats a given number to a string
toString	converts a variable (int, string, boolean) to a string
toInt	converts an integer or string to an integer.
indexOf	gets the index of the first occurrence of a given string in another string
lastIndexOf	gets the index of the last occurrence of a given string in another string
subString	gets a substring from a given string
trim	removes all leading and trailing blanks from a given string
strLen	gets the length of a string

VSE Script Language Built-In Console Functions

Function	Description
executeConsoleCmd	executes a console command and returns the result lines
putConsoleCmd	executes a console command. The output can be retrieved line by line using getConsoleMsg
getConsoleMsg	retrieves one line from the console
closeConsole	closes the console

VSE Script Language Built-In POWER Functions

Function	Description
executePowerJob	executes a POWER job.
listPowerQueue	retrieves the member list of a POWER queue.
getPowerQueueEntry	receives a specified entry from a specified POWER queue.
putPowerQueueEntry	creates a new POWER queue member in the specified queue.
deletePowerQueueEntry	deletes a specified POWER queue entry
cancelPowerQueueEntry	cancels a specified POWER queue entry
releasePowerJob	releases a specified POWER queue entry in the reader queue
getPowerEntryproperties	gets various properties of a POWER queue entry
setPowerEntryProperties	sets various properties of a POWER queue entry

VSE Script Language Built-In VSAM Functions

Function	Description
getVsamRecord	retrieves a VSAM record from a given VSAM file
getVsamRecordGE	retrieves a VSAM record from a VSAM file using the GreaterEqual search
getNextVsamRecord	retrieves the next record based on given key values.
getPrevVsamRecord	retrieves the previous record based on given key values
deleteVsamRecord	deletes a VSAM record from a VSAM file
insertVsamRecord	inserts a new VSAM record into a VSAM file.
updateVsamRecord	updates a VSAM record of a VSAM file
vsamBrowseStart	starts a browse session for a given VSAM map or view. Only one session can be active for the same map or view. Afterwards you can browse through the returned records using vsamBrowseNext or vsamBrowsePrev. A call to vsamBrowseEnd ends the session and discards the internal browse data.
vsamBrowseNext	can be called after a successful vsamBrowseStart. It positions to the next record of the retrieved records. The function can be called sequentially until the end of data is reached.
vsamBrowsePrev	can be called after a successful vsamBrowseStart. It positions to the previous record of the retrieved records. The function can be called sequentially until the first record is reached.
vsamBrowseEnd	ends the browse session for the specified filename.

Sample Files You Can Use for Writing VSE Script Clients

The files listed in Table 9 are copied to the directory where you installed the VSE Script Server (see "Step 1.2: Perform the Installation of the VSE Script Server" on page 44 for details):

- VSEScriptServlet.java
- VSEScriptWebService.java

You can use the files in Table 9 to write your own VSE Script Clients.

Table 9. Files Supplied for Writing VSE Script Clients

File	Description
VSEScriptClient.dll	Windows DLL for use, for example, with MS Office and Lotus Products
VSEScriptClient.h	C-Header for use with VSEScriptClient.dll
VSEScriptClient.lib	C-Library file for use with VSEScriptClient.dll
VSEScriptClient.123	Lotus 1-2-3 sample that uses VSEScriptClient.dll
VSEScriptClient.lss	Lotus Script source of the Visual Basic script
VSEScriptClient.xls	Microsoft Excel sample that uses VSEScriptClient.dll
VSEScriptClient.bas	Visual Basic source of the Visual Basic Script
VSEScriptCgi.c	Sample C-source of a CGI that uses VSEScriptClient.dll
VSEScriptCgi.exe	Compiled CGI
com\ibm\vse\script\client\ VSEScriptServlet.java	Java source of sample servlet
com\ibm\vse\script\client\ VSEScriptWebService.java	Java source of sample WebService

Example of Writing a VSE Script Client (and Its VSE Script)

This section provides an example of how to write a VSE Script Client and its corresponding VSE Script, to insert VSAM data into office applications such as Lotus 1-2-3 or Lotus Wordpro. The example uses a Windows DLL file **VSEScriptClient.dll**, which is available for use with any Windows programs (for example a Lotus Spreadsheet). You must, however, declare this DLL file using a statement like the one shown for Lotus 1-2-3 in Figure 179 on page 360. These statements are also contained in the online documentation, and you can easily Copy and Paste them as required.

This section has the following sections:

- "Step 1: Setup the VSE Script Server Properties File"
- "Step 2: Setup the Connections Properties File"
- "Step 3: Define the Sample VSAM Data" on page 357
- "Step 4: Modify the Sample VSE Script" on page 357
- "Step 5: Start the VSE Connector Server on the VSE/ESA Host" on page 358
- "Step 6: Start the VSE Script Server Locally" on page 358
- "Step 7(a): Open the Sample Lotus 1-2-3 Spreadsheet File" on page 359
- "Step 7(b): Open the Sample MS Office Spreadsheet" on page 362
- "Step 7(c): Start a Sample VSE Script from the Command Line" on page 364

Other examples of how to program VSE Script Clients for other client platforms are provided in the online documentation supplied with the VSE Script Connector.

Step 1: Setup the VSE Script Server Properties File

The parameters for the VSE Script Server are defined in the file **VSEScriptServer.properties**, which you can find in the directory where you installed the VSE Script Server (see "Step 1.2: Perform the Installation of the VSE Script Server" on page 44 for details). An example of **VSEScriptServer.properties** is shown below.

```
#VSEScriptServer

#print messages (on) or do not print (off)
messages=on

#port where the server listens
listenport=4711

#number of maximum connections allowed
maxconnections=256

#root directory for VSE Scripts
scriptdirectory=./scripts

#name of the connection config file
connectionconfig=Connections.properties
```

The properties file is pre-customized for use with the sample, and you should not normally need to change this file. If you wish to do so, however, see "Step 2: Configure the VSEScriptServer Properties File" on page 45 for details.

Step 2: Setup the Connections Properties File

The **Connections.properties** file defines the settings for the VSE/ESA systems with which you wish to work. You can find **Connections.properties** in the directory where you installed the VSE Script Server. An example of this file is shown below.

```
connection.1.password=mypassw
connection.1.ip=9.12.34.56
connection.1.name=VSE01
connection.1.port=2893
connection.1.userid=jsch
connection.timeout=100
```

For details of the **Connections.properties**, see "Step 3: Configure the Connections Properties File" on page 46.

You can edit the above file using any text editor. Type your VSE password in clear text in the parameter `password`. When the VSE Script Server starts and reads the Connections properties file, the password is encrypted to parameter `encpassword`, and `password` is removed from the file. For details of the Connections properties file, see "Step 3: Configure the Connections Properties File" on page 46.

Step 3: Define the Sample VSAM Data

From VSE/ESA 2.5 onwards, a sample job SKVSSAMP is provided which is stored in ICCF library 59. It enables you to define sample data for various VSE/ESA e-business Connectors samples.

Submit the Job SKVSSAMP to define a VSAM cluster with the name VSAM.CONN.SAMPLE.DATA, in the VSESP.USER.CATALOG. It contains a number of records describing "used cars". For details of job SKVSSAMP, see "4. Define the VSAM Data Cluster" on page 219. Job SKVSSAMP automatically uses the IDCAMS RECMAP command to define a VSAM map that contains the field names of the sample data. For details, see "Defining a Map Using RECMAP" on page 132.

Step 4: Modify the Sample VSE Script

Figure 176 on page 358 shows the sample VSE Script **getdata.src**, which reads a record from the VSAM file defined in "Step 3: Define the Sample VSAM Data", and inserts the data into the spreadsheet. You can find **getdata.src** in sub-directory **/scripts/samples** of the directory where you installed the VSE Script Server. The record is identified by its key, which is passed to the VSE Script by parameter. The VSE Script is used by the Lotus 1-2-3 example of a VSE Script Client, **VSEScriptClient.123**.

Make sure that you have the correct VSE/ESA host defined (using the constant **host**, shown as **VSE01** below). This host name *must* match the one defined in the **Connections.properties**.

```
// constants
String host      = "VSE01";
String file      = "VSESP.USER.CATALOG\\VSAM.CONN.SAMPLE.DATA\\USEDCARS";

// Variables
String keyfields;
String keyvalues;
String fields;
String values;
int rc;

// prepare the fields
keyfields[0] = "ARTICLENO";
keyvalues[0] = argv[0];   // argument is key

fields[0] = "ARTICLENO";
fields[1] = "MANUFACTURER";
fields[2] = "TYPE";
fields[3] = "MODEL";
fields[4] = "HP";
fields[5] = "DISPLACEMENT";
fields[6] = "CYLINDERS";
fields[7] = "COLOUR";
fields[8] = "FEATURES";
fields[9] = "PRICE";

// get the record
getVSAMRecord(host,file,&keyfields,&keyvalues,&fields,&values, &rc);

// print received data

if (rc!=0) do;
  println("Not found");
else do;
  println(values[0]);
  println(values[1]);
  println(values[2]);
  println(values[3]);
  println(values[4]);
  println(values[5]);
  println(values[6]);
  println(values[7]);
  println(values[8]);
  println(values[9]);
endif;
```

Figure 176. VSE Script Provided With the VSE Script Connector Example

Step 5: Start the VSE Connector Server on the VSE/ESA Host

Make sure that the VSE Connector Server is started in **non-SSL** mode. The server runs in class R by default. Use job STARTVCS, which is located in the reader queue, to start the server (for details see "Starting the VSE Connector Server" on page 36).

