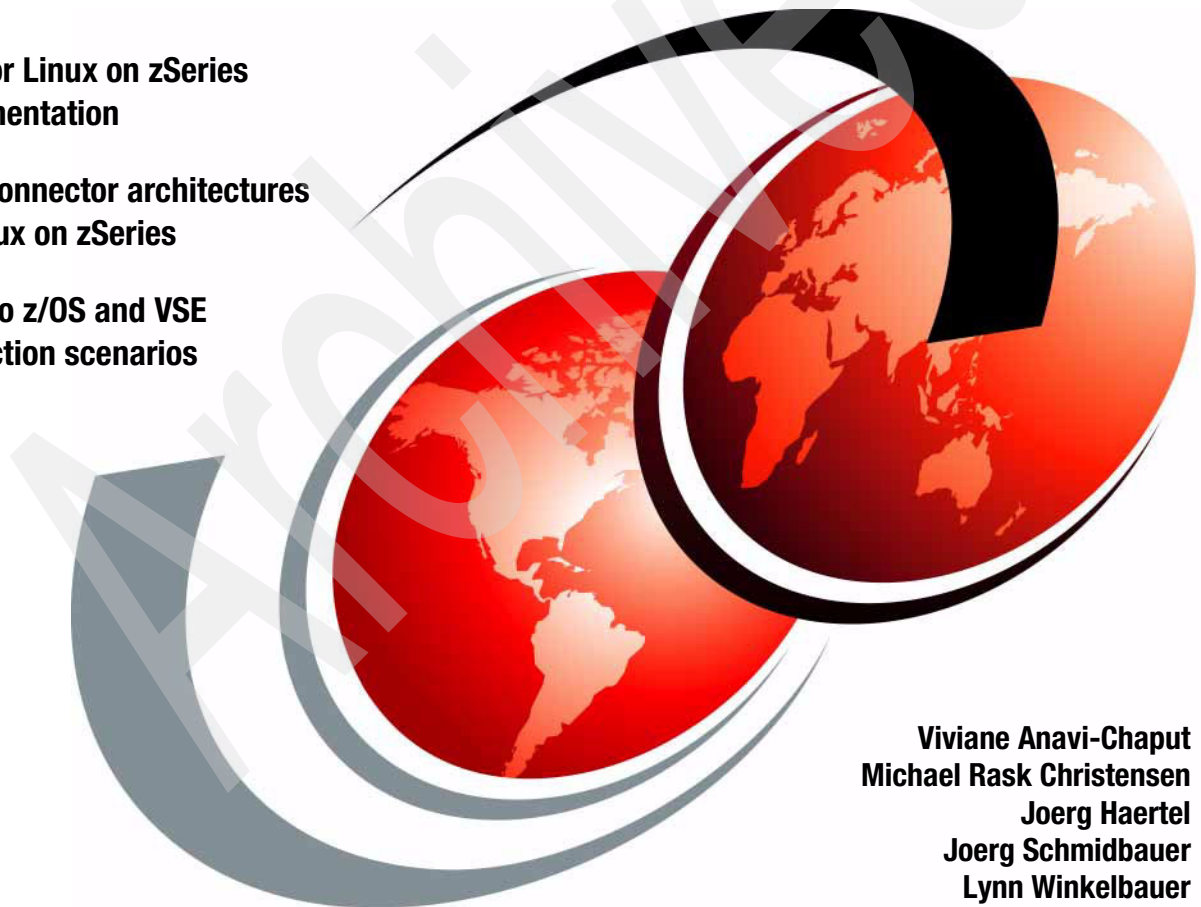


WebSphere V5 for Linux on zSeries Connectivity Handbook

WAS for Linux on zSeries
implementation

J2EE Connector architectures
for Linux on zSeries

Linux to z/OS and VSE
connection scenarios



Viviane Anavi-Chaput
Michael Rask Christensen
Joerg Haertel
Joerg Schmidbauer
Lynn Winkelbauer



International Technical Support Organization

**WebSphere V5 for Linux on zSeries Connectivity
Handbook**

June 2004

Archived

Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

First Edition (June 2004)

This edition applies to WebSphere for Linux on zSeries Version 5.0.2.

© Copyright International Business Machines Corporation 2004. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Noticesxi
Trademarks	xii
Preface	xiii
The team that wrote this redbook	xiii
Become a published author	xiv
Comments welcome	xv
Chapter 1. J2EE connector architecture overview	1
1.1 J2C overview	2
1.1.1 Resource adapter	2
1.1.2 Common client interface	2
1.1.3 System contracts	3
1.1.4 Application contracts	5
1.1.5 Deployment and packaging protocols	5
1.2 System contracts	5
1.3 Connection management	5
1.3.1 Connection pooling	5
1.3.2 Managed access to the resource adapter	6
1.3.3 Non-managed access to the resource adapter	8
1.4 Transaction management	9
1.4.1 Global transactions	11
1.4.2 Local transactions	12
1.4.3 One-phase commit optimization	13
1.4.4 Local transactions versus one-phase commit optimization	13
1.5 Security management	13
1.5.1 Terminology	13
1.5.2 Security model overview	14
1.6 Common client interface	17
1.6.1 Enterprise application integration	17
1.7 Deployment and packaging	19
1.7.1 Packaging of the resource adapter module	20
1.7.2 Deployment descriptor	20
1.7.3 Deploying the resource adapter	22
Chapter 2. Introducing the connectors test environment	25
2.1 System configurations	26
2.1.1 Hardware	27
2.1.2 Software	27

2.2	Frontend environment	28
2.3	Backend environments	29
2.3.1	z/OS	29
2.3.2	VSE	29
2.4	Connector scenarios overview	29
2.4.1	The Trader application scenarios	31
Chapter 3. The Trader applications		33
3.1	Introducing the Trader application	34
3.1.1	Trader application components	34
3.1.2	Trader datastores	35
3.1.3	Trader frontend GUI	36
3.1.4	Trader Web frontend architecture	38
3.1.5	Packaging	40
3.1.6	Dependencies	41
3.1.7	Trader connector paths	42
3.1.8	Downloading Trader application modules	45
3.2	Running Trader in WSAD.IE test environment	45
3.2.1	Configuring the test server in WSAD.IE	47
3.2.2	Runing Trader application in WSAD.IE test environment	56
3.2.3	Configuring DB2 tables in the test environment	58
3.2.4	Running Trader application in WSAD.IE test environment	59
3.3	Deploying Trader application	59
3.3.1	Installing Trader application	60
3.3.2	Running Trader application	60
3.4	Trader DB2 table definitions	63
3.5	Trader VSAM file definitions	65
3.6	Trader CICS Transaction Gateway usage	66
3.7	WebSphere Studio Integration Edition hints and tips	66
Chapter 4. WebSphere Application Server setup		67
4.1	Planning for WebSphere Application Server setup	68
4.1.1	Space requirements for installation	68
4.1.2	WebSphere MQ client for Linux on zSeries	68
4.2	Installing WebSphere MQ Client for Linux on zSeries	69
4.2.1	Verifying the MQ Client installation	70
4.2.2	Testing the MQ installation	72
4.3	Installing WebSphere Application Server	72
4.3.1	Installing with the installation wizard GUI	73
4.3.2	Getting started	78
Chapter 5. CICS J2EE connectors		83
5.1	Overview of the test environment	84
5.1.1	CICS connection to z/OS	84

5.1.2	CICS connection to VSE	85
5.2	Setting up the CTG on Linux for zSeries.	86
5.2.1	Installing the CTG on Linux for zSeries.	86
5.2.2	Configuring the CTG on Linux for zSeries	88
5.3	Setting up the CICS regions on z/OS and VSE.	96
5.3.1	Setting up the CICS TS for z/OS environment	96
5.3.2	Setting up the CICS TS for VSE environment.	98
5.3.3	Testing the CTG to VSE CICS/TS connectivity.	100
5.4	Testing the connectivity from Linux for zSeries.	103
5.4.1	Testing the connections	103
5.4.2	Creating a simple TestECI program on Linux for zSeries.	103
5.5	Configuring WebSphere for CICS connections	105
5.5.1	Installing the resource adapter	105
5.5.2	Configuring a J2C connection factory for z/OS.	108
5.5.3	Configuring a J2C connection factory for VSE	112
5.5.4	Deploying the application in WebSphere	114
5.5.5	Implementing application security in WebSphere	115
5.6	Problem determination	116
5.6.1	Common errors	116
 Chapter 6. Using SOAP to communicate with CICS.		119
6.1	SOAP overview	120
6.1.1	SOAP on z/OS	121
6.1.2	SOAP on VSE.	121
6.2	Configuring CICS on VSE for SOAP support	122
6.2.1	Step 1: Specify TCP/IP=YES in CICS setup.	122
6.2.2	Step 2: Define the symbolic name of VSE to TCP/IP	123
6.2.3	Step 3: Define the TCP/IP service	123
6.2.4	Step 4: Activate the ASCII to EBCDIC converter	124
6.3	Compiling the SOAP service on VSE	124
6.4	Testing the SOAP communication	126
6.4.1	Software prerequisites for the Java SOAP client	127
6.4.2	Implementing a Java-based SOAP client	127
6.4.3	Running the Java-based SOAP client.	128
6.5	Writing your own SOAP programs on VSE	129
6.6	Considerations for using SOAP in WebSphere.	129
 Chapter 7. DB2 connectors		131
7.1	DB2 Connect scenario	132
7.2	Installing DB2 Connect V8.1	132
7.2.1	Simple connect to DB2 for z/OS	142
7.2.2	Simple connect to DB2 for VSE	143
7.3	Customizing WebSphere Application Server for DB2 Connect.	144

7.3.1	Updating the WebSphere Application Server startup script	144
7.3.2	Configuring a WebSphere data source	145
7.4	Deploying TraderDB in WebSphere Application Server	149
Chapter 8.	WebSphere MQ connectors	153
8.1	Introducing the MQ environment	154
8.1.1	MQ-CICS bridge	156
8.1.2	MQ-IMS bridge	156
8.2	WebSphere MQ setup on Linux for zSeries frontend	157
8.3	WebSphere MQ setup on z/OS backend	162
8.3.1	Configuring queues and channels on z/OS	162
8.3.2	Configuring queues and channels on Linux for zSeries	167
8.4	WebSphere MQ setup on VSE backend	168
8.4.1	Configuring queues and channels on Linux for zSeries	168
8.4.2	Defining MQ resources to Linux for zSeries	168
8.4.3	Shell script to define the Linux-VSE connection	169
8.5	Configuring VSE for MQ	175
8.5.1	Defining the VSAM data files	176
8.5.2	Defining the MQ files to CICS using RDO	180
8.5.3	Defining MQ resources to VSE/ESA	184
8.5.4	Defining MQ local queues on VSE	185
8.5.5	MQ troubleshooting	195
8.6	Configuring the MQ connector in WebSphere	196
8.6.1	Defining a WebSphere MQ queue connection factory	197
8.6.2	Defining WebSphere MQ queue destinations	197
8.6.3	Defining message listeners	197
8.6.4	Deploying TraderMQ application to WebSphere	198
8.6.5	Defining resources for VSE in WebSphere	200
Chapter 9.	IMS J2EE connectors	207
9.1	IMS connectors overview	208
9.1.1	IMS Connect	208
9.1.2	IMS Connector for Java	208
9.2	Installing IMS Connector for Java	209
9.3	Installing the IMS resource adapter in WebSphere Application Server	211
9.4	Configuring IMS J2C connection factories	214
9.5	Deploying TraderIMS application in WebSphere Application Server	217
9.6	IMS Connect configuration	218
9.7	Testing TraderIMS application	220
9.8	Problem determination	220
9.8.1	Common errors	221
9.9	Thread identity support	222
Chapter 10.	VSE Java-based connector to access VSAM data	223

10.1	VSE Java-based connectors overview	224
10.1.1	Client-server components	225
10.2	Installing the VSE Java-based connector	226
10.2.1	Client	226
10.2.2	Server	227
10.3	Using VSE Connector Client as resource adapter	227
10.3.1	Defining a resource adapter	227
10.3.2	Related servlet code	231
10.4	Using the VSE connector client as a JDBC provider	231
10.4.1	Defining a JDBC provider	232
10.4.2	Related servlet code	240
10.5	Setting up sample data for Trader	241
10.5.1	Installing the VSE Navigator	242
10.5.2	Installing the VSAM maptool	242
10.5.3	Creating the sample VSAM files	243
10.5.4	Creating the VSAM maps	244
10.5.5	Populating the sample VSAM files	249
10.6	Installing an application in WebSphere	252
10.6.1	Setting up an EAR file in the Application Assembly Tool	253
10.6.2	Deploying the EAR file in WebSphere Administrative Console	260
10.7	Configuring for SSL secure connections	262
10.7.1	Installing Keyman/VSE	263
10.7.2	Generating keys and certificates	264
10.7.3	Uploading certificate items to VSE	270
10.7.4	Transferring the keyring file to WebSphere	272
10.7.5	Checking VSE keyring library	272
10.7.6	Defining an SSL connection factory in WebSphere	273
10.7.7	Adding SSL resource reference in EAR file	275
10.7.8	Redeploying the EAR file	275
10.7.9	Configuring VSE Connector Server for SSL	277
10.7.10	Restarting VSE Connector Server	279
10.7.11	Changing your servlet code to support SSL	279
10.7.12	Configuring an SSL JDBC data source	280
10.7.13	Considerations on SSL key lengths	282
10.7.14	Considerations on different SSL scenarios	283
10.8	Problem determination	284
10.8.1	Activating stdout trace in WebSphere	284
10.8.2	Tracing a servlet	285
Chapter 11. VSE Java-based connector to access DL/1 data		287
11.1	DL/I database access overview	288
11.1.1	Prerequisites for the DL/1 connector	289
11.1.2	The DL/I example	289

Chapter 12. VSE VSAM Redirector connector	293
12.1 VSAM Redirector connector overview	294
12.2 Client-server components	295
12.3 Installing the Redirector server	297
12.3.1 Downloading the Redirector server	297
12.3.2 Installing the Redirector server	297
12.3.3 Configuring the Redirector server	298
12.3.4 Starting the Redirector server	299
12.4 General setup using the HtmlHandler	300
12.4.1 Step 1: Decide on the VSAM source file	301
12.4.2 Step2: Create a new file as target file	302
12.4.3 Step 3: Configure and activate Redirector client exit	303
12.4.4 Step 4: Modify the HtmlHandler Java source	305
12.4.5 Step 5: Create a sample VSAM application	307
12.4.6 Step 6: Run the REPRO job	307
12.4.7 Step 7: Check the HTML output file	308
12.5 Special setup for Trader using the DB2 handler	309
12.5.1 Providing VSAM map definition	309
12.5.2 Populating the sample VSAM files	312
12.5.3 Preparing for creating DB2 tables	312
12.5.4 Creating DB2 tables	312
12.5.5 Setting up the VSE side	314
12.5.6 Testing the setup	315
12.5.7 Checking the target DB2 table	316
12.5.8 Trader scenario with redirected VSAM files	317
Appendix A. VSE/ESA code samples	319
Sample Java SOAP client program	320
Sample Java program to populate VSAM files on VSE	322
IBM-provided SOAP service C program	325
IBM-provided include file for SOAP	330
Appendix B. Additional material	337
Locating the Web material	337
Using the Web material	337
System requirements for downloading the Web material	338
How to use the Web material	338
Related publications	339
IBM Redbooks	339
Other publications	339
Online resources	340
How to get IBM Redbooks	342
Help from IBM	342

Archived

Archived

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. 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 IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law. INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.