Step 6: Start the VSE Script Server Locally

To start the VSE Script Server locally, you use either:

- **runserver.bat** on Windows
- **runserver.cmd** on OS/2
- **runserver.sh** on Linux/Unix workstations.

Step 7(a): Open the Sample Lotus 1-2-3 Spreadsheet File

The sample Lotus Spreadsheet file **VSEScriptClient.123** already has the necessary setup to run the VSE Script. You can find **VSEScriptClient.123** in the directory where you installed the VSE Script Server. To open the file, double-click it using the Windows Explorer and you will see the **Execute Script** button as shown in Figure 177.

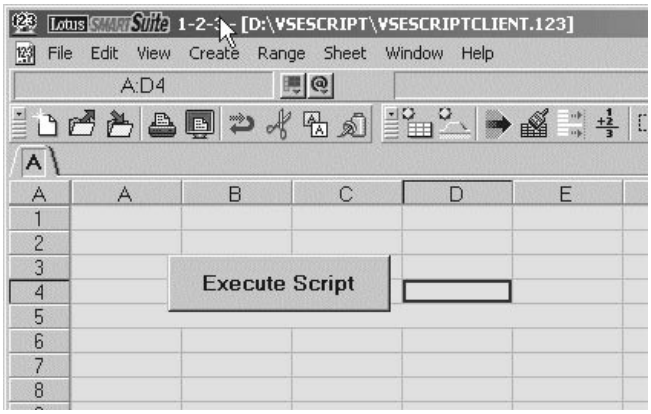


Figure 177. Sample Lotus 1-2-3 Spreadsheet for VSE Script Connector Example

Providing the VSE Script Server is running on the same workstation as your spreadsheet, you can now press **Execute Script** and the data is transferred from the VSAM cluster into the Lotus Spreadsheet as shown in Figure 178. If the VSE Script Server is *not* running on the same workstation as your spreadsheet, you must first modify the Visual Basic script using, for example, a Script editor as shown in "How the Sample VSE Script is Defined in Lotus 1-2-3", so that it contains the IP address or Host name of the workstation where the VSE Script Server is running.

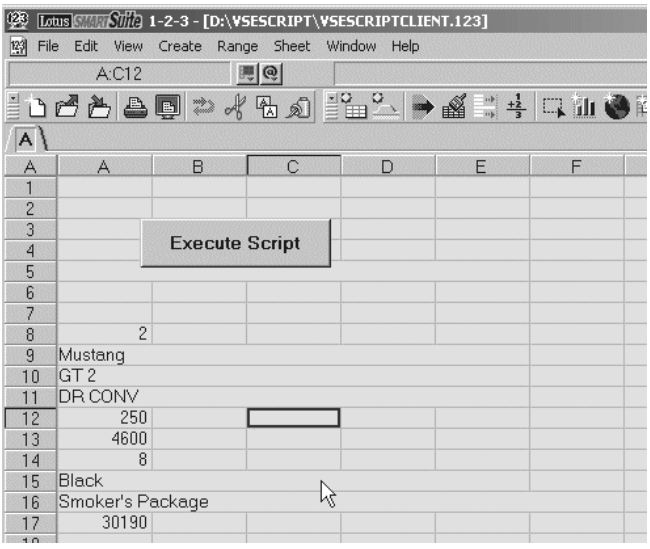


Figure 178. Transferring Data from VSAM Cluster to Lotus 1-2-3 Spreadsheet

How the Sample VSE Script is Defined in Lotus 1-2-3

Before the sample VSE Script can be used from within an office application such as Lotus 1-2-3, you must define it using, for example, a Visual Basic script (as shown in Figure 179 on page 360). When you open the sample file **VSEScriptClient.123**

and open the script editor, you can see the global function declarations in the "Globals" section, as shown in Figure 179.

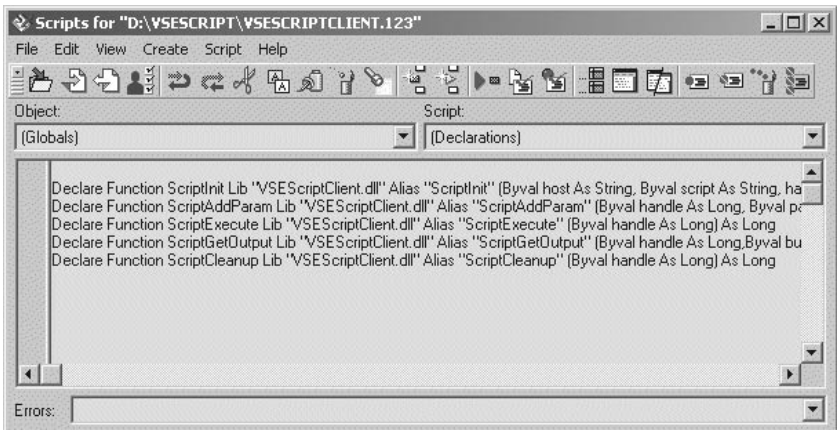


Figure 179. Sample Script As Defined in Lotus 1-2-3

You can also see the Visual Basic script in the section related to **Button 1** in Figure 180. (In Figure 178 on page 359, **Button 1** is labelled as **Execute Script**).

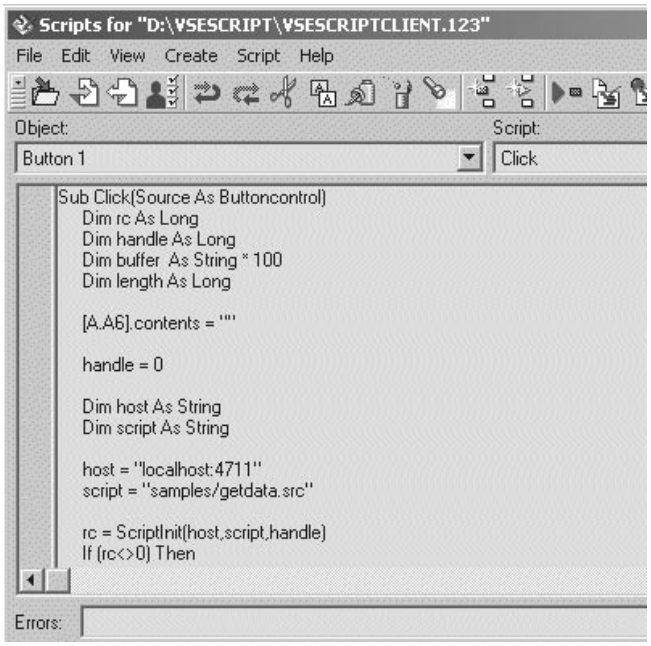


Figure 180. Visual Basic Script Used With Lotus 1-2-3 Spreadsheet Example

Note: **VSEScriptClient.dll** must be accessible by the office application. You may either copy the DLL into the Lotus 1-2-3 DLL directory or simply double-click the sample spreadsheet file in the same directory as the DLL.

Here is the complete Visual Basic code of the VSE Script Client, that is used to run the VSE Script in Lotus 1-2-3. The parts shown in **bold** are specific to the Lotus Spreadsheet environment.

```
Sub Click(Source As Buttoncontrol)
  Dim rc As Long
  Dim handle As Long
  Dim buffer  As String * 100
```

```
  Dim length As Long

  [A.A6].contents = ""

  handle = 0

  Dim host As String
  Dim script As String

  host = "localhost:4711"
  script = "samples/getdata.src"

  rc = ScriptInit(host,script,handle)
  If (rc0) Then
     [A.A6].contents = "rc = " &rc
     Goto finish
  End If

  rc = ScriptAddParam(handle,"2")
  If (rc0) Then
     [A.A6].contents = "rc = " &rc
     Goto finish
  End If

  rc = ScriptExecute(handle)
  If (rc0) Then
     [A.A6].contents = "rc = " &rc
     Goto finish
  End If

  Dim counter As Long
  Dim rows As Range
  Dim cell As Variant

  Set rows = Bind("A8..A65535")

  counter = 0
  Do
     rc = ScriptGetOutput(handle,buffer,100,length)
     If (rc=0) Then
        buffer = Left(buffer,length)

        Set cell = rows.Cell(counter,0,0)
        cell.contents = buffer

        counter = counter + 1
     End If
  Loop While rc=0

  Finish:

  rc = ScriptCleanup(handle)
End Sub
```

Step 7(b): Open the Sample MS Office Spreadsheet

The sample MS Office Spreadsheet file **VSEScriptClient.xls** already has the necessary setup to run the VSE Script. You can find **VSEScriptClient.xls** in the directory where you installed the VSE Script Server. To open the file, double-click it using the Windows Explorer and you will see the **Execute Script** button as shown in Figure 181.

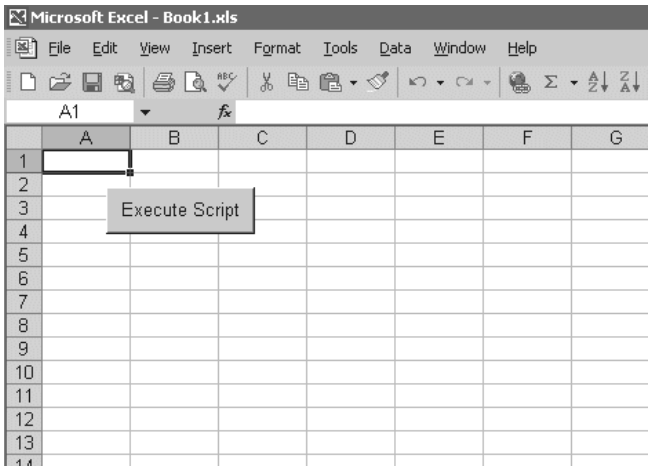


Figure 181. Sample Spreadsheet for MS Office Spreadsheet Example

Providing the VSE Script Server is running on the same workstation as your spreadsheet, you can now press **Execute Script** and the data is transferred from the VSAM cluster into the MS Office Spreadsheet as shown in Figure 182. If the VSE Script Server is *not* running on the same workstation as your spreadsheet, you must first modify the Visual Basic script using, for example, a Script editor as shown in Figure 183 on page 363, so that it contains the IP address or Host name of the workstation where the VSE Script Server is running.

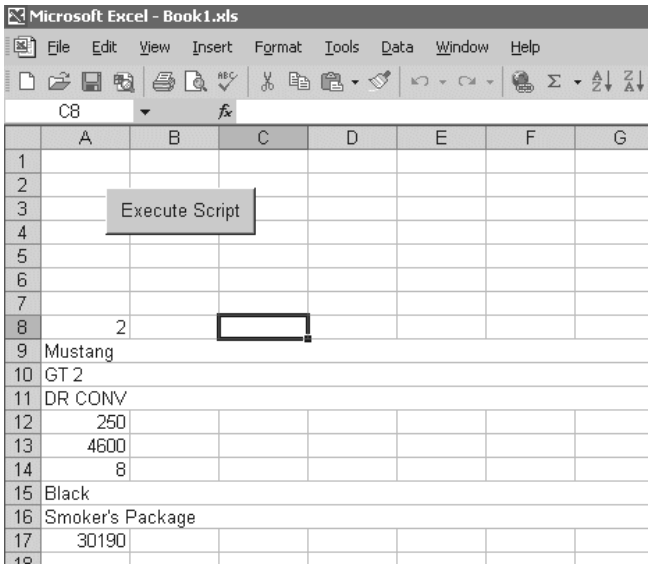


Figure 182. Transferring Data from VSAM Cluster to MS Office Spreadsheet

How the Sample VSE Script is Defined in MS Office

Before the sample VSE Script can be used from within an office application such as MS Office, you must define it using, for example, a Visual Basic script (as shown in Figure 183). When you open the sample file **VSEScriptClient.xls** and open the Visual Basic editor, you can see the global function declarations and the Visual Basic script.

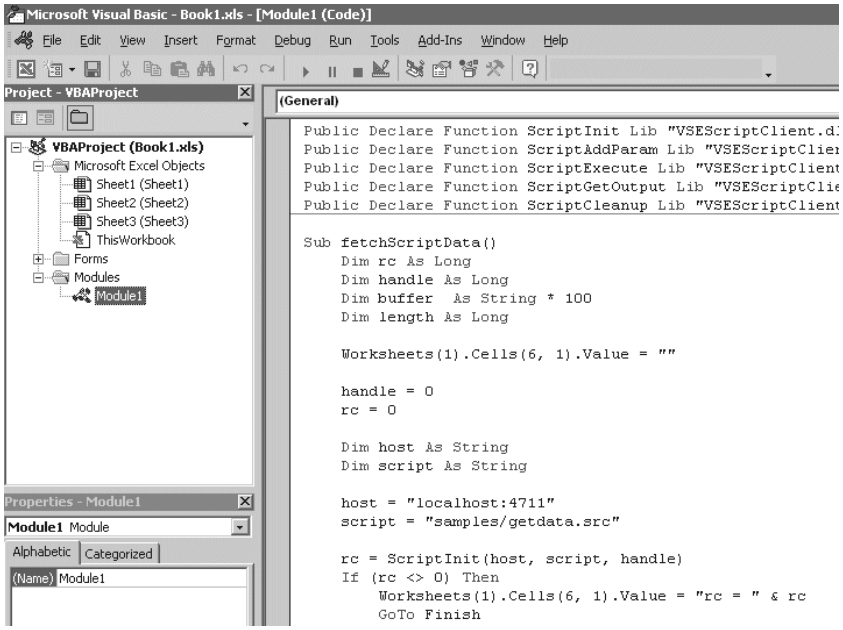


Figure 183. Sample Script as Defined in MS Office

Note: **VSEScriptClient.dll** must be accessible by the office application. You may either copy the DLL into the MS Office directory or simply double-click the sample spreadsheet file in the same directory as the DLL.

Here is the complete Visual Basic code of the VSE Script Client, that is used to run the VSE Script in MS Office. The parts shown in **bold** are specific to the MS Office Spreadsheet environment.

```
Attribute VB_Name = "Module1"
Public Declare Function ScriptInit Lib "VSEScriptClient.dll" (ByVal host As
   String, ByVal script As String, handle As Long) As Long
Public Declare Function ScriptAddParam Lib "VSEScriptClient.dll" (ByVal handle
   As Long, ByVal parameter As String) As Long
Public Declare Function ScriptExecute Lib "VSEScriptClient.dll" (ByVal handle
   As Long) As Long
Public Declare Function ScriptGetOutput Lib "VSEScriptClient.dll" (ByVal handle
   As Long, ByVal buffer As String, ByVal maxlen As Long, retlen As Long) As Long
Public Declare Function ScriptCleanup Lib "VSEScriptClient.dll" (ByVal handle
   As Long) As Long
Sub fetchScriptData()
    Dim rc As Long
    Dim handle As Long
    Dim buffer  As String * 100
    Dim length As Long

    Worksheets(1).Cells(6, 1).Value = ""

    handle = 0
    rc = 0

    Dim host As String
```

```
    Dim script As String

    host = "localhost:4711"
    script = "samples/getdata.src"

    rc = ScriptInit(host, script, handle)
    If (rc <> 0) Then
        Worksheets(1).Cells(6, 1).Value = "rc = " & rc
        GoTo Finish
    End If

    rc = ScriptAddParam(handle, "2")
    If (rc <> 0) Then
        Worksheets(1).Cells(6, 1).Value = "rc = " & rc
        GoTo Finish
    End If

    rc = ScriptExecute(handle)
    If (rc <> 0) Then
        Worksheets(1).Cells(6, 1).Value = "rc = " & rc
        GoTo Finish
    End If

    Dim counter As Long

    counter = 0
    Do
        rc = ScriptGetOutput(handle, buffer, 100, length)
        If (rc = 0) Then
            buffer = Left(buffer, length)

            Worksheets(1).Cells(counter + 8, 1).Value = buffer

            counter = counter + 1
        End If
    Loop While rc = 0

Finish:

    rc = ScriptCleanup(handle)
End Sub
```

Step 7(c): Start a Sample VSE Script from the Command Line

To start a sample VSE Script from the command line, you can use the batch file **runscript.bat** which is located in the directory where you installed the VSE Script Server.

For example, to start the sample script that has the name **break_cont.src**, you would enter at the command line:

```
runscript samples\break_cont.src
```

Note: The **samples** directory contains other sample scripts that you can use for learning and testing purposes.

Appendix. AIBTDLI DL/I Messages and Return Codes

These are the new messages that can be generated by the AIBTDLI interface:

DLZ150I LOAD FOR DLZMPX00 FAILED, CDLOAD RETURN CODE = rc

Explanation: The language interface module DLZLX000 has tried to load the DL/I connector module DLZMPX00 on a DL/I call entered via AIBTDLI. The load has failed for the reason given in the return code 'rc' shown in decimal format. The program is canceled.

For an explanation of the CDLOAD return code *rc* refer to *VSE/ESA Messages and Codes*, Section "VSE/Advanced Functions Codes and SVC errors".

User Response: Correct the error and rerun the program.

DLZ151I NON-COMPATIBLE ENVIRONMENT FOR AIBTDLI INTERFACE

Explanation: A user program has issued a DL/I call via AIBTDLI. The language interface module DLZLX000 has detected that either

- DLZRRC00 for DL/I batch
- DLZMPI00 for MPS batch or
- CICS/DLI online

is already active in this partition. DL/I calls via AIBTDLI are only supported, when no specific DL/I environment has been started in the partition. That means that DLZRRC00, DLZMPI00 or CICS/DLI online may not be running. The program is abnormally terminated.

User Response: Rerun the program in a partition without a pre-established DL/I environment.

DLZ152I DL/I EXIT ROUTINE DLZBSEOT NOT LOADED IN SVA

Explanation: A user program has issued a DL/I call via AIBTDLI, but the DL/I task termination routine DLZBSEOT was not residing in the SVA. The program is canceled.

User Response: Load DLZBSEOT into the SVA and rerun the program.

DLZ153I GETVIS FOR AIB FAILED, RETURN CODE = rc

Explanation: The DL/I connector module DLZMPX00 encountered an error while executing a GETVIS request for the Application Interface Block (AIB) on a DL/I call passed via AIBTDLI. The GETVIS return code 'rc' is given in decimal format. The program is canceled.

For an explanation of the GETVIS return code *rc* refer to *VSE/ESA Messages and Codes*, Section "VSE/Advanced Functions Codes and SVC errors".