This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

1-2-3®	Hummingbird®	Redbooks (logo)  ™
CICS/ESA®	IBM®	RACF®
CICS/VSE®	ibm.com®	SupportPac™
CICS®	IMS™	Tivoli®
DB2 Connect™	Lotus®	VSE/ESA™
DB2®	MQSeries®	WebSphere®
developerWorks®	OS/2®	z/OS®
DFS™	OS/390®	zSeries®
DRDA®	POWER™	z/VM®
HiperSockets™	Redbooks™	

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

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

Preface

This IBM® Redbook discusses Linux-based Java™ applications connecting to z/OS® and VSE backend environments on zSeries®. The book describes the implementation and deployment of Websphere Application Server, Java frontend applications, and the J2EE connectors needed on Linux for zSeries to connect to backend applications, such as CICS®, IMS™, DB2® and MQ on zSeries. This book explains the J2EE Connector architecture and provides the following comprehensive connector scenarios for connections to both z/OS and VSE backend environments:

- ▶ CICS Transaction Gateway
- ▶ IMS Connect
- ▶ WebSphere® MQ
- ▶ JDBC to DB2
- ▶ SOAP to CICS

The team that wrote this redbook

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

Viviane Anavi-Chaput is a Senior IT Specialist for zSeries Software Solutions at the IBM International Technical Support Organization, New York. She writes extensively, teaches worldwide, and presents at international conferences. Before joining the ITSO in 1999, Viviane was a Senior Data Management Consultant at IBM Europe, France. She was also an ITSO Specialist for DB2 at the San Jose Center from 1990 to 1994.

Michael Rask Christensen is a Senior IT Specialist in Denmark. He has three years of experience in J2EE application design and development on various platforms. He has worked for IBM for three years. Before that he worked since the early 1980s as a PL/I and VAG developer for various IBM customers. His areas of expertise include J2EE, MQ and DB2.

Joerg Haertel is a Senior IT Specialist working for System Sales Technical Support zSeries in Germany. He holds a diploma in communications engineering. He has 16 years of technical experience in the VM and VSE environment. He has worked at IBM for 18 years. His areas of expertise include Linux for zSeries, TCP/IP, DB2 and zSeries-related hardware. He has written

extensively on the CICS, CTG and MQSeries® setup on Linux as well as on VSE.

Joerg Schmidbauer is a VSE developer in the IBM Boeblingen Lab, Germany. He mainly works on VSE connectors, including SSL and crypto-related functions.

Lynn Winkelbauer is a system engineer in Poughkeepsie, NY. She has 19 years of experience in CICS, z/VM®, z/OS and Linux for zSeries. She holds a Bachelor of Science degree in Computer Science and Business Administration. Her areas of expertise include Linux for zSeries, CICS TS on z/OS, and WebSphere.

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

Dave Bennin, Rich Conway, Greg Geiselhart, Robert Haimowitz, Franck Injey, Tamas Vilaghy, Julie Czubik
International Technical Support Organization, Poughkeepsie Center

Claus Schroder-Hansen
IBM Denmark

Ingo Franzki, Karsten Graul, Wilhelm Mild
IBM Boeblingen Lab, Germany

Hans Joachim Ebert, Dagmar Kruse
System Sales Technical Support zSeries, Muenchen, Germany

Hidenori Fujioka
IBM Japan

Phil Wakelin
IBM UK

Bob Cronin, Mitch Johnson, Allen Schmutzler
IBM USA

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an Internet note to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Archived

J2EE connector architecture overview

This chapter gives an overview of J2EE connector architecture (JCA, also referred to as J2C) for connecting the J2EE platform to enterprise information systems (EIS).

We describe the following:

- ▶ J2C overview
- ▶ System contracts
 - Connection management
 - Transaction management
 - Security management
- ▶ Common client interface
- ▶ Deployment and packaging

For detailed coverage of the J2C specification see the Java Community Process site at:

<http://jcp.org/en/jsr/detail?id=16>

On the Java software site there is a whitepaper that also gives a short introduction to the J2C, at:

<http://java.sun.com/j2ee/white/connector.html>

1.1 J2C overview

As Java became the defacto language for Internet application development, many businesses have their enterprise applications based on it. Java 2 Platform Enterprise Edition (J2EE) is the specification for a standard Java platform to meet the requirements of the enterprise. It provides a component-based, server-centric, multi-tier application architecture.

As the J2EE platform is designed for enterprise computing, it should be no surprise that applications need to connect to EIS, such as:

- ▶ Enterprise resource planning (ERP) systems, such as SAP R/3
- ▶ Mainframe transaction processing systems, such as CICS
- ▶ Legacy applications and non-relational database systems, such as IMS

In the past, most EIS vendors and application server vendors used vendor-specific architectures to provide EIS integration. This means that for each application server an EIS vendor wants to support, the EIS vendor needs to provide a specific connector (also called adapter); and for every connector an application server wants to support it will need to extend the server.

To overcome this problem the J2C defines a standard for connecting a compliant J2EE platform to EIS through the usage of a resource adapter and the common client interface.

1.1.1 Resource adapter

If both EIS vendors and application server vendors follow this architecture, then only one resource adapter (RA) needs to be written for an EIS, which can plug into any J2EE-compliant application server.

1.1.2 Common client interface

Conversely, an application server only needs to be extended (or support the common client interface, CCI) to conform to the J2C, and this will ensure that any J2EE EIS connector will work with it.

Figure 1-1 shows on one side an EIS connecting to multiple application servers and, on the other side, one application server connecting to many EISs.

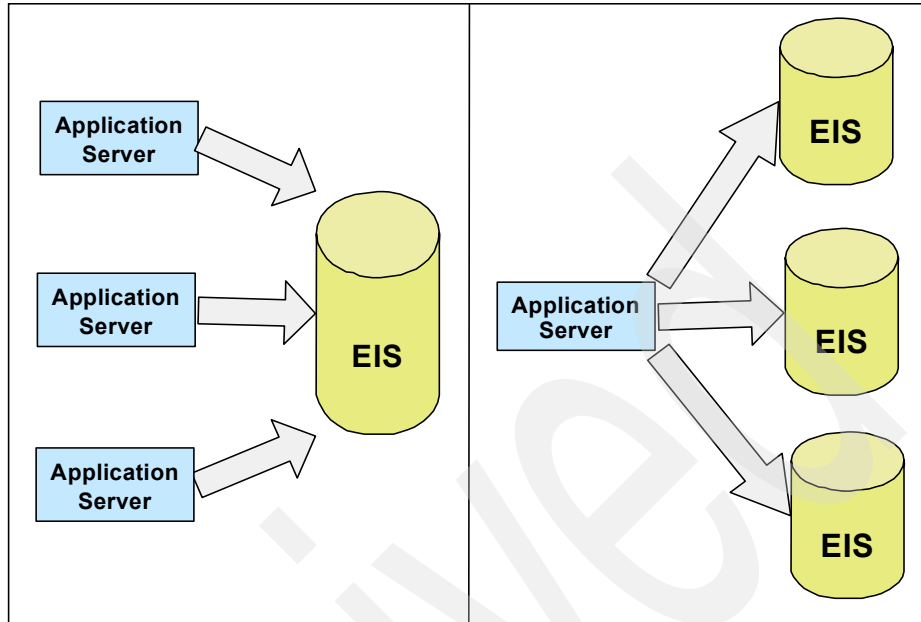


Figure 1-1 One resource adapter - Two different topologies

1.1.3 System contracts

The architecture provides this pluggability by defining a standard set of system contracts between an application server and the resource adapter. In the current release of J2C 1.0 the contracts defined are:

- ▶ Connection management
- ▶ Transaction management
- ▶ Security management

More contracts will be supported in later releases of the specification, but these are the three most pressing concerns for enterprise integration and are mandatory in the implementation of J2C.

The J2EE connector architecture makes system-level mechanisms transparent to the application components by having the application server and EIS manage them. This allows the application developer to focus on the business logic and presentation, without needing to worry about the low-level integration issues. This in turn leads to easier and faster development times, lowering costs of both development and maintenance. Figure 1-2, describes the J2EE connector architecture and how system contracts are made transparent to the application component.

Certain system functions can be container-managed or component-managed:

► Container-managed

In an application server, different functions are grouped together in containers. The J2EE implementation in an application server is one such container, and the others are the servlet container and EJB container. In this chapter when something is described as container-managed, it means that the container handles the management of the function to be executed. Container-managed also allows the application server to introduce quality of service and other system services. An example of this is the ability to use connection pooling provided by the application server; if you wanted connection pooling to be component-managed you would need to implement it yourself.

► Component-managed

When the application component manages something, it means it is bypassing the application server and dealing directly with the resource adapter instead of letting the application server deal with the adapter on its behalf.

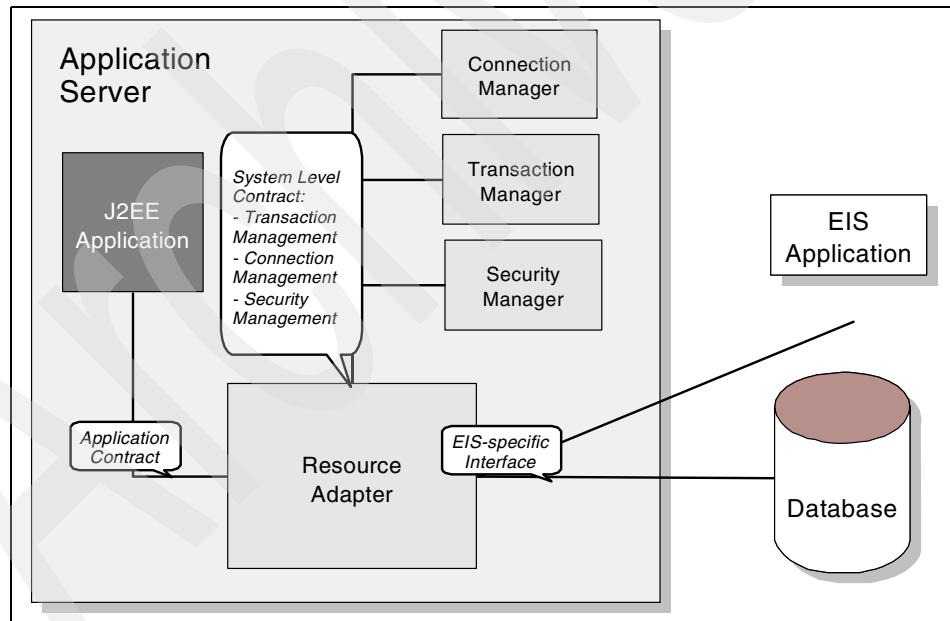


Figure 1-2 J2EE connector architecture overview

1.1.4 Application contracts

The architecture also defines an application contract, which is implemented as the common client interface (CCI). This defines a common API set for interacting with J2EE resource adapters. The CCI is targeted at EIS development tools and other sophisticated users of EISs. The CCI provides a way to minimize the EIS-specific code required by such tools. Most J2EE developers access EISs using these tools rather than using CCI directly.

1.1.5 Deployment and packaging protocols

There are also definitions for a standard deployment and packaging protocols for the resource adapters.

1.2 System contracts

To achieve the ease of pluggability between the application server and EIS, the architecture defines a set of system-level contracts. The application server is extended to support these contracts, and the EIS vendors implement a resource adapter. The resource adapter implements the system contracts to collaborate with the application server and use an EIS-specific API to communicate with the EIS. System contracts include the following topics:

- ▶ Connection management
- ▶ Transaction Management
- ▶ Security management

1.3 Connection management

The connection management contract simply gives an application component a connection to an EIS. This is sometimes all the application developer needs or wants to know, but with enterprise computing you need to also know that the connection you are getting is fast and scalable. To deliver this performance and scalability, connection management contracts implement connection pooling.

1.3.1 Connection pooling

Connection pooling is a quality of service offered by the application server. When retrieving data from an EIS, a large portion of the time is spent in opening and closing the connection. To save the open/close time of the connection, what you need is a connection that is defined in advance, so that when you actually want to use the connection it is already there waiting to be used. This is exactly what

connection pooling is all about; it is a collection (pool) of connections that have already accomplished the hard work of making the connection to the EIS.

When you call for a connection, you are just passed a handle to the next available connection that is in a ready-to-use state. This considerably increases the performance by removing the actual connection time, and scalability is achieved by predefining as many connections in the pool as you need.

1.3.2 Managed access to the resource adapter

This is typically an environment in which the application component and resource adapter are running in an application server. The resource adapter and application component use the runtime supplied by the server. This is a scenario where the application accesses the resource adapter through the application sever. Figure 1-3 shows the architecture of the connection management between an application server and resource adapter in a managed environment.

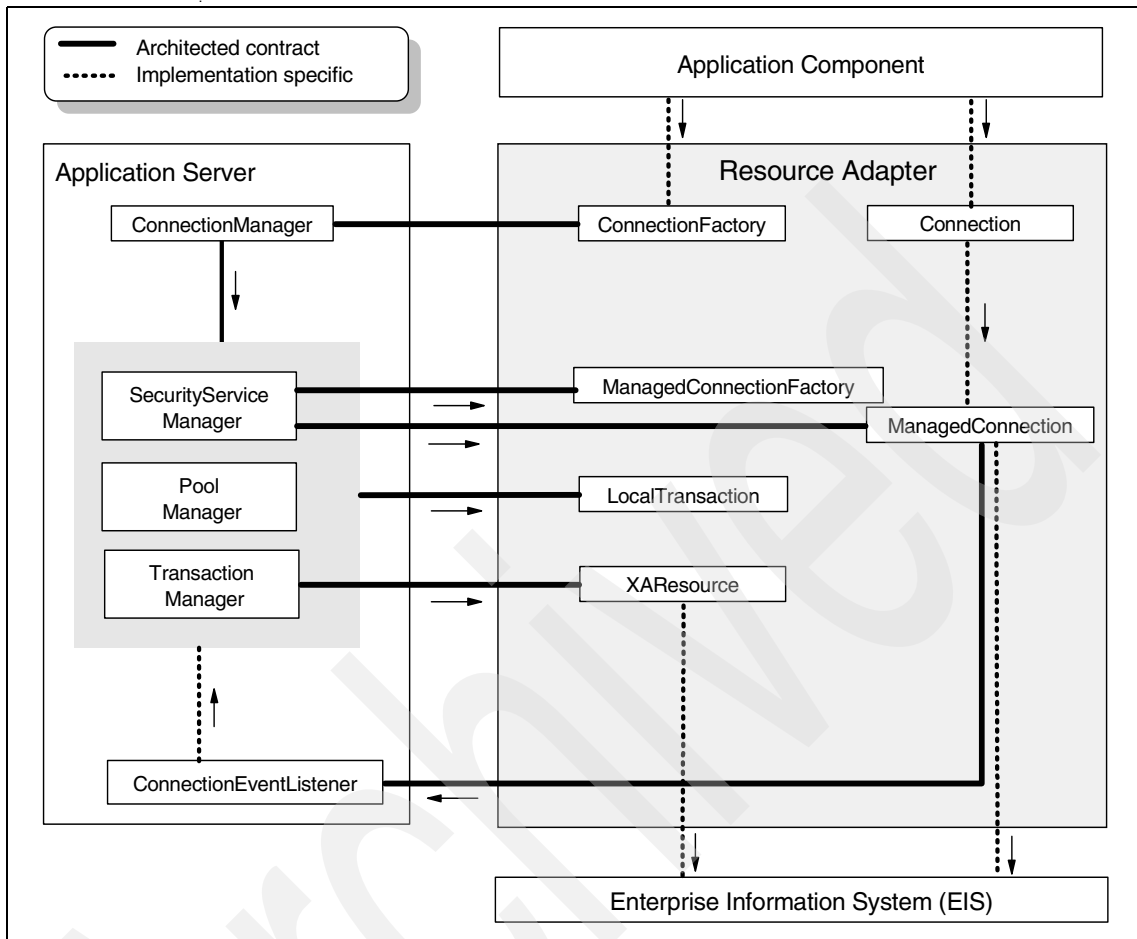


Figure 1-3 Connection management architecture

When you ask for a connection, the resource adapter provides a *Connection* from a *ConnectionFactory*, which acts as a factory for EIS connections. The CCI defines *javax.resource.cci.ConnectionFactory* and *javax.resource.cci.Connection* as interfaces for a connection factory and a connection, respectively.

The application component uses JNDI to do a lookup of a connection factory. The connection factory instance then delegates the creation request for a connection to the *ConnectionManager* instance.

The ConnectionManager that is supplied by the application server enables different qualities of services to be provided. These qualities of services include transaction management, security, error logging and tracing, and connection pool management. The application server can provide these services in any implementation-specific way it chooses; the specification does not specify how it should be done.

The ConnectionManager instance looks in the connection pool provided by the application server to see if there is a connection that will satisfy the request. If there is no connection in the pool that can satisfy the connection request, the application server uses the ManagedConnectionFactory interface to create a new physical connection to the underlying EIS. If a matching connection is found in the pool, then it uses the matching ManagedConnection instance to satisfy the connection request.