User Response: Correct the error and rerun the program.

DLZ154I MISSING OR INVALID AIB PARAMETER AT SCHEDULING CALL

Explanation: On a scheduling call entered via AIBTDLI, the mandatory parameter, where DL/I returns the address of the AIB, was missing or had an invalid address. The program is canceled.

User Response: Correct the CALL statement and rerun the program.

DLZ155I WRONG 'PARTID=' OR 'APPLID=' PARAMETER AT SCHEDULING CALL

Explanation: The 'PARTID=' or 'APPLID=' parameter was incorrectly specified on a scheduling call entered via AIBTDLI.

User Response: Correct the CALL statement and rerun the program.

DLZ156I ERROR AT ONLINE PARTITION IN *id*

Explanation: The CICS/DLI online partition identified by *id* has encountered an error while processing a DL/I call passed via AIBTDLI.

User Response: To locate the problem, check the console error messages for this partition, which were displayed before this message.

© Copyright IBM Corp. 2000, 2003

DL/I Messages and Return Codes

These DL/I CICS return codes are new:

Type of Call	AIBFCTR	AIBDLTR	Explanation
Scheduling	08	0A	Roll Back call when not scheduled
Any	FF	00	Unrecoverable error using AIBTDLI interface. See also "How Return Code X'FF00' Is Used" on page 327 for explanation.

Glossary

This glossary defines technical terms and abbreviations used in the VSE/ESA e-business Connectors User's Guide. If you do not find the term you are looking for, view the *IBM Dictionary of Computing* located at: **www.ibm.com/networking/nsg/nsgmain.htm**.

The glossary includes definitions with symbol * where there is a one-to-one copy from the IBM Dictionary of Computing.

***applet.** An application program, written in the Java programming language, that can be retrieved from a Web server and executed by a Web browser. A reference to an applet appears in the markup for a Web page, in the same way that a reference to a graphics file appears; a browser retrieves an applet in the same way that it retrieves a graphics file. For security reasons, an applet's access rights are limited in two ways: the applet cannot access the file system of the client upon which it is executing, and the applet's communication across the network is limited to the server from which it was downloaded. Contrast with *servlet*.

Asymmetric cryptography. Synonymous with *Public key cryptography*.

***authentication.** (1) In computer security, verification of the identity of a user or the user's eligibility to access an object. (2) In computer security, verification that a message has not been altered or corrupted. (3) In computer security, a process used to verify the user of an information system or protected resources.

bytecode. See *Java bytecode*.

CA. See *Certificate Authority*.

CICS ECI. The CICS External Call Interface (ECI) is one possible requester type of the *CICS business logic interface* that is provided by the CICS Transaction Server for VSE/ESA. It is part of the CICS client and allows workstation programs to call CICS functions on the VSE/ESA host.

CICS EPI. The CICS External Presentation Interface (EPI) is part of the CICS client, and enables a non-CICS Client application to act as a logical 3270 terminal and so control a CICS 3270 application.

CICS EXCI. The EXternal CICS Interface (EXCI) is one possible requester type of the *CICS business logic interface* that is provided by the CICS Transaction Server for VSE/ESA. It allows any VSE batch application to call CICS functions.

Common Connector Framework (CCF). Is part of IBM's *Visual Age for Java*, and allows connections to remote hosts to be created and maintained. The CCF classes are contained in the VSEConnector.jar file and are used internally by the VSE Java Beans. CCF is important for multi-tier architectures where, for example, servlets run on a middle-tier platform. Because CCF allows open connections to be kept in a pool, this avoids the time involved in opening and closing TCP/IP connections to the remote VSE/ESA host each time a servlet is invoked.

***Common Object Request Broker Architecture (CORBA).** A specification produced by the Object Management Group (OMG) that presents standards for various types of object request brokers (such as client-resident ORBs, server-based ORBs, system-based ORBs, and library-based ORBs). Implementation of CORBA standards enables object request brokers from different software vendors to interoperate.

ConnectionManager class. Is part of CCF, and identifies the connection to a remote VSE/ESA host: it holds connections between the middle-tier and the remote VSE/ESA server. Servlets can reserve a connection from the pool, work with it and give it back later. This is performed internally using VSE Java Beans.

connector. In the context of VSE/ESA, a connector provides the middleware to connect two platforms: Web Client and VSE/ESA host, middle-tier and VSE/ESA host, or Web Client and middle-tier. These connectors are described in this publication (the CICS Connector, MQSeries Connector, and so on).

container. Is part of the JVM of application servers such as the IBM WebSphere Application Server, and facilitates the implementation of servlets, EJBs, and JSPs, by providing resource and transaction management resources. For example, an EJB developer must not code against the JVM of the application server, but instead against the interface provided by the container. The main role of a container is to act as an intermediary between EJBs and clients, and also to manage multiple EJB instances. After EJBs have been written, they must be stored in a container residing on an application server. The container then manages all threading and client-interaction with the EJBs, and co-ordinate connection- and instance pooling.

cryptographic token. Usually referred to simply as a *token*, this is a device which provides an interface for performing cryptographic functions like generating digital signatures or encrypting data.

© Copyright IBM Corp. 2000, 2003

***cryptography.** (1) The transformation of data to conceal its meaning. (2) In computer security, the principles, means, and methods for encrypting 'plaintext' and decrypting 'ciphertext'.

DB2-Based Connector. Is a feature introduced with VSE/ESA 2.5, which includes a customized DB2 version, together with VSAM and DL/I functionality, to provide access to DB2, VSAM, and DL/I data, using DB2 Stored Procedures.

DB2 Stored Procedure. In the context of VSE/ESA, a DB2 Stored Procedure is a Language Environment (LE) program that accesses DB2 data. However, from VSE/ESA 2.5 onwards you can also access VSAM and DL/I data using a DB2 Stored Procedure. In this way, it is possible to exchange data between VSAM and DB2.

Data Encryption Standard (DES). In computer security, the National Institute of Standards and Technology (NIST) Data Encryption Standard, adopted by the U.S. government as Federal Information Processing Standard (FIPS) Publication 46, which allows only hardware implementations of the data encryption algorithm.

***Decryption.** In computer security, the process of transforming encoded text or ciphertext into plaintext.

DES. See *Data Encryption Standard*.

***digital signature.** In computer security, encrypted data, appended to or part of a message, that enables a recipient to prove the identity of the sender.

Digital Signature Algorithm (DSA). The Digital Signature Algorithm is the U.S. government-defined standard for digital signatures. The DSA digital signature is a pair of large numbers, computed using a set of rules (that is, the DSA) and a set of parameters such that the identity of the signatory and integrity of the data can be verified. The DSA provides the capability to generate and verify signatures.

DSA. See *Digital Signature Algorithm*.

ECI. See *CICS ECI*.

***Encryption.** In computer security, the process of transforming data into an unintelligible form in such a way that the original data either cannot be obtained or can be obtained only by using a decryption process.

Enterprise Java Bean (EJB). An EJB is a distributed Java Bean. "Distributed" means, that one part of an EJB runs inside the JVM of a web application server, while the other part runs inside the JVM of a Web browser. An EJB either represents one data row in a database (entity bean), or a connection to a remote database (session bean). Normally, both types of an EJB work together. This allows to represent and access data in a standardized way in heterogenous environments with relational and non-relational data. See also *Java Bean*.

EPI. See *CICS EPI*.

EXCI. See *CICS EXCI*.

***firewall.** In communication, a functional unit that protects and controls the connection of one network to other networks. The firewall (a) prevents unwanted or unauthorized communication traffic from entering the protected network and (b) allows only selected communication traffic to leave the protected network.

hash function. A hash function is a transformation that takes a variable-size input and returns a fixed-size string, which is called the *hash value*. In cryptography, the hash functions should have some additional properties:

- The hash function should be easy to compute.
- The hash function is one-way; that is, it is impossible to calculate the 'inverse' function.
- The hash function is collision-free; that is, it is impossible that different input leads to the same hash value.

hash value. The fixed-sized string resulting after applying a *hash function* to a text.

home interface. Provides the methods to instantiate a new EJB object, introspect an EJB, and remove an EJB instantiation. As for the remote interface, only an interface is needed because the deployment tool generates the implementation class. Every Session bean's home interface must supply at least one *create()* method.

HTTP Session. In the context of VSE/ESA, identifies the Web-browser client that calls a servlet (in other words, identifies the connection between the client and the middle-tier platform).

***internet.** A wide area network connecting thousands of disparate networks in industry, education, government, and research. The Internet network uses TCP/IP (Transmission Control Protocol/Internet Protocol) as the standard for transmitting information.

***interface definition language (IDL).** In CORBA, a declarative language that is used to describe object interfaces, without regard to object implementation.

JAR. Is a platform-independet file format that aggregates many files into one. Multiple applets and their requisite components (.class files, images, and sounds) can be bundled in a JAR file, and then downloaded to a Web browser using a single HTTP transaction (much improving the download speed). The JAR format also supports compression, which reduces the files size (and further improves the download speed). The compression algorithm used is fully compatible with the ZIP algorithm. The owner of an applet can also digitally sign individual entries in a JAR file, to authenticate their origin.

Java applet. See *applet*.

Java application. A Java program that runs inside the JVM of your Web browser. The program's code resides on a local hard disk or on the LAN. Java applications may be large programs using graphical interfaces. Java applications have unlimited access to all your local resources.

***JavaBeans.** A platform-independent, software component technology for building reusable Java components called "beans." Once built, these beans can be made available for use by other software engineers or can be used in Java applications. Also, using JavaBeans, software engineers can manipulate and assemble beans in a graphical drag-and-drop development environment.

Java bytecode. Bytecode is created when a file containing Java source language statements is compiled. The compiled Java code or "bytecode" is similar to any program module or file that is ready to be executed (run in a computer so that instructions are performed one at a time). However, the instructions in the bytecode are really instructions to the *Java Virtual Machine*. Instead of being interpreted one instruction at a time, bytecode is instead recompiled for each operating-system platform using a just-in-time (JIT) compiler. Usually, this enables the Java program to run faster. Bytecode is contained in binary files that have the suffix **.CLASS**.

***Java Database Connectivity (JDBC).** An application programming interface (API) that has the same characteristics as Open Database Connectivity (ODBC) but is specifically designed for use by Java database applications. Also, for databases that do not have a JDBC driver, JDBC includes a JDBC to ODBC bridge, which is a mechanism for converting JDBC to ODBC; it presents the JDBC API to Java database applications and converts this to ODBC. JDBC was developed by Sun Microsystems, Inc. and various partners and vendors.

***Java Development Kit (JDK).** A software package that can be used to write, compile, debug, and run Java applets and applications.

***Java Runtime Environment (JRE).** A subset of the Java Development Kit (JDK) that contains the core executables and files that constitute the standard Java platform. The JRE includes the Java Virtual Machine, core classes, and supporting files.

***JavaScript.** A scripting language that resembles Java and was developed by Netscape for use with the Netscape browser.

Java Server Page (JSP). A web page, similar to a HTML page. Parts of a JSP are compiled into a servlet by the web server's JSP engine while sending the web page to the requesting browser. JSPs have the advantage, that the developer does not have to recompile the servlet each time it is changed. Changes are always made in the JSP file. A JSP also creates web pages dynamically.

Java servlet. See *servlet*.

***Java Virtual Machine (JVM).** A software implementation of a central processing unit (CPU) that runs compiled Java code (applets and applications).

Lotus Domino Server. The Lotus Notes server which is used for storing each user's *Composite Database*.

***MQSeries.** Pertaining to a family of IBM licensed programs that provide message queuing services.

The VSE/ESA e-business Connectors includes the MQSeries as a connector from a middle-tier platform to a VSE/ESA host.

Multipurpose Internet Mail Extensions (MIME). An Internet standard for identifying the type of object that is transferred across the Internet. MIME types include several variants of audio, graphics, and video.

persistence. A term used to describe the storage of objects in a database to allow them to persist over time, rather than being destroyed when the application containing them terminates. Enterprise Java Bean containers, such as WebSphere, provide persistence services for EJBs deployed within them.

persistent storage. See *Persistence*.

PKCS (Public Key Cryptography Standards). PKCS is a set of standards, issued by RSA Data Security, Inc., for implementation of public key cryptography.

PKI. See *Public key infrastructure*.

***port.** A connector on a device to which cables for other devices such as display stations and printers are attached.

***private key.** In computer security, a key that is known only to the owner. See *Public key cryptography*.

***proxy server.** A server that receives requests intended for another server and that acts on the client's behalf (as the client's proxy) to obtain the requested service. A proxy server is often used when the client and the server are incompatible for direct connection (for example, when the client is unable to meet the security authentication requirements of the server but should be permitted some services).

***public key.** In computer security, a key made available to anyone who wants to encrypt information. See *Public key cryptography*.

***public key cryptography.** In computer security, cryptography in which a public key is used for encryption and a private key is used for decryption. Synonymous with *Asymmetric cryptography*.

remote method invocation (RMI). Is a Java-only version of CORBA. Because RMI is specific to Java, it is easier to use than CORBA. Instead of writing IDL to describe objects, a program called **rmic** can be run on Java class files, which then creates stub and skeleton classes directly from the class files. RMI is used, for example, by EJBs to communicate between the client stub and the server part of the EJB.

remote interface. In the context of VSE/ESA, the remote interface allows a client to make method calls to an EJB although the EJB is on a remote VSE/ESA host. The container uses the remote interface to create client-side stubs and server-side proxy objects to handle incoming method calls from a client to an EJB.

***remote procedure call (RPC).** (1) A facility that a client uses to request the execution of a procedure call from a server. This facility includes a library of procedures and an external data representation. (2) A client request to a service provider located in another node.

RSA algorithm. Public key algorithm named after its inventors Rivest, Shamir, and Adleman.

secret key. Synonymous with *private key*.

Secure Electronic Transaction (SET). An open specification for securing payment card transactions over open networks such as the Internet. SET was developed by Visa and MasterCard with participation from several technology companies, such as GTE, IBM, Microsoft, Netscape, SAIC, Terisa Systems, and VeriSign. SET will be based on encryption technology from RSA Data Security.

Secure Sockets Layer (SSL). A security protocol that allows the client to authenticate the server and all data and requests to be encrypted. SSL was developed by Netscape Communications Corp. and RSA Data Security, Inc..

***servlet.** An application program, written in the Java programming language, that is executed on a Web server. A reference to a servlet appears in the markup for a Web page, in the same way that a reference to a graphics file appears. The Web server executes the servlet and sends the results of the execution (if there are any) to the Web browser. Contrast with *applet*.

SET. See *Secure Electronic Transaction*.

skeleton. In the context of CORBA, is the equivalent of a stub but used on the server (VSE/ESA host). It reassembles the data that arrives from the network into a recognizable format, calls a server-side method, and sends a reply back to the client.

***socks enabled.** Pertaining to TCP/IP software, or to a specific TCP/IP application, that understands the *socks protocol*. "Socksified" is a slang term for socks-enabled.

***socksified.** See *socks enabled*.

***socks protocol.** A protocol that enables an application in a secure network to communicate through a firewall via a *socks server*.

***socks server.** A circuit-level gateway that provides a secure one-way connection through a firewall to server applications in a nonsecure network.

SSL. See *Secure Sockets Layer*.

***thread.** A stream of computer instructions that is in control of a process. A multithreaded process begins with one stream of instructions (one thread) and may later create other instruction streams to perform tasks.

Uniform Resource Locator (URL). (1) A sequence of characters that represent information resources on a computer or in a network such as the Internet. This sequence of characters includes (a) the abbreviated name of the protocol used to access the information resource and (b) the information used by the protocol to locate the information resource. For example, in the context of the Internet, these are abbreviated names of some protocols used to access various information resources: http, ftp, gopher, telnet, and news; and this is the URL for the IBM home page: `http://www.ibm.com`. (2) The address of an item on the World Wide Web. It includes the protocol followed by the fully qualified domain name (sometimes called the host name) and the request. The Web server typically maps the request portion of the URL to a path and file name. For example, if the URL is `http://www.networking.ibm.com/nsg/nsgmain.htm`, the protocol is `http`, the fully qualified domain name is `www.networking.ibm.com`, and the request is `/nsg/nsgmain.htm`.

URL. See *Uniform Resource Locator*.

VSE Connector Server. Is the host part of the VSE Java Beans, and is started using the job STARTVCS which is placed in the VSE/POWER reader queue during installation of VSE/ESA. Runs by default in dynamic class R.

VSE Java Beans. Are Java Beans that allow access to all VSE-based file systems (VSE/VSAM, Librarian, VSE/POWER, and VSE/ICCF), submit jobs, and access the VSE/ESA operator console. The class library is contained in the *VSEConnector.jar* archive. See also *JavaBeans*.

WebSphere Application Server. Is used together with an IBM HTTP Server to process servlets, JSPs, and EJBs. Also requires a specific JDK and DB2.

Index

Special characters

$SOCKOPT phase 98

Numerics

2-tier environments
  description 12
  example of use 12
  overview diagram 12
  using applets 202
  using CICS connectivity 17
  using Java-based connector 17
  using VSAM Redirector connector 17
3-tier environments
  description 13
  example of use 13
  MQSeries connectivity 18
  overview diagram 13
  programs installed on middle-tier 22
  storing session information 245
  using applets 203
  using CICS connectivity 18
  using DB2-based connector 18
  using EJBs 273
  using Java-based connector 18
  using JSPs 263
  using servlets 243