If a new ManagedConnection instance is created, the application server adds the new ManagedConnection instance to the connection pool.

A ConnectionEventListener is registered with the ManagedConnection instance by the application server. This listener enables the application server to get event notifications related to the state of the ManagedConnection instance. The application server can then use these notifications to manage connection pooling, transactions, cleanup connections, and handle any error conditions.

The application receives a handle to the connection from the application server, which gets it from the ManagedConnection instance. The application component then uses this handle to access the EIS.

1.3.3 Non-managed access to the resource adapter

In the non-managed environment there is no application server. There is a runtime machine supplied by the Java platform and the application accesses directly the resource adapter.

The ConnectionManager class may be implemented by either the resource adapter as a default ConnectionManager, or by the application developer.

On a connection request, the default ConnectionManager instance intercepts the request and passes it on to the ManagedConnectionFactory instance. The physical connection to the EIS is then created by the instance of the ManagedConnectionFactory. The ConnectionManager gets a handle to the connection from the ManagedConnection and returns it to the connection factory, which then passes it to the application, as shown in Figure 1-4.

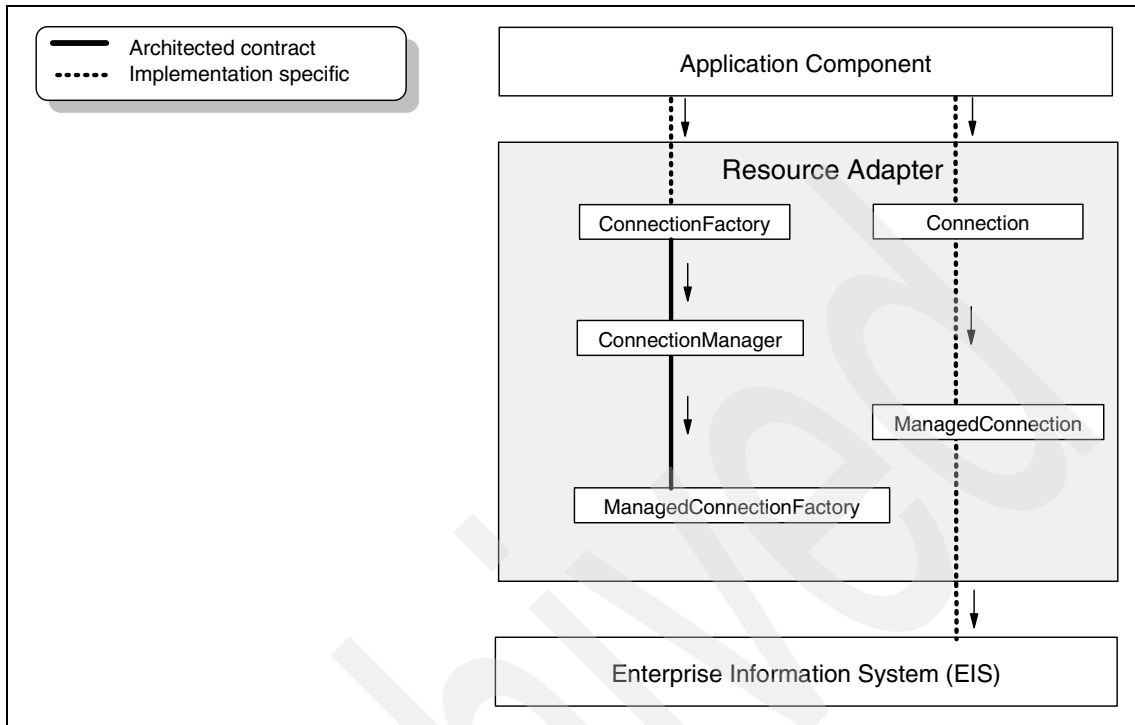


Figure 1-4 Connection management in a non-managed environment

1.4 Transaction management

One of the prime requirements of any type of computing is data integrity, that is, you have full confidence that your data is correct and consistent.

What is a transaction

Quite often you need to manipulate data by performing more than one operation on it. A transaction is a group of such operations, that is, to maintain data integrity all must happen or else none should happen at all.

A good transaction example is moving money from one account to another. You might have two simple steps, debit money from account A and credit money in account B. If everything goes correctly, after you complete your two steps the same amount of money removed from account A will have been added to account B. What happens if after debiting account A there is a computer crash before you can credit it to account B, and you lose knowledge of the transaction and neither account has the money?

The idea of a transaction is when you take the money out of account A you do not commit the change until after you successfully put the money in account B. At that time you commit the changes to both accounts and all is well. If there is a crash before you can make the commit, when the system restarts it can see that there was an uncommitted transaction and rolls back the changes so that both accounts are put back in the same state as they were before the transaction started.

Transactions are handled in the J2C by the transaction management contract.

Transaction management contract

Figure 1-5 shows the architecture of the transaction management contract.

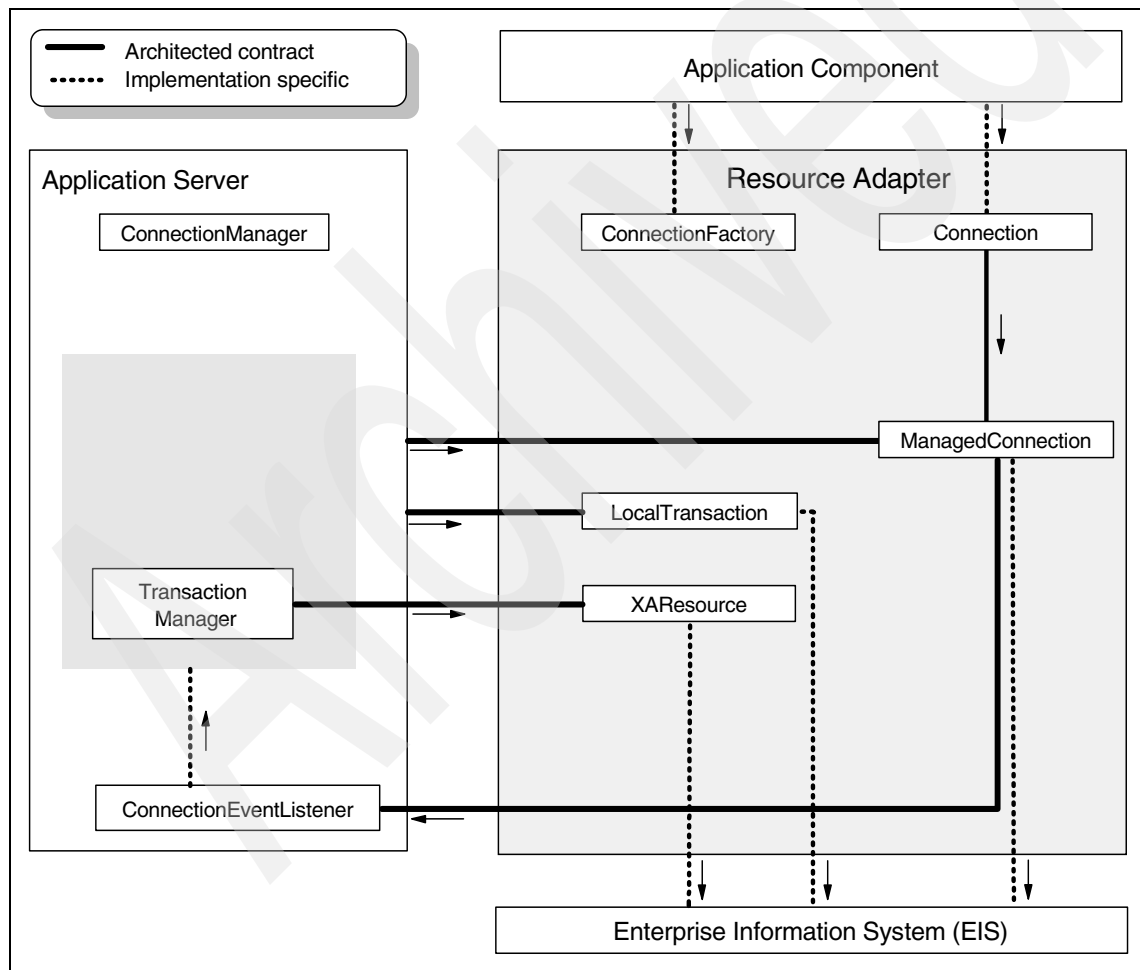


Figure 1-5 Transaction management contract

The application component may need to perform transactions over one or more resource managers (*resource manager* refers to the combination of resource adapter and the resource manager of the underlying EIS). To do this, the application server uses a transaction manager to manage transactions across multiple resource managers. The application server and the transaction manager both communicate with the resource adapter via the transaction management contract.

A resource manager has three options for supporting transactions:

- ▶ No support—The resource adapter and underlying EIS do not support transactions.
- ▶ Local transaction—The resource manager is responsible for coordinating the transaction.
- ▶ Global transactions—There are multiple resource managers involved and an external *transaction manager* must be used to coordinate the transaction using two-phase commit. Optionally, the transaction manager may use one-phase commit when only one resource is involved in the transaction.

The level of transaction support can be set by the resource manager (or deployer) in the deployment descriptor described in 1.7.2, “Deployment descriptor” on page 20.

1.4.1 Global transactions

These transactions are also referred to in the specification as JTA transactions and are supported by the resource adapter implementing the `javax.transaction.xa.XAResource` interface.

In a managed environment, the application server uses a transaction manager to coordinate the transaction. The application server will inform the transaction manager when a transaction begins. It will then perform some actions and then tell the transaction manager to commit the transaction.

In a non-managed environment, the application component is responsible for doing the job of the transaction manager. By using the managed environment, the programmer does not even need to think about managing the transaction, as the transaction manager is one of the qualities of services provided.

The transaction manager uses the `XAResource` interfaces of the resource adapters to coordinate the two-phase commit process across multiple resource managers. With two-phase commit each resource manager is queried about the success of its transaction by the transaction manager issuing a prepare statement. Each resource manager is then required to return information about the success of its transaction. If all resource managers reply with success then

the transaction manager issues a COMMIT causing all the resource managers to commit their resources. If any of the resource managers reply with a failure then the transaction manager issues a ROLLBACK, forcing all the resource managers to back out of the transaction.

Although the XAResource interface is intended to support two-phase commit, the specification does not force an adapter to support two-phase commit. However, if the resource adapter does implement XAResource it must also implement support for one-phase commit. This allows the transaction manager to do one-phase commit optimization by setting the onePhase flag to true when doing a commit.

If the resource manager has indicated in the deployment descriptor that it supports xa_transaction, and it has an implementation of XAResource, then the application server assumes that two-phase commit is supported. If the resource manager has only implemented one-phase commit, two things could happen:

- ▶ Only one resource is referenced—the transaction manager will perform one-phase commit optimization so everything will work.
- ▶ Multiple resources are referenced—the transaction manager will issue a prepare statement, at which point the resource manager is forced to acknowledge that it does not really support two-phase commit by issuing an exception.

1.4.2 Local transactions

A local transaction is managed by the resource manager without the need for an external transaction manager, and can be utilized when only one resource is involved. Local transactions only support one-phase commit, because they only reference one EIS.

To support local transactions, the resource manager must implement the `javax.resource.spi.LocalTransaction` interface. If the resource adapter supports the CCI, then it will also send a number of transaction events to the application server.

The application server is required to implement the interface `javax.resource.spi.ConnectionEventListener`, which, among other events, allows the application server to hear and react to the following local transaction events:

- ▶ LOCAL_TRANSACTION_STARTED
- ▶ LOCAL_TRANSACTION_COMMITTED
- ▶ LOCAL_TRANSACTION_ROLLEDBACK

By listening for these events the application server can do various things, such as local transaction cleanup.

1.4.3 One-phase commit optimization

One-phase commit optimization is forcing the use of one-phase commit in the situation when two-phase commit is not needed. This would be when only one EIS was referenced, so two-phase commit would be an unnecessary overhead.

When the application server needs to do a global transaction, it informs the transaction manager of its intention to begin a transaction. The application server then performs whatever operations it has to, and when finished informs the transaction manager to commit the transaction. The transaction manager now has the ability to do one-phase optimization. If the number of EISs referenced is only one, then the transaction manager skips the prepare statement and goes straight to commit with the onePhase flag set to true.

1.4.4 Local transactions versus one-phase commit optimization

Since there is an overhead with using the transaction manager, if you know there is only one resource involved in the transaction, the transaction support in the deployment descriptor can be set to `local_transaction`. When this is set, the application server uses the local transaction methods instead of going to the transaction manager and allowing it to perform the one-phase commit optimization.

1.5 Security management

The single biggest concern of e-business enterprise computing is security. The end users are concerned that their personal details are secure, and the business is concerned that only authorized access to their EISs is allowed.

Three mechanisms can be used to provide security:

- ▶ Authentication—Verifying that the users are who they say they are
- ▶ Authorization—Controls access to services
- ▶ Secure communications—Securing the link between end points, for example, using secure socket layer (SSL)

1.5.1 Terminology

We explain some terminology before we continue with our description of security.

- ▶ Principal—An entity that can be authenticated using an authentication mechanism
- ▶ Security attribute—A set of security attributes associated with a principal

- ▶ Credential—Security information about a principle that can be used to authenticate the principal to other services
- ▶ Security principal—A security principal under whose security context a connection to an EIS is established

1.5.2 Security model overview

There are two sign-on scenarios when an application component requests a connection to an EIS under the security context of a resource principal.

The *res-auth* element in the deployment descriptor must be set to indicate which method is being used.

Container-managed

The deployer must set up the resource principal and sign-on information, for example, the deployer sets the username and password to be used for the connection.

The *res-auth* descriptor element should be set to *Container*. The deployer also configures the descriptor to hold the authentication data (user ID, password, etc.) needed for the authentication.

The application then uses the `getConnection` method of the `ConnectionFactory` and lets the application server manage the security to sign on to the EIS. Example 1-1 is a simple sample of application code to get the connection.

Example 1-1 Requesting a connection with the application server handling the security

```
// create a Context
Context ctx = new InitialContext();

// use JNDI to get a ConnectionFactory
javax.resource.cci.ConnectionFactory cxf =
(javax.resource.cci.ConnectionFactory) ctx.lookup("some/jndi/name");

// request a connection
// note that no security information is passed in the call to getConnection
javax.resource.cci.Connection con = cxf.getConnection();
```

Component-managed

The component code is responsible for supplying the sign-on information to be used by the resource principal.

The res-auth element needs to be set to Application. The application code must then supply all the security information when making the connection. This can be seen in the code sample in Example 1-2.

Example 1-2 Requesting a connection with the application code handling the security

```
// create a Context
Context ctx = new InitialContext();

// use JNDI to get a ConnectionFactory
javax.resource.cci.ConnectionFactory cxf =
(javax.resource.cci.ConnectionFactory) ctx.lookup("some/jndi/name");

// create a connectionSpec to hold the security information
sample.ConnectionSpec cs = new sample.ConnectionSpec();

// set the userid/password
cs.setUsername("userid");
cs.setPassword("password");

// request the connection passing the ConnectionSpec to getConnection
javax.resource.cci.Connection con = cxf.getConnection(cs);
```

To create a connection to an EIS, there must be some form of signing on to the EIS—this is to authenticate who the connection requester is. Re-authentication can also take place if supported by the EIS—this is when the security context is changed after a connection is made (connection pooling could cause a re-authentication when the connection is redistributed). Not all EISs support re-authentication, so the J2C only recommends this and does not make it mandatory for a resource adapter to implement.

Performing the sign-on generally involves one or more of the following steps:

- ▶ Determine the resource principal under whose security context the connection will be made.
- ▶ Authenticate the resource principal.
- ▶ Establish secure communications.
- ▶ Determine authorization (that is, access control).

Resource principal

The resource principal security context is used when a connection is made. The deployer can set the resource principal with the following options:

- ▶ Configured identity—A resource principal has its own identity that can be configured at deployment time or dynamically by the application component when connecting.

- ▶ Principal mapping—A resource principal maps the identity and/or security attributes of the calling principal. It does not inherit the attributes but only maps them.
- ▶ Caller impersonation—A resource principal acts on behalf of the calling principal. This requires that the caller's credential is delegated to the EIS. The way this is implemented is dependent on the security mechanism used.
- ▶ Credentials mapping—If the security mechanisms used by the application server and the EIS are different, then the credentials of the calling principal are mapped to a form that is understood by the security mechanism of the EIS.

Authentication

Authentication is the mechanism of determining that you are what you say you are. The EIS must be sure of who is connecting to it and only allow connections from trusted parties.

The connector architecture does not require any specific authentication methods and is independent of the security mechanism.

Although the specification does not require any specific mechanism, it does identify the following two options as commonly supported authentication mechanisms:

- ▶ Basic password—Standard user-password mechanism specific to a EIS.
- ▶ kerbv5—This is an authentication mechanism called Kerberos (Version 5).

If a specific type of mechanism is used, the deployer uses the *auth-mech-type* element in the deployment descriptor to identify this.

Secure communication

If the links on which information is being sent are not secured, there is a possibility that the information can be intercepted. You then cannot be certain that you received all data or that the data you did receive was not modified. Methods such as cryptographic keys and message sequence numbers can be used to secure the link.

Securing the link between the application server and the EIS is always handled by the resource adapter and can be implemented using any security mechanism the resource adapter chooses.

Authorization

Checking that a principal is authorized to access certain resources can be done either at the applications server or in the EIS.

If the checking is done in the EIS, then it is done in an EIS-specific way and is independent of the J2C specification.

The application server can also be configured to only allow connections to be made from an authorized principal. In J2EE both the EJB and Servlet engines have the capacity to support both programmatic and declarative security to set up the authorization policy.

1.6 Common client interface

The common client interface (CCI) defines a standard client API for application components to access the resource adapter. The intention of the CCI is not for application developers to use it directly, as it is a quite low-level API, but for enterprise application integration (EAI) frameworks to use, to generate EIS access code for the developer. The CCI is designed to be an EIS-independent API, such that an EAI can produce code for any J2EE-compliant resource adapter that also implements the CCI interface.

Important: The CCI is only recommended in the specification. It is not mandatory. IBM connectors will implement it.

1.6.1 Enterprise application integration

The CCI does not replace the standard JDBC API; instead, it complements it. JDBC is used for accessing relational databases, and the CCI can be used to access all other EISs that are not relational databases. This can be seen in Figure 1-6.

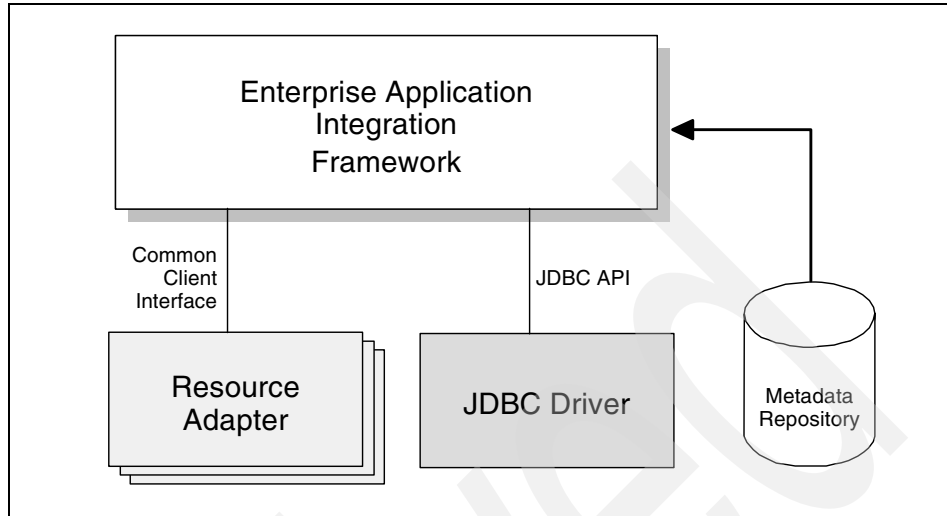


Figure 1-6 Enterprise application integration framework

The vendor of the EAI uses the CCI as a standard way to plug into resource adapters. The framework they supply sits on top of the functionality provided by the resource adapter.

The framework can also use a metadata repository to get meta information about the underlying EIS. This metadata may include things like the underlying data structure of the commarea that is passed by a CICS ECI interaction with a CICS J2EE resource adapter.

The enterprise application development tool uses the metadata to produce Java classes. These classes encapsulate CCI-based API calls and expose simple interactions to the application developer, usually as JavaBeans. Among other things, the application developer then has a simple set of commands to set input data, read output data, and execute the transaction. This model can be seen in Figure 1-7.

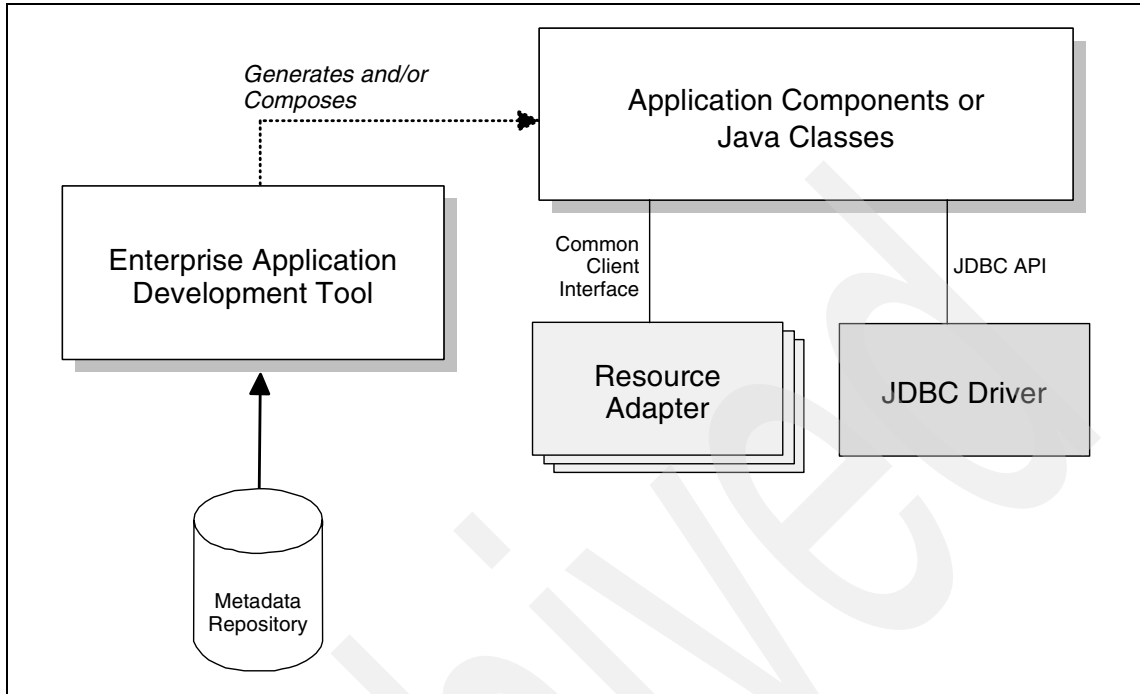


Figure 1-7 Enterprise application development tool

1.7 Deployment and packaging

To use the resource adapter on a J2EE application server, the resource adapter needs to be deployed. There are two options for deploying the resource adapter:

- ▶ As a stand-alone module in the application server so that multiple applications can access it
- ▶ As part of the deployment of a J2EE application, which can have a number of modules as well as the resource adapter module

Reference was made to deploying a *module*. In J2EE a module is the basic unit of composition of a J2EE application. Other examples of J2EE modules are the EJB module, a Web client module, and an application module.

1.7.1 Packaging of the resource adapter module

A packaged resource adapter consists of:

- ▶ Java classes that implement the J2EE/CA contracts and the other functionality of the adapter
- ▶ Any utility classes
- ▶ Native libraries required for any platform dependencies
- ▶ Documentation
- ▶ A deployment descriptor

This is all packaged together using the Java Archive (JAR) format into a Resource Adapter Archive (RAR). The deployment descriptor must be stored in the rar file with the name META-INF/ra.xml. The Java classes used should be stored as jar files, and multiple jar files can be stored if needed.

Example 1-3 on page 20 is an example of how the RAR file directory structure could look.

Example 1-3 Example directory structure of RAR file

```
/META-INF/ra.xml  
/doco.html  
/adaptor.jar  
/windows.dll  
/aix.so
```

1.7.2 Deployment descriptor

The provider of the resource adapter is responsible for specifying the deployment descriptor, while the deployer is responsible for configuring it in a target environment.

The following must be specified in the descriptor by the provider:

- ▶ General information
 - Name of the resource adapter
 - Description
 - URI of a UI icon
 - Vendor name
 - Licensing requirements
 - Type of EIS supported
 - Version of J2EE/CA supported
 - Version of resource adapter

- ▶ ManagedConnectionFactory fully qualified class name
- ▶ Connection factory interface and implementation fully qualified class name
- ▶ Connection interface and implementation fully qualified class name
- ▶ Transactional support, that is, none, local, or xa
- ▶ Configurable properties per ManagedConnectionFactory instance: Name, type, description and optional default values that have to be configured per instance of a ManagedConnectionFactory
- ▶ Authentication mechanisms supported
- ▶ Reauthentication supported, that is, yes or no
- ▶ Extended security permissions

Example 1-4 shows a sample deployment descriptor for a sample CICS Connector.

Example 1-4 Sample deployment descriptor for CICS Connector

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE connector PUBLIC "EN" "connector_1_0.dtd">
<connector>
  <display-name>J2CICS Connector</display-name>
  <description>Provides access to the CICS systems</description>
  <vendor-name>IBM Corporation</vendor-name>
  <spec-version>1.0</spec-version>
  <eis-type>CICS</eis-type>
  <version>1.0</version>
  <resourceadapter>

  <managedconnectionfactory-class>com.ibm.connector2.cics.J2CICSManagedConnection
Factory
  </managedconnectionfactory-class>

  <connectionfactory-interface>javax.resource.cci.ConnectionFactory</connectionfa
ctory-interface>

  <connectionfactory-impl-class>com.ibm.connector2.cics.J2CICSConnectionFactory
  </connectionfactory-impl-class>
  <connection-interface>javax.resource.cci.Connection</connection-interface>

  <connection-impl-class>com.ibm.connector2.cics.J2CICSConnection</connection-imp
l-class>
  <transaction-support>local_transaction</transaction-support>
  <config-property>
    <config-property-name>logonLogoffClassName</config-property-name>
    <config-property-type>java.lang.String</config-property-type>
```

```

<config-property-value>com.ibm.connector2.sample.appclient.ClientLogonClass
</config-property-value>
</config-property>
<config-property>
  <config-property-name>ServerName</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
  <config-property-value>testsrv</config-property-value>
</config-property>
<config-property>
  <config-property-name>GatewayHostName</config-property-name>
  <config-property-type>java.lang.String</config-property-type>
  <config-property-value>testgateway</config-property-value>
</config-property>
<config-property>
  <config-property-name>Port</config-property-name>
  <config-property-type>java.lang.Integer</config-property-type>
  <config-property-value>testport</config-property-value>
</config-property>
<config-property>
  <config-property-name>useUserData</config-property-name>
  <config-property-type>java.lang.Boolean</config-property-type>
  <config-property-value>true</config-property-value>
</config-property>
<auth-mechanism>
  <auth-mech-type>basic-password</auth-mech-type>
  <credential-interface>javax.resource.security.PasswordCredential
  </credential-interface>
</auth-mechanism>
<reauthentication-support>False</reauthentication-support>
</resourceadapter>
</connector>

```

1.7.3 Deploying the resource adapter

As mentioned before, the deployer is responsible for configuring the resource adapter in the target environment. To do this, the deployer uses a deployment tool. The deployment tool should be part of a J2EE-supported application servers suite of tools.

The deployment tool should read the ra.xml file in the rar file. It should then install all the components of the resource adapter module into the application server.

To configure the resource adapter, the deployer must create a property set (one set per ManagedConnectionFactory instance). This is done using the deployment tool to set valid values for each field based on the name, type, and description in the deployment descriptor.

One resource adapter can be used to provide connections to multiple instances of the same EIS, so there can be multiple property sets to configure (one per ManagedConnectionFactory instance). The deployment tool may make multiple copies of the deployment descriptor to accomplish this.

The deployer also needs to configure the application server based on the type of transactions supported by the resource adapter, as well as the security requirements specified by the resource adapter. The deployer can also check that the EIS also supports the same security mechanism as the adapter. If it does not, the deployer may decide not to configure that type of security. The deployer does not have to check this, but if the mechanisms do not match there will be runtime exceptions to indicate a problem.

Archived

Archived



Introducing the connectors test environment

In this chapter, we introduce the test environment infrastructure we used to test our connectors.

We describe the following:

- ▶ System configurations
- ▶ Frontend environment
- ▶ Backend environments
- ▶ Connector scenarios overview

2.1 System configurations

Figure 2-1 describes the system configurations that we used to test the Trader application, which uses the connectors on our Linux for zSeries to connect to z/OS and VSE backends.

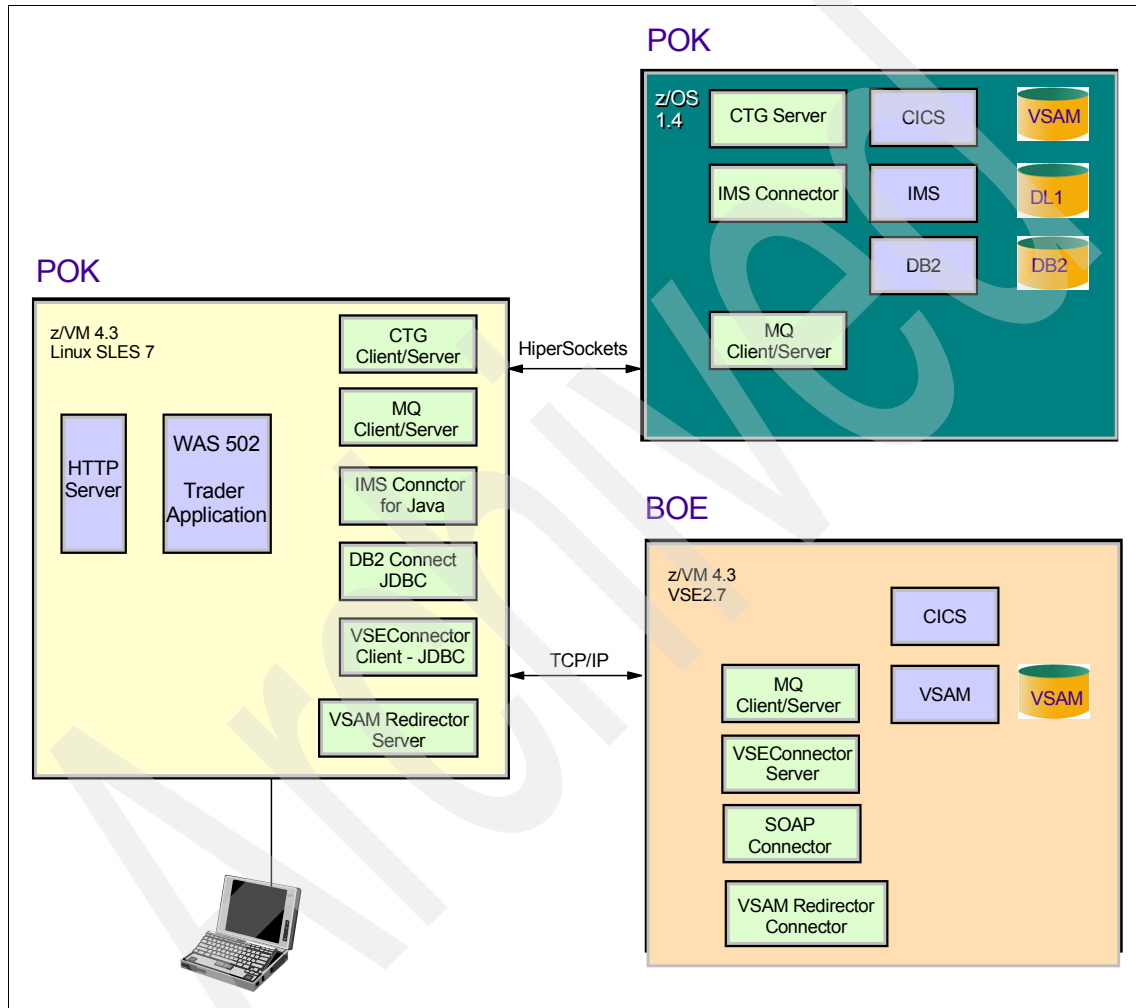


Figure 2-1 System configurations

2.1.1 Hardware

We used two zSeries servers:

- ▶ A z900 server with:
 - One LPAR (SC52) running z/OS 1.4
 - One LPAR (WTSCVMT) running z/VM 4.3

On the z/VM LPAR, we defined a virtual guest running Linux SuSE SLES 7 with the following configuration:

- 2 GB real storage
- 0 MB expanded storage
- Two 3390-3 DASD devices for root and for /opt
- 200 cylinder minidisk for swap
- HiperSockets™ connectivity to access the z/OS backend
- Remote TCP/IP connectivity to access the VSE backend

The Linux for zSeries had one OSA card defined to it, which is a HiperSocket. The TCP/IP gateway used is 9.12.9.1.

- ▶ A z900 server with one LPAR running z/VM 4.3

On the z/VM LPAR we defined a virtual guest running VSE/ESA™ V2.7.

2.1.2 Software

We used the following software products:

- ▶ Linux for zSeries
 - SuSE SLES 7 - Kernel 2.4.7 SuSE SMP #1
- ▶ Middleware Software installed on the Linux for zSeries
 - DB2 Connect™ Enterprise Edition V8.1.0-16
 - WebSphere MQ Client V5.3.0.4
 - WebSphere MQ Server V5.3.0.4 (for local testing only)
 - WebSphere Application Server V5.0.2 (build number ptf2M0325.01)
 - CICS Transaction Gateway V5.0.1
 - IMS Connect for Java V2.1.0.1
 - HTTP Server V1.3.26
 - VSE/ESA V2.7 Connector Client with PTF UQ77749
 - VSE/VSAM Redirector Server with PTF UQ77749
- ▶ z/OS Software
 - CICS Transaction Gateway V5.0.1
 - CICS TS V2.2

- IMS Connect V2.1
- IMS V8.1
- DB2 for z/OS V7
- WebSphere MQSeries V5.3.1
- ▶ VSE Software
 - VSE/ESA 2.7.1
 - CICS TS V1.1
 - MQSeries for VSE/ESA V2.1.1
 - DB2 for VSE/ESA V7.3
 - VSE/ESA v2.7 Connector Server with PTF UQ77749
 - VSE/VSAM Redirector Client with PTF UQ77749
 - VSE/ESA v2.7 CICS/SOAP Service with PTF UQ81044
- ▶ Desktop Software
 - Windows® 2000
 - Hummingbird® Exceed V8
 - WSAD IE V5.0
 - WebSphere Application Assembly Tool (part of WebSphere Application Server V5)

2.2 Frontend environment

Our frontend environment is on Linux for zSeries. We used the standard SuSE SLES 7.2 build to create this system. The only additional package that we needed was the Public Domain Korn Shell, as many of the installation scripts require this: pdksh-5.2.14-248.

We used a Linux for zSeries NFS server to hold our middleware software. We NFS mounted it from a local directory called /mnt/sw.

To run many of the configuration tools and installation wizards on Linux for zSeries, we need an X-Window server on our Windows 2000 workstation. We needed to specify the address of the X-Window server on our Windows 2000 machine in the DISPLAY environment variable on Linux:

```
export DISPLAY=9.12.6.140:0.0
```

For our X-Window server, we chose to use Hummingbird's Exceed V8.0.

There are two backend operating systems that our Linux for zSeries system will connect to: VSE and z/OS. We use WebSphere Application Server V5.0.2 on our Linux for zSeries system as our application server of choice to deploy our enterprise Web service solutions on.

2.3 Backend environments

We considered both z/OS backend and VSE backend in our connector scenarios, as shown in Figure 2-1 on page 26. We do not give a detailed description of how to set up those environments because this is outside the scope of this book. We assume that you have a system support team who installs and maintains the EIS systems including the connectors that reside on the backend environment. However, we do describe how you should customize your backend environments to support our specific connector scenarios if any changes are necessary to allow remote invocations.

2.3.1 z/OS

The z/OS backend comprises EIS environments such as CICS, IMS and DB2. It hosts connector/middleware softwares such as the CTG sever, the MQ server, and the IMS connector.

2.3.2 VSE

The VSE backend comprises EIS environments such as CICS and VSAM applications. It hosts connector/middleware software such as the VSE Connector server and the MQ server for VSE/ESA. There is no CTG middleware on this platform. The CTG connector to VSE scenario uses a CTG on the frontend Linux platform. A new CICS-based SOAP connector is provided with VSE/ESA 2.7 and CICS TS 1.1 for VSE; this allows accessing CICS transactions via the Web Services using SOAP and XML protocols.

In addition to these standard zSeries operating system connectors, VSE provides some VSE-specific connectors:

- ▶ The VSE Java-based connector, which consists of the VSE Connector Client (VCC) and VSE Connector Server (VCS). This connector provides access to all kinds of VSE file systems and functions from any kind of Java program, including WebSphere applications.
- ▶ The VSE/VSAM Redirector connector, which consists of the VSAM Redirector Client (VRC) and the VSAM Redirector Server (VRS). This connector allows transparently redirecting access to VSAM files on any remote platform.

2.4 Connector scenarios overview

We describe how to set up the appropriate resources, especially the middleware, to allow our J2EE applications (such as Trader) running in WebSphere

Application Server on Linux for zSeries to connect to backend environments running in VSE and z/OS, as shown in Figure 2-1 on page 26.

Chapter 3, “The Trader applications” on page 33, describes the Trader application.

Chapter 4, “WebSphere Application Server setup” on page 67, describes the WebSphere setup on Linux for zSeries. All of the J2EE connectors use WebSphere as their application deployment environment.

We continue by describing our connector scenarios and how to configure the infrastructure to support them as follows.

Chapter 5, “CICS J2EE connectors” on page 83, describes how to set up the WebSphere connections for CICS. We describe the configuration steps to connect to a CICS TS for VSE application and to a CICS TS for z/OS application using the CICS resource adapter.

Chapter 6, “Using SOAP to communicate with CICS” on page 119, describes the connection configuration to connect to a CICS TS for VSE application from a stand-alone Java program on Linux for zSeries.

Chapter 7, “DB2 connectors” on page 131, describes how to set up the connections for DB2. We describe how to connect a Java application running in WebSphere Application Server to connect to DB2 for z/OS using the Type 2 JDBC driver.

Chapter 8, “WebSphere MQ connectors” on page 153, describes how to set up the connections from WebSphere MQ for Linux for zSeries to WebSphere MQ on z/OS and VSE.

Chapter 9, “IMS J2EE connectors” on page 207, describes how to set up WebSphere connections for IMS. We describe how to connect a Java application running in WebSphere Application Server to IMS running on z/OS using the IMS resource adapter.

Chapter 10, “VSE Java-based connector to access VSAM data” on page 223, describes how a Java program can access VSAM data in VSE environments. We explain how to set up the VSE Java-based connectors in WebSphere.

Chapter 11, “VSE Java-based connector to access DL/1 data” on page 287, describes the usage of the VSE Java-based connector to access DL/1 data in VSE. We illustrate this with a DL/1 example.

Chapter 12, “VSE VSAM Redirector connector” on page 293, describes how to redirect VSE VSAM accesses, typically requested by existing VSE programs, to

any database or file system residing on a Java-enabled platform that can be reached via TCP/IP.

2.4.1 The Trader application scenarios

In most scenarios we use the Trader (Java) application to illustrate the use of the connectors. The Trader application was developed by the ITSO to run on WebSphere for z/OS for the redbook *WebSphere for z/OS V5 Connectivity Handbook*, SG24-7064. We reused those Java applications and ran them on the Linux for zSeries platform connecting them to Enterprise Information Systems (EIS) running on z/OS and VSE backends. The Trader application has multiple modules, each connecting respectively to CICS, IMS, JDBC, and MQ. The Trader modules are discussed in Chapter 3, “The Trader applications” on page 33.

You can also refer to Appendix B, “Additional material” on page 337, for information on how to download the programs that can be used as small demo applications to study and test the J2EE connectors.

Figure 2-2 describes the connector scenarios to the z/OS backend.

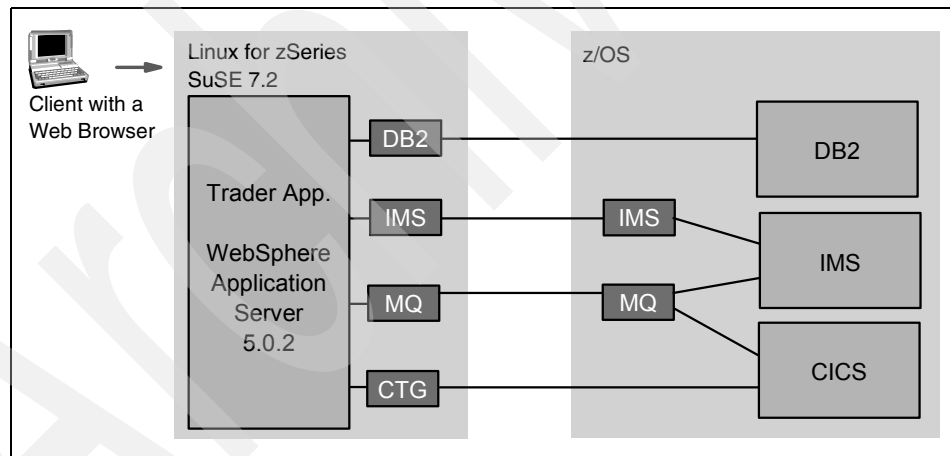


Figure 2-2 Overview of the Trader application connections to backend EISs on z/OS

Figure 2-3 describes the connector scenarios to the VSE backend.

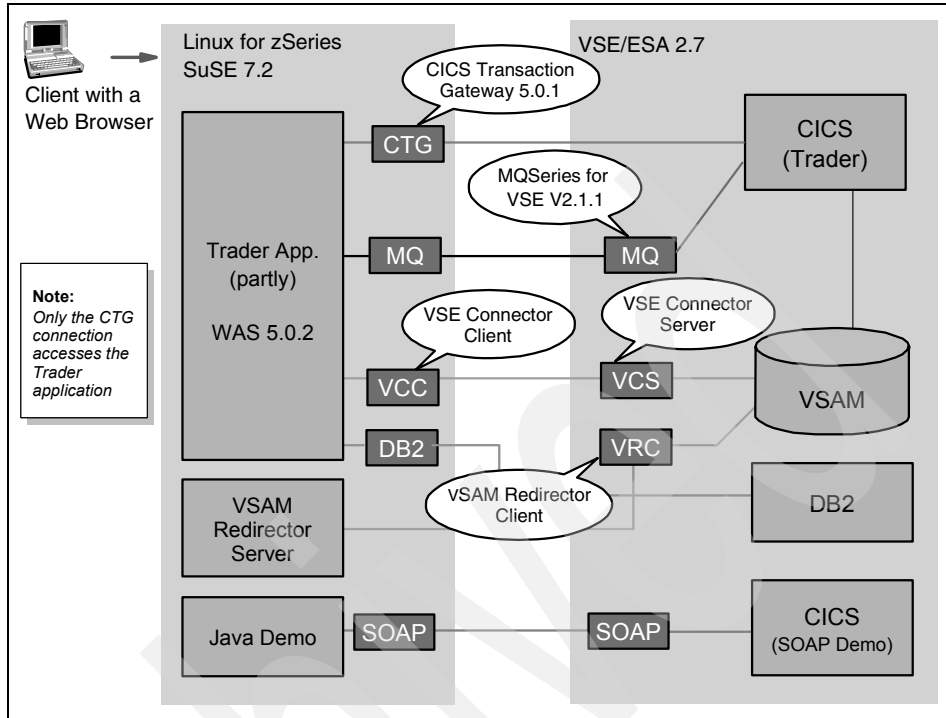


Figure 2-3 Overview of the Trader application connections to backend EIS on VSE

On the VSE backend, Trader is only involved in the CTG connection because:

- ▶ There is currently no MQ Bridge on CICS for VSE. That, however, should not stop anyone from writing their own MQ wrapper functions on CICS for VSE if they want to expose existing CICS transactions to enterprise applications in WebSphere Application Server.
- ▶ Trader does not use direct access to VSAM or SOAP. However, other sample programs demonstrating this feature are provided.

The Trader applications

In this chapter we describe the Trader applications, how to install them in the WebSphere Studio Application Developer Integration Edition (WSAD.IE), and how to deploy them to WebSphere Application Server on Linux for zSeries.

- ▶ Trader application components
- ▶ Trader data stores
- ▶ Trader frontend GUI
- ▶ Trader Web frontend architecture
- ▶ Implementing and testing Trader in WSAD.IE
 - Downloading
 - Importing MQ
 - Configuring DB2 tables
 - Running
- ▶ Deploying Trader to WebSphere Application Server
 - Installing Trader to WebSphere Application Server runtime
 - Running

3.1 Introducing the Trader application

Trader is a sample application which shows different scenarios describing how J2EE connectors can be utilised in an application running in a WebSphere Application Server and/or WSAD.IE environment.

As we already mentioned, the Trader application was developed by the International Technical Support Organization (ITSO) to run on WebSphere for z/OS for the redbook *WebSphere for z/OS V5 Connectivity Handbook*, SG24-7064; the current chapter has been lifted from the previously mentioned redbook for your convenience, and you can of course refer to the *WebSphere for z/OS V5 Connectivity Handbook* for more information on the Trader application. We reused those Java applications and ran them on the Linux for zSeries platform, connecting them to Enterprise Information Systems (EIS) running on z/OS and VSE backends.

The Trader application on Linux for zSeries has multiple modules, each connecting respectively to CICS, IMS, JDBC, and MQ on z/OS and VSE.

You can also refer to Appendix B, “Additional material” on page 337, for information on how to download the programs that can be used as small demo applications to study and test the J2EE connectors.

3.1.1 Trader application components

Trader is a very simple application, mimicking trading stocks in four different companies. It has four components:

- ▶ A backend
- ▶ A datastore
- ▶ A middle tier providing access to the backend
- ▶ A frontend, implemented as a Web application

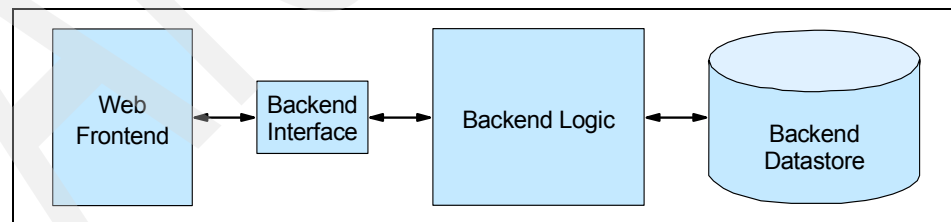


Figure 3-1 Trader application components

The Web frontend is a regular J2EE Web module. The middle tier (backend interface) is based on EJBs.

It uses the following connectors:

- ▶ The CICS ECI J2C resource adapter, providing direct access from Linux to the backend logic hosted in CICS TS, using the CTG
- ▶ The IMS J2C resource adapter, providing direct access from Linux to the backend logic hosted in IMS TS
- ▶ A JDBC connection using either straight JDBC from an EJB session or using CMP Entity EJBs, in which case the backend logic is located with the Web frontend and the backend interface and is implemented as session EJBs
- ▶ The WebSphere MQ JMS provide access to either IMS transactions via the IMS-MQ bridge or CICS terminals via the CICS-MQ bridge

We created a Trader application for each of the connectors utilized: TraderCICS, TraderIMS, TraderDB and TraderMQ.

3.1.2 Trader datastores

The datastores for all Trader applications share the same structure. Figure 3-2 describes the DB2 example used for the application TraderDB.

TRADER.COMPANY			
Column name	Type	Len	Nulls
COMPANY	CHARACTER	20	No
SHARE_PRICE	REAL	4	Yes
UNIT_VALUE_7DAYS	REAL	4	Yes
UNIT_VALUE_6DAYS	REAL	4	Yes
UNIT_VALUE_5DAYS	REAL	4	Yes
UNIT_VALUE_4DAYS	REAL	4	Yes
UNIT_VALUE_3DAYS	REAL	4	Yes
UNIT_VALUE_2DAYS	REAL	4	Yes
UNIT_VALUE_1DAYS	REAL	4	Yes
COMM_COST_SELL	INTEGER	4	Yes
COMM_COST_BUY	INTEGER	4	Yes

TRADER.CUSTOMER			
Column name	Type	Len	Nulls
CUSTOMER	CHARACTER	60	No
COMPANY	CHARACTER	20	No
NO_SHARES	INTEGER	4	Yes

Figure 3-2 DB2 table definitions for TraderDB application

The datastores used by the different Trader applications are as follows:

- ▶ TraderDB uses a DB2 database as a datastore.
- ▶ TraderCICS uses a VSAM file as a datastore.
- ▶ TraderIMS uses a DL/1 database as a datastore.
- ▶ TraderMQ rely on the same backend logic as the IMS and CICS samples and uses the same datastores.

3.1.3 Trader frontend GUI

All the Trader Web modules provide the same user interactions.

Figure 3-3 shows the Trader screen flows.

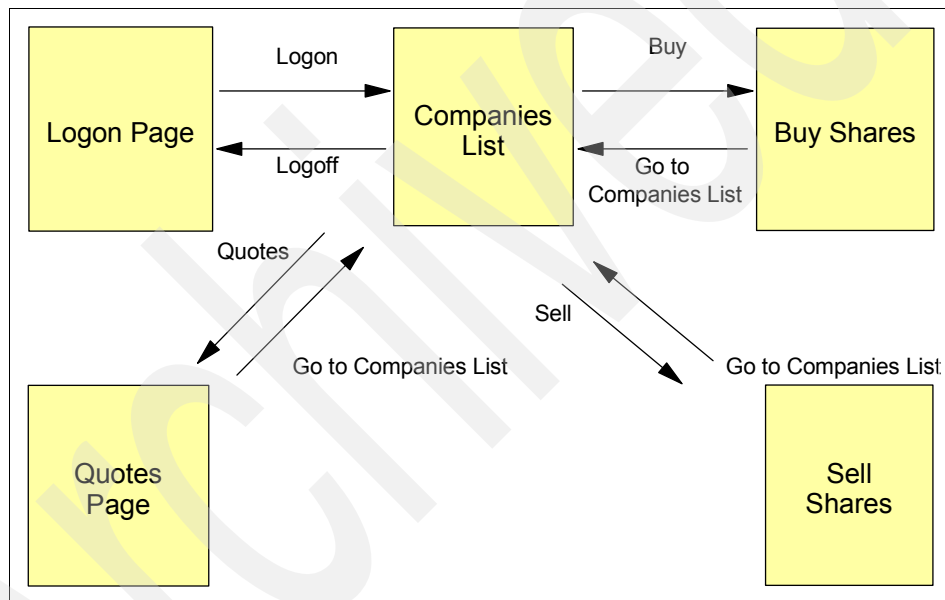
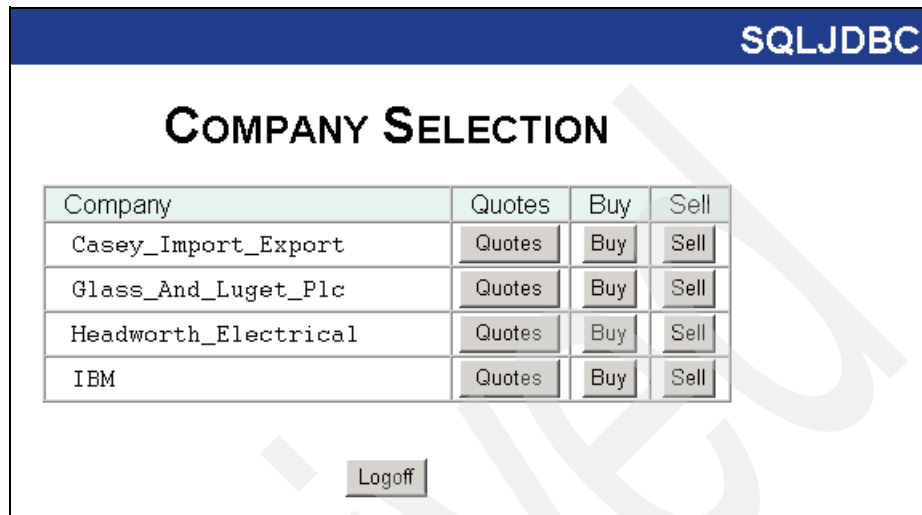


Figure 3-3 Trader screen flows

The entry page is the logon page. It provides you with a field to enter a username and one or more push buttons that will take you into the applications. The number of push buttons depends on the actual Trader application. For example, TraderMQ provides you with a choice between using CICS or IMS as the processor of the MQ messages. There is also a possibility to modify the way the connector is used. For example, TraderDB provides both straight JDBC and JDBC encapsulated in Entity EJBs using CMP connections.

If the logon is successful, a list of companies is presented on the next page (Figure 3-4).

For each company, there are push buttons to get quotes and holdings status, buy and sell shares. This list is obtained from the backend datastore.



Company	Quotes	Buy	Sell
Casey_Import_Export	Quotes	Buy	Sell
Glass_And_Luget_Plc	Quotes	Buy	Sell
Headworth_Electrical	Quotes	Buy	Sell
IBM	Quotes	Buy	Sell

Figure 3-4 Trader companies list

Clicking the **Logoff** button will take you back to the logon page of the Trader application.

Clicking the **Buy** or **Sell** buttons will take you to a page with a field where you can enter the number of shares you want to buy/sell and a pushbutton to start the transaction.

When the transaction is done, the application displays the companies list again (Figure 3-4).

To see the result of a transaction, you will have to go to the quotes page. This is done by clicking the **Quotes** button on the Companies list page.

Figure 3-5 shows the Quotes page.

User		Comission Cost	
User	asd	For Selling	2
Company	Glass_And_Luget_Plc	For Buying	2
Share Values		Number of shares held	100.0
Now	19.0		
1 week ago	17.0	Value of shares held	1900.0
6 days ago	22.0		
5 days ago	20.0		
4 days ago	16.0		
3 days ago	20.0		
2 days ago	25.0		
1 day ago	22.0		

Go to companies list

Figure 3-5 The company quotes page

3.1.4 Trader Web frontend architecture

The overall architecture of the Trader Web applications is presented in Figure 3-6. It is a classic MVC approach.

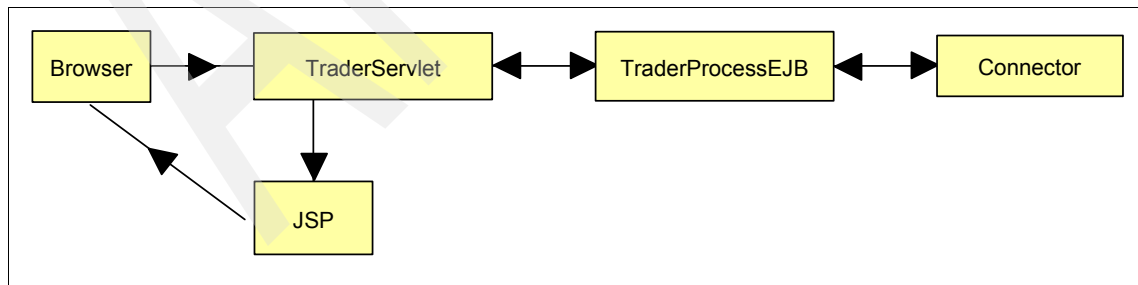


Figure 3-6 Component diagram of the Trader Web applications

The TraderServlet contains the control logic, providing a method for each user interaction. We decided—for time constraints—not to use the Command pattern. Our advice is to use the Command pattern for applications larger than Trader. It gives a better separation of control and command logic, which makes the application easier to maintain.

The TraderProcessEJB contains the frontend business logic: Buy, sell, getCompanies, etc. The implementation is split in two: An interface (TraderProcess), which is what is used and seen by the TraderServlet; and the actual implementation, which is dependent on the connector used.

The JSPs format the output for the browser.

To simplify the implementation of the different variations of Trader, we used the simplified class diagram, as shown in Figure 3-6.

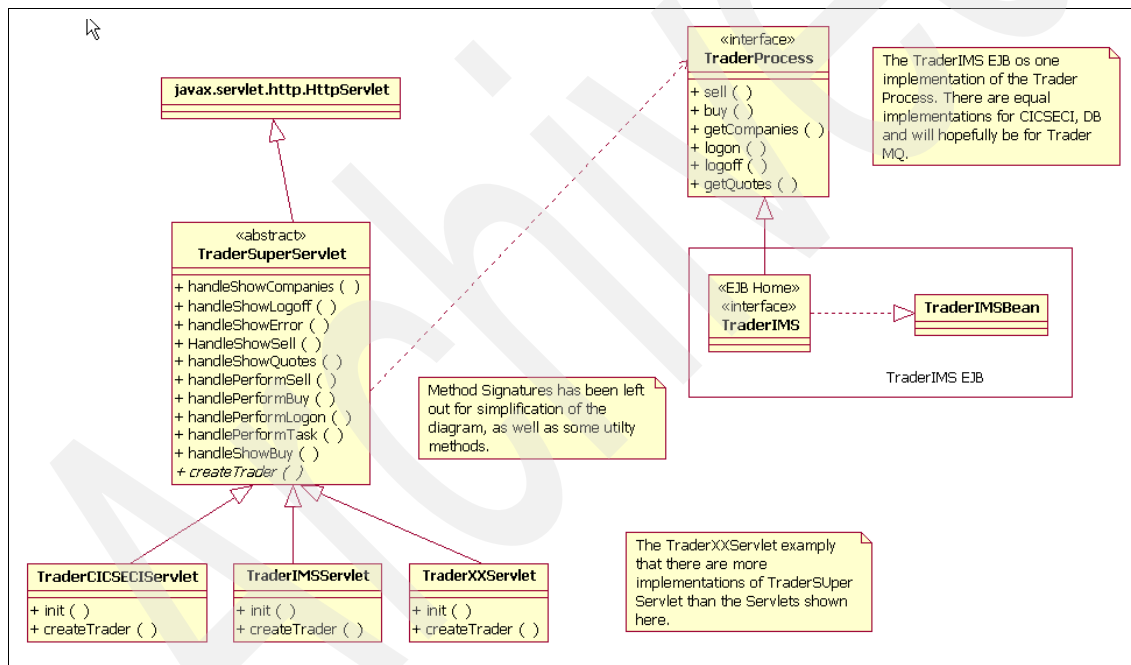


Figure 3-7 Trader class diagram - simplified overview