A

Access Control Function of BSM 103
accessing
  DL/I data using AIBTDLI interface 321
  DL/I data using DB2 Stored Procedure 320
  DL/I data using VSE Java Beans 167
  ICCF data using VSE Java Beans 177
  Librarian using VSE Java Beans 175
  operator console using VSE Java Beans 180
  POWER data using VSE Java Beans 172
  VSAM data using DB2 Stored Procedure 312
  VSAM data using VSE Java Beans 164
  VSE data using applets 201
  VSE data using JSPs 263
  VSE data using servlets 243
  VSE data using VSE Java Beans 153
  VSE data via VSE Navigator 182
action codes for VSE Connector Server
  PLGACT_CHECKCANCEL 300
  PLGACT_FINISH 300
  PLGACT_NOTHING 300
  PLGACT_RECEIVE 300
  PLGACT_SEND 300
  PLGACT_SEND_RESP 300
  PLGACT_WAIT 300
action codes for VSE Connector Server (continued)
  PLGACT_WAITRECV 300
  PLGACT_WAITTIMER 301
  supported by VSE Connector Server 300
AIBTDLI
  creating programs that use 323
  format of database and checkpoint call 325
  format of Roll Back call 325
  format of scheduling call 324
  format of termination call 325
  invoking 324
AIBTDLI callable interface 321
AIBTDLI Interface
  and return code X'FF00' 327
  callable interface AIBTDLI 321
  compiling & link-editing programs 326
  error messages description 329
  errors with no return code 327
  for accessing DL/I data 321
  format of AIB 326
  new messages generated by 365
  partition layout 322
  return & status codes 326
  single / multiple MPS systems 328
  task termination & abend handling 329
Apache Server 22
APAR PQ39683 41, 71
applets
  disadvantages & restrictions of 206
  for accessing DL/I data via a DB2 Stored Procedure 230
  for accessing VSAM data via a DB2 Stored Procedure 216
  for defining a map 207
  in 2-tier environments 202
  in 3-tier environments 203
  sample applet 201
  using VSEAppletServer 205
  VSAM data mapping example, description 207
  VsamSpaceUsage 207
AppletViewer 215
Application Framework for e-business
  and core applications 3
  and Secure Electronic Transaction (SET) 3
  and Secure Sockets Layer (SSL) 3
  extending/levering core applications 3
  purpose 3
Application Interface Block (DLIAIB) 326
application server 22
archive tag 206
ASCII, considerations in plugins 306

B

Basic Security Manager 103
BSSDCERT service function
  and Client-Certificates/User-IDs dialog 125
  building mapping list of client-certificate/User-ID pairs 123
  change library and member-names defaults 125

C

callback mechanism, of VSE Java Beans 159
cci.jar 146
CEDA transaction, for defining SOAP server 346
CEDA transaction, for defining SOAP service 345
CEDA, using for SOAP sample 345
certificates
  cataloging server and root 92
  cataloging server certificate 92
  root 101
  server 98
  verifying 103
CIALCLNT utility, installing 95
CIALCLNT utility, starting on Web client 96
CIALCREQ utility 97
CIALSIGV utility 103
CIALSRVR utility 95
CICS connectivity
  what it offers 9
CICS Transaction Gateway
  using for CICS connectivity in 3-tier env. 9
CICS Transaction Server
  and ECIRequest class 9
  and EPIRequest class 9
  and the CICS Universal Client 9
  and the JavaGateway 9
  customizing 72
  obtaining root certificate 101
  the VSAM-via-CICS service 87
  using for CICS connectivity in 3-tier env. 9
CICS Universal Client 9
  using for CICS connectivity in 3-tier env. 9
CIPHERSUITES 109
CleanupHandler function 299
CleanupPlugin function 294
CLI (Call Level Interface) 311, 314
CLI interface (for C programs)
  activities on the requestor 314
  activities on the VSE/ESA host 315
  example of syntax (VSAMSQLCloseTable) 316
  program flow when using 317

© Copyright IBM Corp. 2000, 2003

CLI interface (for C programs) *(continued)*
supported functions for accessing mapped VSAM data 315
supported SQL statements 318
client authentication
and Client-Certificates/User-IDs dialog 125
batch service function BSSDCERT 123
configuring Java-based connector 115
configuring VSE Connector Client 117
configuring VSE Connector Server 116
service functions for 123
client certificate
generating, and storing in client keyring file 118
Client Configuration Assistant (CCA) 85
client-certificate/User-ID pairs, mapping list 123
Client-Certificates/User-IDs dialog
creating the output job 127
selecting an option 126
starting 125
submitting/ storing the output job 128
codebase tag 206
configuration PHASE for VSAM Redirector Connector 58
connection possibilities under VSE/ESA
2- and 3-tier environments 11
choosing connectors in 3-tier env. 18
downloading client-part 25
downloading host-part 60
installing VSE Connector Client 25
overview of what connectors can be used for 17
products it supports 3
uninstalling VSE Connector Client 29
connections properties file, example for VSE Script Server 356
Connections.properties 46, 356
console functions for VSE Script Language 353
container (EJB) 267, 268
CORBA 270
CRYPTO.KEYRING library 92

D

data-mapping applet (sample)
activities required on Host 208
calling 209
creating file index.html 208
deploying 209
description 207
for adding map to VSAM cluster 211
for modifying a map 212
for modifying data fields of map 213
initializing 210
running using AppletViewer 215
setting up the class 210
DATAEX 305
DB2 Connect
used for accessing VSAM data 313
DB2 Connect, install / configure 85
DB2 Server for VSE 310
DB2 Stored Procedures
accessing DL/I data via 320
accessing VSAM data via 313
advantages 310
for accessing VSAM data 312
how you use 310
interfaces you can use 311
programming requirements for 311
DB2-based connector
customizing 72
define sample database 72
description 6, 309
using in 3-tier env. 18
DB2ConnectorJDBCApplet.java 223
DB2CTVAR job, catalog ARISIVAR.Z 73
DB2DEFCT job, define user catalog 73
DB2DLIConnectorJDBCApplet.java 235
DB2DRDA job, activate DRDA support 75
DB2DRDA job, set up for DRDA support 80
DB2Handler, sample VSAM request handler 64
DB2JMGR, job manager 74
DB2SPSCA job, set up Stored Procedure Server, define to DB2 81
db2vsewm database-alias 217
digital signature 98
Distributed Relational Database Architecture
used with DB2-based connector 6
DL/I applet (sample)
and DLIREAD (sample DB2 Stored Procedure) 238
calling 233
client-side program 235
compiling DB2 Stored Procedure for 232
creating JAR file for 233
defining DB2 Stored Procedure for DB2 Server for VSE 232
defining DL/I database for 233
HTML file for calling 232
main window for 234
prerequisite steps for 232
DL/I CICS return codes 365
DL/I data
accessing using AIBTDLI interface 321
accessing using DB2 Stored Procedures 320
accessing using VSE Java Beans 167
configuring to access DL/I data via VSE Java Beans 41
DL/I database, for the sample DL/I applet 233
DL/I messages (for AIBTDLI interface) 365
DL/I return codes (for AIBTDLI interface) 365
DLIREAD, compiling 232
DLIREAD, how used 230
DLZBSEOT (task termination exit) 41, 71
DLZLX000 323
DLZMPX00 (AIBTDLI interface) 321
DR2JMGR job, install DB2 sample database 77
DR2JMGR job, prepare DB2 sample DB 75
DRDA application 310
DTSECTAB table 103

E

e-business applications
description 4
EBCDIC, considerations in plugins 306
ECI Interface 9
ECIRequest 9
Enterprise Java Bean (EJB)
accessing from an EJB client 283
and CORBA 270
and EJB container 267
architecture, overview 267
as entity beans 267, 268
as session beans 267, 268
compared to JavaBeans and servlets 269
deploying 282
description 267
entities involved in EJB method call 270
example of implementing 274
for implementing client applications 270
representing VSE data using 267
sample, compile Java source files 282
sample, create record layout for employees 275
sample, defining VSAM cluster for 275
sample, implement EJB code 277
sample, implement RecordPK class 277
sample, specify EJB home interface 276
sample, specify EJB remote interface 276
used in 3-tier environment 273
entity bean 154
entity beans, properties 267, 268
EPI Interface 9
EPIRequest 9
error messages and codes
for AIBTDLI interface 365
ExecuteHandler function 297

G

general functions for VSE Script Language 352
getdata.src (sample VSE Script) 357
GetHandledCommands function 295
getquote.c 344
getquote.c (SOAP service) 339
GetQuote.java 344
GetQuote.java (SOAP Java client) 343
glossary 367
grouping 310

H

home interface, of EJB 276
homepage
  DB2 85
  for downloading Java code 22
  for downloading VSAM Redirector Server 60
  for downloading VSE Connector Client 25
  for downloading VSE Script Server 43
  for obtaining KeyMan tool 117
  for WebSphere Application Server 10
  Thawte Corporation 102
  VSE xix
homepage, VSE xix
HTML file for calling the DL/I applet 232
HTML file for calling the VSAM applet 218
HTMLHandler, IBM-supplied example
  cleanup method 67
  close method 67
  code for 65
  console and output 68
  data layout used with 66
  description 65
  finish method 68
  implementation of 66
  initialize method 66
  open method 67
  request method 67
HTTPServletRequest 245
httpSession 245

I

IBM HTTP Server, installing 22
ibmjsse.jar 147
ICCF data
  accessing using VSE Java Beans 177
iesincon.w 25
IESMAPD 131
IESPLGIN.H 302
IESPLGSK.C 302
iesscrpt.w, obtaining 43
IESSOAPD (SOAP decoder) 333, 350
IESSOAPE (SOAP encoder) 334
IESSOAPH.H header file for SOAP 335
init() method, example 224
installing VSE Connector Client 25
installing/ configuring your system
  choosing connectors in 3-tier env. 18
  CIALCLNT utility 95
  CIALCREQ utility 97
  CIALSIGV utility 103
  CIALSRVR utility 95
  configuring DL/I for access via VSE Java Beans 41
  configuring TCP/IP 21
  configuring the VSAM-via-CICS service 87
  configuring the VSE Connector Server 30
  connectivity possibilities 17
  for 2 and 3-tier environments 11
installing/ configuring your system *(continued)*
  installing Java 21
  installing prerequisite programs 21
  installing the VSE Script Server 43
  installing VSAM Redirector connector 47
  installing VSAM Redirector Server 60
  installing VSE Connector Client 25
  installing VSE Navigator 183
  installing WebSphere Application Server 23
  programs installed on middle-tier 22
  SKSSLKEY Job 92
  the VSE HTTP Server 21
  using the Java-based connector (2-tier env.) 17
  VSAM Redirector Client 50
  VSE Connector Client for SSL 108
  VSE Connector Server for server authentication 105
Internet address
  for WebSphere Application Server 10
  VSE homepage xix

J

J2EE package, installing 344, 346
JAR file, for DL/I applet 233
JAR file, for VSAM applet 221
Java
  choosing between JDK and JRE 22
  downloading the Java base code 22
  downloading the Java Development Kit 22
  installing & configuring 21
  installing JDK 21
  installing Swing classes 22
  SOAP client, compile 346
  SOAP client, description 343
  SOAP client, running 347
  SOAP-client packages, downloading 344
Java Development Kit (JDK)
  installing 21
Java Runtime Environment (JRE) 22
Java Server Page (JSP)
  accessing VSE data using 263
  advantages in using 265
  example of 265
Java-based connector
  description 4
  extending, with plugins 287
  overview of client-part 4
  overview of server-part 5
  protocol used by 305
  using in 2-tier env. 17
  using in 3-tier env. 18
  what it is 4
JavaBeans
  compared to EJBs 269
Javadoc, example for VSE Java Bean 158
JavaGateway 9
JDBC (Java database connectivity) 314
JDBC (Java Database Connectivity)
  advantages of 187
JDBC (Java Database Connectivity) *(continued)*
  example of use 191
  specifying table names 190
  SQL statements supported 187
JDBC classes, importing 223
JDBC driver class, loading 224
jobs supplied
  DB2CTVAR (catalog ARISIVAR.Z) 73
  DB2DEFCT (define user catalog) 73
  DB2DRDA (activate DRDA support) 75
  DB2DRDA (set up for DRDA support) 80
  DB2JMGR (job manager) 74
  DB2SPSCA (set up Stored Procedure Server, define to DB2) 81
  DR2JMGR (install DB2 sample database) 77
  DR2JMGR (prepare DB2 sample DB) 75
  PSERVER 75

K

key pair, generating 96, 117
KeyMan tool (IBM) 117
keyring file (KeyRing.pfx)
  storing client certificate in 119
  used with VSE Connector Clients 111
KeyRing.pfx file 111
KEYRINGFILE 109
KEYRINGPWD 110

L

Librarian
  accessing using VSE Java Beans 175
Lotus 1-2-3 spreadsheet file, sample 359
Lotus 1-2-3, example using a VSE Script Client 356
Lotus Domino 3
Lotus Domino Go Server 22

M

map, creating 129
mapping list, of client-certificate/User-ID pairs 123
mapping VSAM data
  creating a map 129
  creating a view 129
  why you need to do so 129
maps
  creating using the VSAM MapTool 140
  defining 132
  defining using a Java application 134
  defining using RECMAP 132
  defining using sample applet 134
  example of creating a local VSAM map object 135
  example of creating a view for a map 137
  example of creating data fields for a map 136

maps *(continued)*
example of displaying properties of a map 136
example of how to delete 138
storing on VSE/ESA host 130
structure of 130
MessageDialog.java 223
middle-tier
configuring for SSL 108
programs you install 22
MPS systems 328
MQSeries connectivity
using in 3-tier env. 18
what it offers 9
MS Office spreadsheet, sample 362
MS Office, example using a VSE Script Client 356

N

non-Java access using VSE Script Connector 349

O

ODBC (Open DataBase Connectivity) 311, 314
online documentation, description 28
operator console (VSE)
accessing using VSE Java Beans 180

P

PLGACT_CHECKCANCEL action code 300
PLGACT_FINISH action code 300
PLGACT_NOTHING action code 300
PLGACT_RECEIVE action code 300
PLGACT_SEND action code 300
PLGACT_SEND_RESP action code 300
PLGACT_WAIT action code 300
PLGACT_WAITRECV action code 300
PLGACT_WAITTIMER action code 301
plugin functions
CleanupHandler 299
CleanupPlugin 294
ExecuteHandler 297
GetHandledCommands 295
PluginMainEntryPoint 292
SetupHandler 296
SetupPlugin 293
PluginMainEntryPoint function 292
plugins
and protocol for transferring data 307
ASCII / EBCDIC considerations 306
Big / Little Endian considerations 306
choosing access method for 306
compiling 302
design considerations 305
implementing server plugin 288
structure of 287
structuring client-part view 307
POWER functions for VSE Script Language 353
PowerGridLayout class 210
PowerGridLayout.java 223
private key
cataloging 92
generating 96
properties file for VSAM Redirector Server 61
properties file, example for VSE Script Server 356
protocol
extending with own commands 305
used by Java-based connector 305
protocol, between VSE Script Client / VSE Script Server 350
protocol, for communicating between client and server 307
PSERVER job 75
public key
cataloging on a certificate 92

R

RECMAP command 132
RecordPK class 277
redbooks, to which can refer xix
remote interface, of EJB 276
rename() method 212
return code X'FF00' 327
return codes
DL/I CICS 365
root certificate
description 101
importing into client keyring file 120
obtaining 101
Thawte 120
RSA key pair
generating 96

S

Sample script, running from command line 364
sample VSE Script (getdata.src) 357
samples provided with VSE/ESA e-business connectors
applet 201
applet for VSAM data mapping 207
DL/I applet 230
EJB 275
servlet 246
VSAM applet 216
Secure Electronic Transaction (SET) 3
glossary description 370
supported by Application Framework for e-business 3
Secure Sockets Layer (SSL)
activate SSL profile for VSE Connector Server 106
and Java properties file 109
client keyring file on Web clients or middle-tier 111
configuring VSE Connector Client 108
configuring VSE Connector Server for server authentication 105
flag in VSEConnectionSpec class 108
Secure Sockets Layer (SSL) *(continued)*
glossary description 370
installing / activating 92, 95
profile for VSE Connector Server 105
root certificate 101
supported by Application Framework for e-business 3
server
VSEAppletServer 205
server certificate
copying to client keyring file on Web client or middle-tier 110
used with VSE Connector Client's client keyring file 110
server certificate, cataloging 92
server certificate, signing 98
service functions for client authentication 123
servlet
accessing VSE data using 243
compared to EJBs 269
compiling and calling 245
in 3-tier environments 243, 245
sample servlet, create new flight 259
sample servlet, create new order 260
sample servlet, creating VSAM clusters for 247
sample servlet, description 246
sample servlet, display flight properties 253
sample servlet, displaying list of flights 250
sample servlet, get flight instances 251
sample servlet, HTML constructs used 247
sample servlet, place an order 256
sample servlet, using forms to get input 248
session bean 154
session beans, properties 267, 268
session information, storing 245
SetupHandler function 296
SetupPlugin function 293
SKCPSTP, compile DB2 Stored Procedures 83
SKCRESTP, create DB2 Stored Procedures 83
SKDB2SPS, catalog startup job SPSERV01 81
SKDB2STR, put DB2START in POWER reader 84
SKDB2VAR, customize DB2-based connector 72
SKDLICMP, compile/link COBOL stored procedures 83
SKDLISMP (initialize DL/I sample database) 42, 83
SKDLISTP, create stored procedures for DL/I access 83
skeletons
CIALCREQ.JCL 97
CIALSIGV.JCL 103
CIALSRVR.JCL 95
SKCPSTP (compile DB2 Stored Procedures) 83

skeletons *(continued)*
SKCRESTP (create DB2 Stored Procedures) 83
SKDB2SPS (catalog startup job SPSERV01) 81
SKDB2STR (put DB2START in POWER reader) 84
SKDB2VAR (customize DB2-based connector) 72
SKDLICMP (compile/link COBOL stored procedures) 83
SKDLISMP, initialize DL/I sample database 42, 83
SKDLISTP (create stored procedures for DL/I access) 83
SKSSLKEY.JCL 92
SKVCSSSL (configure VSE Connector Server for client authentication) 116
SKVSSAMP (define VSE/VSAM cluster, load data) 83
SKJOURN, and CICS TS 72
SKSSLKEY, job to catalog a keyring set 92
SKVCSCAT, VSE Connector Server catalog members job 31
SKVCSCFG skeleton 106
SKVCSCFG, specify general settings for VSE Connector Server 32
SKVCSLIB, specify libraries for VSE Connector Server 33
SKVCSPLG, specify plugins for VSE Connector Server 33
SKVCSSL skeleton 105
SKVCSSSL skeleton (configure VSE Connector Server for client authentication) 116
SKVCSSSL, configure VSE Connector Server for SSL 35
SKVCSSTJ, for placing startup job in reader queue 30
SKVCSUSR, and VSE Connector Server 38
SKVCSUSR, specify logon access for VSE Connector Server 34
SKVSSAMP job 220
SKVSSAMP sample skeleton 357
SKVSSAMP, define VSE/VSAM cluster, load data 83
SOAP (Simple Object Access Protocol)
and COMMAREA 333, 334, 350
client (IBM-supplied), code for 341
compile /link sample C programs 345
control blocks 335
converter 333, 334, 350
decoder (IESSOAPD) 333, 350
define SOAP server to CICS (CEDA) 346
description 8
encoder (IESSOAPE) 334
general syntax 331
header file IESSOAPH 335
Java client (IBM-modified), code for 343
Java SOAP-client packages 344
running IBM-supplied sample 344
service (IBM-supplied), code for 339
SOAP (Simple Object Access Protocol) *(continued)*
VSE/ESA host acting as SOAP client 334
VSE/ESA host acting as SOAP server 332
writing own programs 348
SOAP client 334
SOAP client (C-program), running 347
SOAP client (Java), compile 346
SOAP client (Java), running 347
SOAP server 332
SOAP_DEC_PARAM control block 338
SOAP_PARAM_HDR control block 335
SOAP_PROG_PARAM control block 337
soapclnt.c 344
soapclnt.c (SOAP client) 341
SQL
comparison with VSE Java Beans terminology 190
statements supported by JDBC 187
statements supported by VSAMSQL CLI 318
sqlds database 216
SSLVERSION 109
STARTVCS job 36
Stored Procedure Server description 310
string functions for VSE Script Language 353
SUN J2EE package, installing 344

T

TCP/IP
configuring on the VSE/ESA host 21
customizing 72
Thawte Corporation 102

U

uninstalling VSE Connector Client 29

V

view, creating 129
views
example of adding data fields 137
example of creating a view for a map 137
example of displaying properties 138
VisualAge for Java 3
VSAM applet (sample)
calling 221
client-side program 223
compiling DB2 Stored Procedure for 218
creating JAR file for 221
defining DB2 Stored Procedure for DB2 Server for VSE 219
defining VSAM data cluster for 219
general description 216, 230
HTML file for calling the applet 218
main window for 221
prerequisite steps for 218
VSAM applet (sample) *(continued)*
server-side DB2 Stored Procedure 226
VSAM data
accessing using DB2 Stored Procedure 312
accessing using JDBC 187
accessing using VSE Java Beans 164
accessing via DB2 Stored Procedures 313
creating maps using the VSAM MapTool 140
defining maps 132
defining maps using a Java application 134
defining maps using RECMAP 132
defining maps using sample applet 134
example of adding data fields to a view 137
example of creating a local VSAM map object 135
example of creating a view for a map 137
example of creating data fields for a map 136
example of displaying properties of a map 136
example of displaying properties of a view 138
program flow when using CLI 317
storing maps on VSE/ESA host 130
structure of VSAM maps 130
supported CLI functions for accessing mapped data 315
VSAM data cluster, for the sample VSAM applet 219
VSAM functions for VSE Script Language 354
VSAM MapTool 140
VSAM Redirector Client
and VSAM logic 48
installing & configuring 50
overview 47
VSAM Redirector connector
diagram of how it works 47
downloading VSAM Redirector Server from Internet 60
installing / configuring VSAM Redirector Client 50
installing & implementing 47
installing VSAM Redirector Server 60
obtaining VSAM Redirector Server from PRD1.BASE 60
overview 6
overview diagram 47
prerequisites for installing 60
VSAM Redirector Server
error reporting 63
installing & implementing 60
obtaining a copy 60
overview 47
properties file for 61
VSAM request handler
calling 63
coding VSAM logic 62

VSAM request handler *(continued)*
datatype conversions 63
DB2Handler sample 64
error reporting 63
getting a map dynamically 64
htmlhandler sample 65
implementing 62
overview 47
overview diagram of where used 47
VSAM-via-CICS service
CICS transactions for use with 89
configuring CICS for 87
description 87
how it works 89
VSAM.RECORD.MAPPING.DEFS 131
VSAM.VSESP.USER.CATALOG 247
VsamDataMapping.java 134
VsamMappingApplet.html 134
VSAMSEL sample Stored Procedure 226
VsamSpaceUsage 207
VSAMSQL CLI environment, de-allocating 229
VSAMSQL CLI environment, initializing 226
VSAMSQL prefix 315
VSAMSQLBindCol(), example 228
VSAMSQLBindParameter(), example 227
VSAMSQLCloseTable - close a specified table (cluster) 316
VSAMSQLExecute(), example 227
VSAMSQLFetch(), example 228
VSAMSQLPrepare(), example 227
VSE Connector Client
and copy of server certificate 110
and Java properties file for SSL profile 109
configuring for client authentication 115, 117
configuring for SSL 108
description 5
downloading from Internet 25
installing 25
installing online documentation component 27
installing samples component 27
installing VSE Java Beans component 27
obtaining from PRD1.BASE 26
prerequisites for installing 25
SSL flag in VSEConnectionSpec 108
uninstalling 29
using client keyring file 111
VSE Connector Server
action codes supported by 300
activate & cataloging SSL profile 106
and security 38
configuration file 106
configuring for client authentication 116
configuring for SSL 105
configuring on the VSE/ESA host 30
configuring SSL profile 105
entering a command for 38
glossary description 370
job SKVCSCAT (catalog members) 31
VSE Connector Server *(continued)*
job SKVCSSTJ (placing startup job in reader queue) 30
listing possible commands for 38
obtaining root certificate 101
overview 5
skeleton SKVCSCFG (specify general settings) 32
skeleton SKVCSLIB (specify libraries) 33
skeleton SKVCSPLG (specify plugins) 33
skeleton SKVCSSSL (configure for SSL) 35
skeleton SKVCSUSR (specify logon access) 34
starting 36
testing communication with VSE Connector Client 37
VSE data
accessing using applets 201
accessing using JSPs 263
accessing using servlets 243
representing using EJBs 267
VSE HTTP Server
configuring 21
VSE HTTP Server, configuring 21
VSE Java Beans
accessing DL/I data using 167
accessing ICCF data using 177
accessing Librarian using 175
accessing operator console using 180
accessing POWER data using 172
accessing VSAM data using 164
and the VSE Navigator application 182
contents of class library 155
glossary description 370
how compare to EJBs 154
how compare to JavaBeans 154
installing 27
submitting jobs using 173
using for connecting to VSE/ESA host 162
using in 3-tier env. 18
using in 3-tier environments 153
using the callback mechanism 159
what they are 154
VSE Java Beans, installing 27
VSE Keyring Library, installing 92
VSE Keyring Library, securing via BSM 103
VSE Navigator
and the PowerGridLayout class 210
description 182
VSE Script Client
description 7
example 356
sample files you can use 355
VSE Script connector
diagram of how it is used 349
overview 7
protocol used between client and server 350
VSE Script Language 351
VSE Script Language
built-in console functions 353
VSE Script Language *(continued)*
built-in functions 352
built-in POWER functions 353
built-in string functions 353
built-in VSAM functions 354
description 351
general rules 351
VSE Script sample (getdata.src) 357
VSE Script Server
description 7
installing 43
obtaining from PRD1.BASE 43
obtaining via the internet 43
performing the installation 44
prerequisites for installing 43
properties file Connections 46
properties file VSEScriptServer 45
starting locally 358
VSE Scripts, writing 351
VSE Security Manager 39
VSE/ESA host database, establishing connection to 224
VSE/POWER data
accessing using VSE Java Beans 172
VSEAppletServer, how used 205
VSECertificateEvent 155
VSECertificateListener 155
VSEConnectionManager 155
VSEConnectionSpec 245
VSEConnectionSpec class 108, 155
VSEConnector.jar 27, 61
VSEConnectorTrace 155
VSEConsole class 155
VSEConsoleExplanation class 155
VSEConsoleMessage class 155
VSEDli class 155
VSEDliPcb class 155
VSEDliPsb class 155
VSEIccf class 155
VSEIccfLibrary class 155
VSEIccfMember class 155
VSELibrarian class 155
VSELibrary class 156
VSELibraryExtent class 156
VSELibraryMember class 156
VSEMessage class 156
VSEPlugin class 156
VSEPower class 156
VSEPowerEntry class 156
VSEPowerQueue class 156
VSEResource class 156
VSEResourceEvent class 156
VSEResourceListener class 156
VSEResourceListener, example 160
VSEScriptServer.properties 45, 356
VSESubLibrary class 156
VSESystem class 156
VSEUser class 156
VSEVsam class 156
VSEVsamCatalog class 156
VSEVsamCluster class 156
VSEVsamField class 156
VSEVsamFilter class 157
VSEVsamMap class 157
VSEVsamRecord class 157
VSEVsamView class 157

W

Web Servers
  Apache 22
  IBM HTTP Server 22
  Lotus Domino Go 22
WebSphere Application Server
  and EJB containers 268
  complementary programs 10
  glossary description 370
  installing on Windows, AIX, Sun Solaris 23
  installing on z/OS 23
  Internet address 10
  managing EJBs 267
  overview 9
  standard, advanced, & enterprise editions 23
  storing session information 245
  used for accessing VSAM data 313

X

XML parser 333, 334

Readers' Comments — We'd Like to Hear from You

IBM VSE/Enterprise Systems Architecture
VSE/ESA e-business Connectors
User's Guide
Version 2 Release 7

Publication No. SC33-6719-04

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Name _____ Address _____

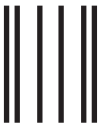
Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SC33-6719-04

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Deutschland Entwicklung GmbH
Department 3248
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or Fold
Along Line

File Number: S370/S390-34
Program Number: 5690-VSE

Printed in U.S.A.

SC33-6719-04

Spine information:

VSE/ESA e-business Connectors, User's Guide Version 2 Release 7 SC33-6719-04