The TraderSuperServlet contains all the control and command logic of the application. The only method implemented by the actual servlets is a method for creating the TraderProcess instance (`createTrader`) and the `init()` method of the servlet, which initializes the text-strings to be used for the construction of the URLs in the applications and to display what type of connector is used.

The TraderProcess implementations are all specialized according to the connectors being used.

3.1.5 Packaging

The different Trader applications are packaged in EAR files:

- ▶ TraderCICS2004.ear
- ▶ TraderIMS2004.ear
- ▶ TraderMQ2004.ear
- ▶ TraderDB2004.ear

Figure 3-8 shows the Trader ear file contents.

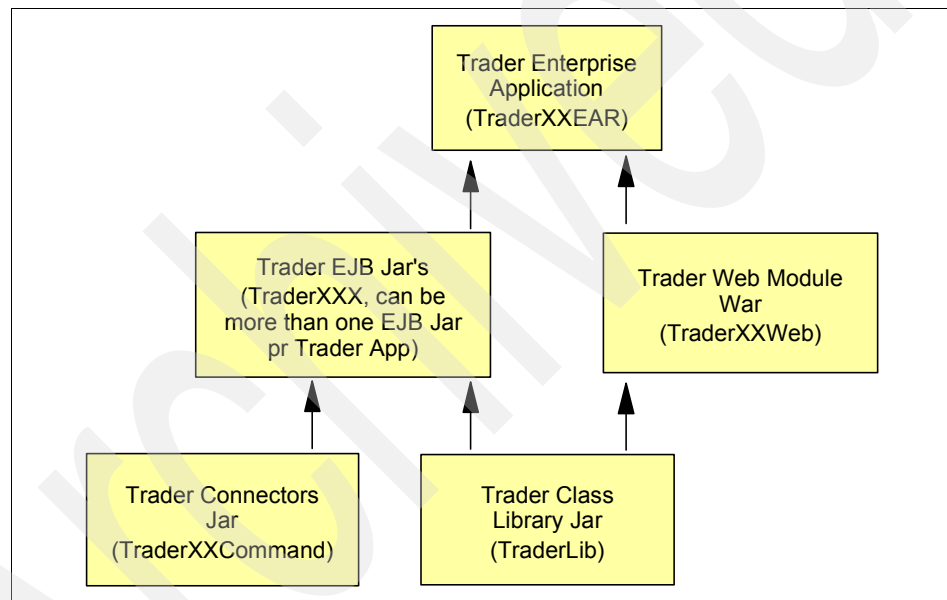


Figure 3-8 Trader Ear file contents

The TraderLib.jar is shared between all the Trader applications. It contains the TraderSuperServlet, TraderProcess, and some utility classes.

The Trader Web module contains the servlet(s) sub-classed from TraderSuperServlet and the JSPs used in the Web-application. Because of the way J2EE 1.3 works it is impossible to share the JSPs the way done with the TraderSuperServlet, so each Web module contains its own copy of the JSPs. The Logon.html is different for each Trader application, the JSP's are not.

The Trader EJB jars contain the EJB used by the servlets. In the case of the TraderDB it also contains the EJBs used for communication with the database and the business logic implementation.

The Trader Connectors jar contains the Web service that provides access to the J2C connectors, including the EJB that connects to the J2C connector and the generated classes used for getting and setting data on the J2C transaction object(s). In CICS this is the ECI CommArea; in IMS they are the InputHandler and OutputHandler objects.

3.1.6 Dependencies

To be able to deploy and run, each Trader application depends on some external resources. These resources are specified, if possible, by their JNDI name and a type.

TraderMQ dependencies

TraderMQ requires the following external resources:

- ▶ jms/TraderQCF, WebSphere MQ JMS provider connection factory
- ▶ jms/TraderCICSReqQ, JMS request destination for CICS
- ▶ jms/TraderCICSRepQ, JMS reply destination for CICS
- ▶ jms/TraderIMSReqQ, JMS request destination for IMS
- ▶ jms/TraderIMSRepQ, JMS reply destination for IMS
- ▶ jms/TraderProcessQ, JMS postprocessing destination for the MDB case
- ▶ TraderMQCICSListener, MDB EJB listener (When a message is received on a Queue listened to, the corresponding MDB is executed.)
- ▶ TraderMQIMSListener, MDB EJB Listener

Depending on your local environment you might also need to define JAAS user ID and password to be used by the MQ Bridge.

If you want TraderMQ to work you need to set up WebSphere MQ for z/OS, the proper queues, and MQ Bridge for CICS or IMS. Details on this are provided in Chapter 8, “WebSphere MQ connectors” on page 153.

TraderDB dependencies

TraderDB requires the following external resources: jdbc/TraderDB2, JDBC Datasource (not specific to be DB2 even if the name indicates so). Note that TraderDB can support any JDBC-compliant database as there is no DB2-specific code in the connector implementation.

TraderCICS dependencies

TraderCICS requires the following external resources:
itso/cics/eci/j2ee/trader/TraderCICSECICCommandCICSECIServiceTraderCICSECICCommandCICSECIPort, an ECI J2C connector to CICS.

TraderIMS dependencies

TraderIMS requires the following external resources:
itso/ims/j2ee/trader/TraderIMSCommandIMSServiceTraderIMSCommandIMSPort, an IMS J2Cconnector to IMS.

Restriction: Not all of the necessary resources have been externalized in the Web deployment and EJB deployment descriptors. Some of the resources have to be looked up directly and not indirectly using the “java:comp/env” context, which also means that the possibility of setting up transaction control and redirection is limited.

3.1.7 Trader connector paths

Figure 3-9 gives a good overview of the different connector paths implemented in Trader.

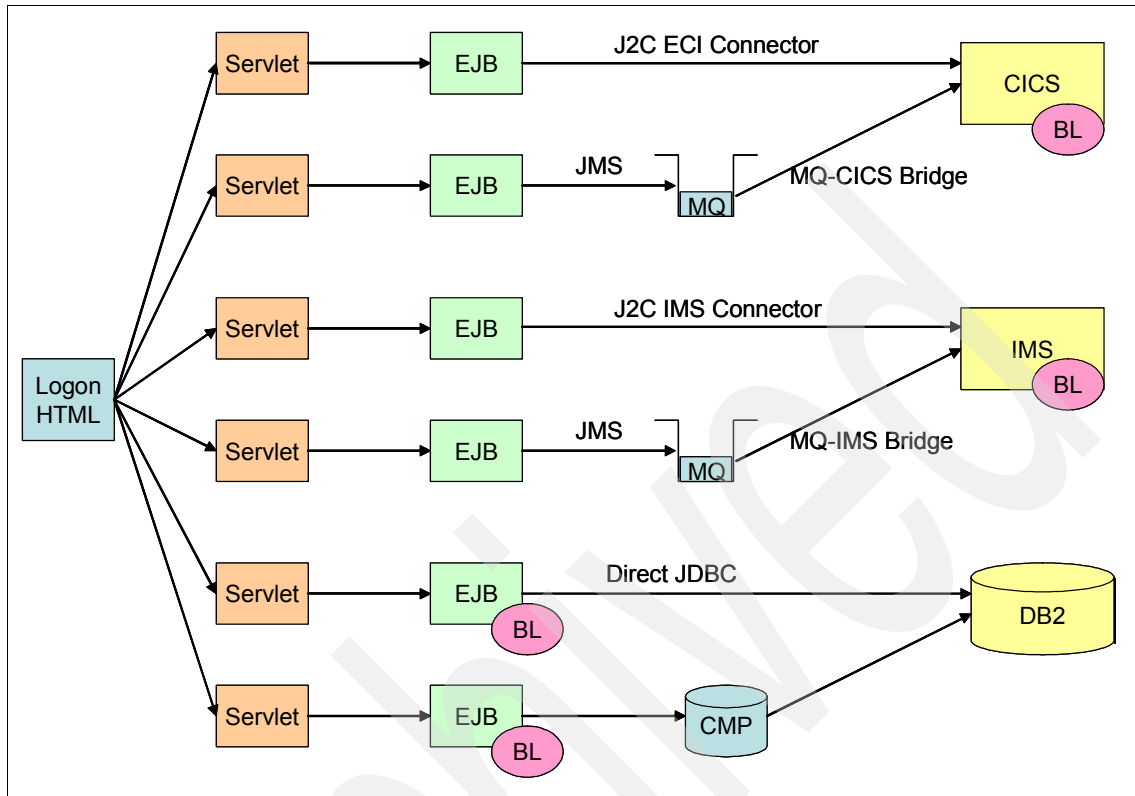


Figure 3-9 Trader connector paths

The following describes the connector-specific issues:

- ▶ CICS ECI Connector:

Uses the J2C CICS ECI Connector. The connector uses the CICS Transaction Gateway (CTG) Java client, and the code for accessing the J2C ECI connector is generated by WebSphere Studio Integration Edition. The generated code consists of a Web-service implemented as an EJB and classes for setting and getting information in the ECI CommArea, based on the object definitions used in the CICS programs.
- ▶ IMS Connector:

Uses a J2C IMS Connector. It works the same way as the CICS ECI Connector and the code is generated the same way.
- ▶ JDBC Connector:

The JDBC connector does not have a backend CICS/IMS transaction with business logic, so we had to implement this logic in the WebSphere

application. This was done using EJBs. The data resided in a DB2 for z/OS database.

Data is accessed in two ways:

- Using traditional straight JDBC from a session EJB
- Using Container-Managed Persistence (CMP) Entity EJBs

► WebSphere MQ

Instead of going straight to IMS/CICS from the application, there is the option of using WebSphere MQ. The TraderMQ application sends a message via WebSphere MQ to the backend business logic in CICS/IMS (the TRADERBL program). The message receiver is the MQ bridge—either the IMS-MQ bridge or the CICS-MQ bridge. When the transaction is completed in either IMS or CICS, the reply is returned via WebSphere MQ to the Trader application in WebSphere. This is a quasi-synchronous solution to frontend any traditional business logic in CICS or IMS.

There is an option of using a Message Driven Bean (MDB) EJB as the receiver in the Trader Web application instead of a session EJB querying the reply queue. When using this option check the MDB box on the TraderMQ logon panel and start the message listener ports on the server. When the MDB listeners are enabled, the normal TraderMQ scenarios will not work (non-MDB case), as the MDB listener will pick up the messages from the TRADER.CICS.REPLYQ or TRADER.IMS.REPLYQ no matter if the check box was marked or not. The XA (two-phase commit) feature has to be enabled on the WebSphere MQ connection factory for this to work.

Important: If the message listeners are started when TraderMQ is executed and the MDB box is *not* checked on the logon screen, TraderMQ will wait for the message to come back from CICS or IMS; it will never get the reply (you have to click the Abort button). The reason for this is that the MDB already picked up the message from the TRADER.CICS.REPLYQ (or IMS) reply queue and put it on the TRADER.PROCESSQ, but as the MDB check box was not marked, the EJB business logic does not receive the message from the TRADER.PROCESSQ.

Restriction: Trader was not implemented with the purpose of being a fully production-qualified application.

Because of this, the screen flow is based on the need to be there, the fault tolerance is limited, the application cannot be expected to run in parallel without flaws, all resources have not been externalized using the “java:comp/env” context, thereby lacking transactional control, and part of the implementation is not in compliance with best practices and recommended implementation patterns. However, the application(s) will assist on verifying that a WebSphere Connector has been set up properly, as an example of how a WebSphere Connector can be used in an application.

3.1.8 Downloading Trader application modules

You can download the various Trader applications from the following Web site:

<http://www.redbooks.ibm.com/redbooks/sg247042>

The applications are provided in a zip file that includes both the EAR file to use for deployment to WebSphere Application Server and a zip file with the corresponding resources (projects, source files, server configurations, etc.) to use for importing into WSAD.IE.

The EAR file does not contain any source files and will not restore the WSAD projects necessary for WSAD.IE to compile and run the application.

See also Appendix B, “Additional material” on page 337.

3.2 Running Trader in WSAD.IE test environment

If you skip running an application in the WebSphere Studio Application developer Integration Edition (WSAD.IE) test environment, all you have to do is to download the application module as described in 3.1.8, “Downloading Trader application modules” on page 45.

When you download the Trader application and extract the .zip file, you can see the following folders:

- ▶ TraderCICS
- ▶ TraderIMS
- ▶ TraderDB
- ▶ TraderMQ

Each folder contains the ear file of the given Trader application and also the zipped workspace. Create a directory that will be the workspace of your Trader

application. We suggest that you use separate workspaces for each application. Unzip the workspace zip file into that workspace directory. You will have a directory that WebSphere Studio Integration Edition will use for the project. All the project folders are there, but you have to import the projects to let WebSphere Studio Integration Edition know about them. Use the following steps to restore your WebSphere Studio Integration Edition workspace.

1. Start WebSphere Studio Integration Edition and specify the repository where you extracted the .zip file, as shown in Figure 3-10.

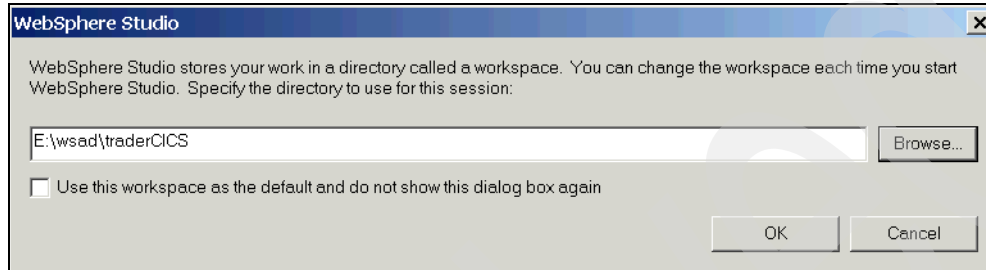


Figure 3-10 WebSphere Studio Integration Edition startup window

2. An empty workspace will appear. You can import using **File** → **Import...** and choose **Existing Project into Workspace**, and click **Next**. In the Project contents field, navigate to the directory that is located under the directory from which you have extracted the zip file contents, and select the project to import. You should import all the projects with names that start with Trader, and also the Server project, but you can import only one project at a time. Then click **Finish**.
3. You have to repeat the import several times to import all of the projects needed by Trader.
4. In the case of the TraderCICS project, you have to import the RAR file to create CICS connectors. We provide an ECICConnector project in the workspace, but the location of our connector is not necessarily the same as your WebSphere Studio Integration Edition installation. To complete this task, do the following:
 - a. Select **File** → **Import...** and choose the RAR file, and click **Next**.
 - b. In the Connector file field, navigate to your WebSphere Studio installation and look for your resource adapters, for example specify D:\Program Files\IBM\WebSphere Studio\resource adapters\cicsecri.rar file. Specify New Project Name as ECICConnector (Figure 3-11).

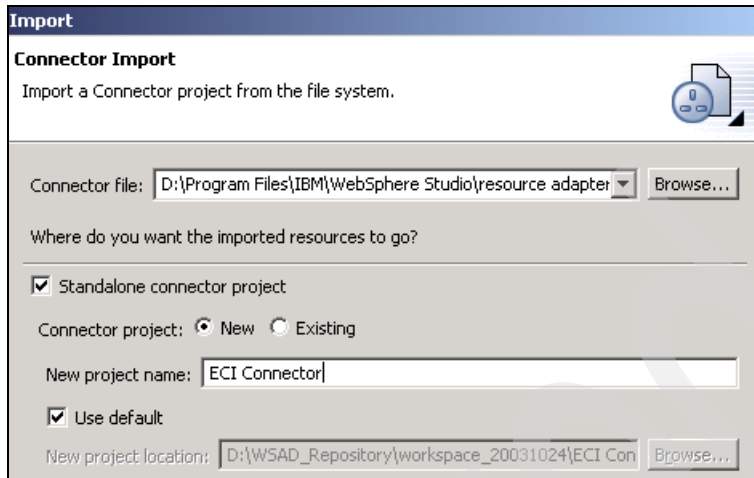


Figure 3-11 Importing ECI J2C connector to WebSphere Studio Integration Edition

- c. Click **Finish**.
5. In the case of the TraderIMS application, do the same step for IMS connector. You have to choose `ims.rar` as the connector file name and specify `IMSConnector` as the new project name.
6. If any tasks or errors remain in the Tasks field, you have to rebuild project, then the tasks go away. You can do this step from the **Project** → **Rebuild All** menu. If you get a broken link message, make sure that the Web project references the lib project. Right-click the Web project and select **Properties**. Select the **Java build path** → **Order and Export** tab and select **TraderLib**. Then rebuild the project.

3.2.1 Configuring the test server in WSAD.IE

This section shows how to configure the test environment to run the Trader application. We provide server definitions in the workspace; however, you have to verify that the resource definitions in the server projects match your environment.

In case you want to use your own server definitions, follow the instructions below.

To configure the test environment, do the following steps:

1. Open Server Perspective. As shown in Figure 3-12, at first you click (1) to open a new perspective, and choose **Server**. Now you can see the Server Perspective and you can go directly to Server Perspective by clicking (2).

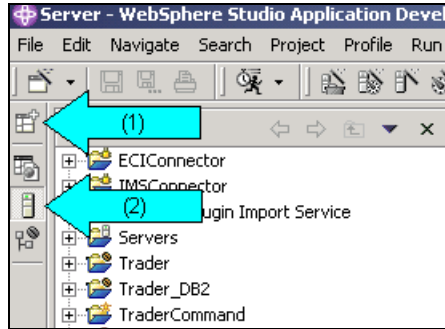


Figure 3-12 Opening Server Perspective

2. At the lower left of the window, you can see the Server Configuration panel. Right-click **Servers**, and proceed with **New** → **Server and Server Configuration**. Specify **TraderServer** in the Server name field, and other options are left as defaults. Then click **Finish**.
3. Double-click **TraderServer**, and the configuration panel opens. First you have to change to Security tab, and create three JAAS authentication entries.
 - LocalUser: Specify your local administrator ID and password. This is used as a JMS user authentication alias.
 - TraderDB2User: Specify your DB2 user ID and password to get to DB2.
 - TraderUser: Specify your CICS and IMS user ID and password. This alias is used as authentication in the connection factory of J2C.

As a result, you can see the aliases as shown in Figure 3-13.

JAAS Authentication Entries:		
Alias	User ID	Description
LocalUser	TOT134	
TraderDB2User	db2admin	
TraderUser	java1	

Figure 3-13 JAAS authentication alias

Setting up CICS/IMS connection factories

These activities depend on which Trader application you try to set up.

- ▶ In case of TraderCICS create CICS ECI J2C connection factory on the J2C tab. In the Node setting, you can see J2C Resource Adapters. Choose **Add...** and you can select **ECIConnector** in Resource adapter name field, then click **OK**.

Next you have to add connection factory. At first you select **ECIConnector**. Second, you click the **Add...** button in the J2C Connection Factory field, as shown below, to Resource adapter settings. The Name and JNDI name fields should be specified as follows:

```
itso/cics/eci/j2ee/trader/TraderCICSECICommandCICSECIServiceTraderCICSECICommandCICSECIPort
```

The container-managed authentication alias should be `TraderUser`. Then click **OK**.

Now you can see the Resource properties of the CICS ECI connector. As shown in Figure 3-14 on page 50, specify server definitions if you have a CICS Trader application. We specified Server name, connection URL, and server port.

- ▶ In the case of TraderIMS you have to add IMSConnector in the same way. In case of TraderRRS, you have to add both.

In case of the TraderIMS application do the same steps 5 and 6 as for the IMS Connector. The Name and JNDI name fields of J2C connection factory should be specified as follows:

```
itso/ims/j2ee/trader/TraderIMSCommandIMSServiceTraderIMSCommandIMSPort
```

For Resource properties, we specified the values `HostName`, `PortNumber`, and `DataStoreName`.

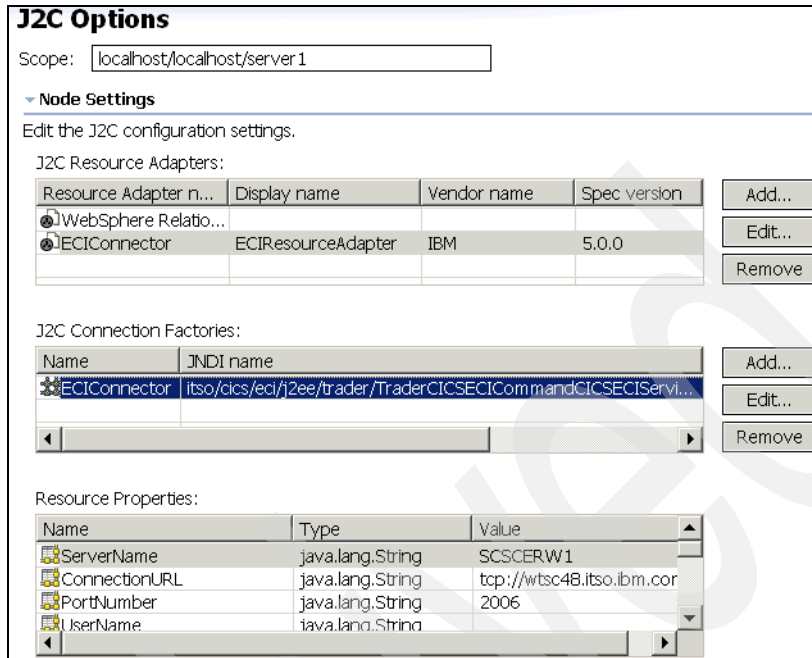


Figure 3-14 J2C connection factory settings

Setting up DB2 datasources

To do this?

- ▶ In the case of the TraderDB application switch to the Datasource tab. Now you have to configure the datasource to connect to DB2. Select **Default DB2 JDBC Provider** and **Add** new datasource. The important thing in this wizard is checking on how to use this datasource from container-managed persistence (CMP) field.

We used the following parameters to configure the DB2 datasource:

- Name: TraderDB2.
- JNDI name: jdbc/TraderDB2.
- Container-managed authentication alias: DB2User.

You may use any defined name here. Go to the Security tab and define an alias that is valid for the given system you want to access.

- We checked the “Use this datasource from container-managed persistence (CMP)” field.
- Database name: DB4B (in resource properties).

Running TraderDB2 also requires setting up DB2 Connect. We installed DB2 Connect Personal Edition. After installation you have to use **Set-up tools** → **Configuration Assistant** to configure the connection to the host. We used the following values:

- Protocol: TCP/IP
- Host name: wtsc48.itso.ibm.com®
- Portnumber: 37720
- Database name: DB4B
- Operating system: OS/390® or z/OS
- System name: SC48
- Security options: Server authentication

Setting up JMS environment

In order to run the TraderMQ application in the WebSphere Studio Integration Edition test environment, you need to define the following:

- ▶ WebSphere MQ connection factory
- ▶ Queue destinations
- ▶ Message listener ports
- ▶ Queues and connections and other z/OS host definitions

For the first three items, you need to access the Admin Console of the imbedded server. Open the server configuration and look at the Configuration tab. Click the **Enable administration console** check box and restart the server. When it restarts, open a Web browser in WebSphere Studio Integration Edition and navigate to the Admin Console, as follows:

```
http://localhost:9090/admin/
```

Log in and open up Resources. Go to WebSphere MQ JMS Provider to define the Connection Factories and Queue Destinations.

We defined the following under Connection Factories:

- ▶ Name: TraderQCF.
- ▶ JNDI Name: jms/TarderQCF.
- ▶ Container-Managed Authentication Alias: Select what you defined under J2C Authentication Data Entries.
- ▶ Queue manager: MQ4C (your z/OS Queue Manager name).
- ▶ Host: wtsc50.itso.ibm.com (your host DNS name).

- ▶ Port: 1561 (where your Channel Initiator is listening on z/OS).
- ▶ Channel: TRADER (the SVRCONN channel you defined on z/OS).
- ▶ Transport type: CLIENT (we are accessing MQ remotely, not from z/OS).
- ▶ Message Retention: Enabled.
- ▶ XA: Enabled.

We defined the following queue destinations:

- ▶ TraderCICSReqQ, TraderCICSRepQ, TraderIMSReqQ, TraderIMSRepQ and TraderProcessQ as Names
- ▶ JNDI Name: The same as the Names with jms/ (for example, jms/TraderIMSRepQ)
- ▶ Persistence and Priority: Queue defined
- ▶ Expiry: Application defined
- ▶ Base queue name: TRADER.CICS.BRIDGEQ, TRADER.CICS.REPLYQ, TRADER.IMS.BRIDGEQ, TRADER.IMS.REPLYQ and TRADER.PROCESSQ, respectively, to the destination names above
- ▶ Base Queue Manager name: MQ4C (your WebSphere MQ manager on z/OS)
- ▶ Integer, decimal and floatingpoint encoding: Normal
- ▶ Target client: MQ for the request queues (TraderCICSReqQ and TraderIMSReqQ, hence the target queue is on z/OS) and JMS for the rest, as the target is our WebSphere JMS environment on Windows
- ▶ Queue Manager Host: wtsc50.itso.ibm.com
- ▶ Port: 1561
- ▶ Server Connection Channel Name: TRADER (the SVRCONN channel you defined on z/OS)

In Example 3-1 we show you the server connection channel definition we used on z/OS for TraderMQ running on WebSphere Studio Integration Edition.

Example 3-1 Server connection channel definition used for TraderMQ

```
CHANNEL (TRADER)
CHLTYPE (SVRCONN)
QSGDISP (QMGR)
TRPTYPE (TCP)
DESCR (Channel for serving from Windows/WSAD)
SCYEXIT ( )
SCYDATA ( )
MSGEXIT ( )
```

MSGDATA()
SENDEXIT()
SENDDATA()
RCVEXIT()
RCVDATA()
PUTAUT(DEF)
MCAUSER()
KAINT(AUTO)
ALTDATE(2004-02-11)
ALTTIME(15.17.00)
SSLCAUTH(REQUIRED)
SSLCIPH()
SSLPEER()
MAXMSGL(4194304)

We defined the following listener ports. Open your Servers—Application Servers—server1 and scroll down to Additional Properties: Message Listener Service and click it, and then click **Listener Ports**. Select **New** and define two message listeners. We used the following values to define their properties:

- ▶ Name: TraderMQCICSListener and TraderMQIMSLListener.
- ▶ Initial state: Stopped. (This is important. When you later test the MDB case, start the message listener manually.)
- ▶ Connection factory JNDI name: jms/TraderQCF.
- ▶ Destination JNDI name: jms/TraderCICSRepQ and TraderIMSRepQ, respectively, for the two listeners. (This defines on which queues the MDB listens.)

Troubleshooting

If you made an error somewhere, sometimes it is hard to find it because we are dealing with a distributed environment. This can be eased with a tool that can be installed on the Windows PC. The tool can be downloaded from:

<http://www-306.ibm.com/software/integration/support/supportpacs/category.html#cat2>

It is WebSphere MQ SupportPac™, called IH03: WBI Message Broker V5 - Message display, test and performance utilities. In order to use that you need the WebSphere MQ Client for Windows, which can be found at:

<http://www-306.ibm.com/software/integration/support/supportpacs/product.html#wmq>

The name of the SupportPac is MACV. Download and install the package. This provides the connectivity to the Queue Manager running on z/OS. After installation navigate to the MQ client “bin” directory, in our case:

```
F:\Program Files\IBM\WebSphere MQ\bin\
```

Open a Command window. Issue two commands, as in Example 3-2. The first command (SET with the parameters on one line) defines the MQSERVER environment variable with server connection channel name (TRADER), the protocol to be used (TCP), the host name (wtsc50.itso.ibm.com) and the port number (1561). The second command connects to the Queue Manager and shows you whether the connection was successful. In our case we connected to Queue Manager MQ4C.

Example 3-2 Testing connection to Queue Manager on z/OS

```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
```

```
F:\Program Files\IBM\WebSphere MQ\bin>SET
MQSERVER=TRADER/TCP/wtsc50.itso.ibm.com(1561)
```

```
F:\Program Files\IBM\WebSphere MQ\bin>amqscnxc
Sample AMQSCNXC start
Connecting to the default queue manager
with no client connection information specified.
Connection established to queue manager MQ4C
Sample AMQSCNXC end
```

After validating the connection from the same command window navigate to the directory where you installed the IH03 SupportPac. Invoke the **rfhutilc** command and the window shown in Figure 3-15 on page 55 will appear.

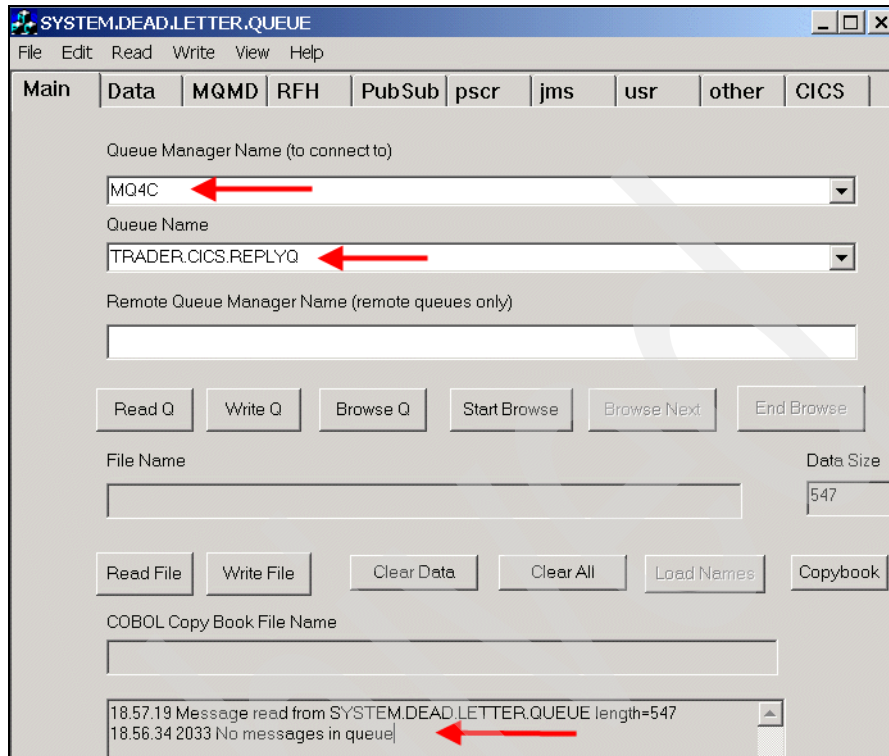


Figure 3-15 *rfhutilc* command window

The output displayed in Figure 3-15 on page 55 already shows that we executed some commands. The important fields are the Queue Manager name (MQ4C) and the queue name (TRADER.CICS.REPLYQ). After specifying these click the **Read Q** button and you may see the number of bytes read from the queue or you can see that there are no messages on the queue. In our case we read 547 bytes from SYSTEM.DEAD.LETTERQ. Now you can click the **Data** tab and you see the message in the queue, as shown on Figure 3-16 on page 56.

```

SYSTEM.DEAD.LETTER.QUEUE
File Edit Read Write View Help
Main Data MQMD RFH PubSub pscr jms usr
Message Data (547 bytes)
00000000 DLH ... ..TRAD C4D3C840 00000001 00000146 E3D9C1C4
00000016 ER. IMS. B RIDGEQ C5D94BC9 D4E24BC2 D9C9C4C7 C5D84040
00000032 40404040 40404040 40404040 40404040
00000048 MQ4C 40404040 40404040 40404040 D4D8F4C3
00000064 40404040 40404040 40404040 40404040
00000080 40404040 40404040 40404040 40404040
00000096 40404040 40404040 40404040 00000311
00000112 ...4MQIM SVS ... 000001F4 D4D8C9D4 B0E5E240 00000013
00000128 MQ4C D4D8F4C3 40404040 40404040 40404040
00000144 2004 40404040 40404040 40404040 F2F0F0F4
00000160 02141553 2503..00 F0F2F1F4 F1F5F5F3 F2F5F0F3 0177F0F0
00000176 ???@?? ?@000000 3F3F3F3F 7C3F3F3F 3F3F7C7C 7C7C7C7C
00000192 00000000 00000000 7C7C7C7C 7C7C7C7C 7C7C7C7C 7C7C7C7C
00000208 00000000 00000000 7C7C7C7C 7C7C7C7C 7C7C7C7C 7C7C7C7C
00000224 00000000 00000000 7C7C7C7C 7C7C7C7C 7C7C7C7C 7C7C7C7C

```

Figure 3-16 rfhutilc message data output

This window shows you the message data on the queue, and on the right side of the window you can select different output formats. The MQMD tab shows you other important message fields, like the correlation ID. The IH03 package comes with a description and other useful utilities that you may use to test your connection outside your WebSphere environment.

3.2.2 Runing Trader application in WSAD.IE test environment

You have configured the test environment in WebSphere Studio Integration Edition. From the Server Perspective, you can start your defined server, and when the server is ready for e-business, you can access the Trader application from an external browser or by simply selecting **Run on Server...** by right-clicking **Logon.html** from the J2EE Navigator view of J2EE the Perspective that gives you an internal browser.

In the case of the TraderMQ application you have to specify the environment variables that are used to control the codepage translation. When a message is passed from the application to MQ and IMS, the message needs to be traslated from character string to bytes array and vice versa. This is controlled by code page traslation. We defined two environment variables to be used:

- ▶ prop/Encoding_send
- ▶ prop/Encoding_receive

The first controls the codepage used for sending messages to WebSphere MQ for z/OS and should be set to cp437 for the WebSphere Studio Integration Edition embedded server, US English case. That is the Windows US English codepage, since we are sending the message from Windows to z/OS.

The second controls the codepage used to receive messages from WebSphere MQ for z/OS and should be set to cp037 for the WebSphere Studio Integration Edition embedded server, US English case. That is the z/OS US English codepage, since we are receiving the message from z/OS.

The variables can be defined by going to the J2EE perspective and clicking the EJB Modules and then double-clicking the TraderMQIMS EJB. This will open the deployment descriptor. Click the **Beans** tab and select the **TraderMQIMS** bean, as shown in Figure 3-17 on page 57.



Figure 3-17 TraderMQIMS bean environment variables - 1

Click **Add** and enter the variable names and values as shown in Figure 3-18 on page 58. Save the deployment descriptor and restart your server. Now you can invoke the TraderMQ application from an external browser or simply select **Run on Server...** by right-clicking **Logon.html** from J2EE Navigator view of the J2EE Perspective that gives you an internal browser. If you want to test the MDBs, do not forget to start the Listener ports.

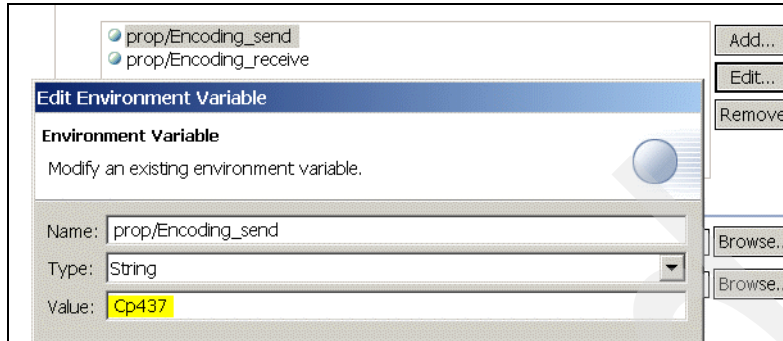


Figure 3-18 Entering the environment variables

3.2.3 Configuring DB2 tables in the test environment

When you run the Trader application in the WSAD.IE test environment, you should configure the DB2 tables in your local PC. From the DB2 command window, you can proceed with this task with DDL, which is included in the Trader application module. Example 3-3 shows the command input you should key in.

Example 3-3 Creating DB2 table in sample database

```
C:\work\db2>db2 connect to sample
C:\work\db2>db2 -tvf DB2table_trader.ddl
```

The Trader application uses two tables, the definitions of which are shown in Example 3-4.

Example 3-4 DB2 table definition of Trader application

TRADER.COMPANY

Column name	Type	Len	Nulls
COMPANY	CHARACTER	20	No
SHARE_PRICE	REAL	4	Yes
UNIT_VALUE_7DAYS	REAL	4	Yes
UNIT_VALUE_6DAYS	REAL	4	Yes
UNIT_VALUE_5DAYS	REAL	4	Yes
UNIT_VALUE_4DAYS	REAL	4	Yes
UNIT_VALUE_3DAYS	REAL	4	Yes
UNIT_VALUE_2DAYS	REAL	4	Yes
UNIT_VALUE_1DAYS	REAL	4	Yes
COMM_COST_SELL	INTEGER	4	Yes
COMM_COST_BUY	INTEGER	4	Yes

TRADER.CUSTOMER

Column name	Type	Len	Nulls
CUSTOMER	CHARACTER	60	No
COMPANY	CHARACTER	20	No
NO_SHARES	INTEGER	4	Yes

3.2.4 Running Trader application in WSAD.IE test environment

At this point, you have configured the test environment in WSAD.IE. You can run all the Trader scenarios from the test environment except the MQ-CICS Bridge portion and MQ-IMS Bridge portion. From the Server Perspective, you can start TraderServer, and when the server is ready for e-business, you can access the Trader application from the following URL or simply select **Run on Server...** by right-clicking **Logon.html** from the J2EE Navigator view of the J2EE Perspective.

[:http://localhost:9080/TraderWeb/Logon.html](http://localhost:9080/TraderWeb/Logon.html)

3.3 Deploying Trader application

After you have configured WebSphere Application and started the server, as described in Chapter 4, “WebSphere Application Server setup” on page 67, you can access the WebSphere Administrative Console (Admin Console) from a Web Browser. The typical URL of the Admin Console is:

[:http://servername:9090/admin/](http://servername:9090/admin/)

Note: ‘port’ should be your TCP/IP port number of Admin Console, which is specified when you set up WebSphere Application Server for z/OS.

When you log on to Admin Console, you can see the window shown in Figure 3-19 on page 60.



Figure 3-19 WebSphere for z/OS V5 admin console

3.3.1 Installing Trader application

You can install the Trader application for IMS, CICS, DB2, and MQ after you have configured the WebSphere resources used by the Trader application. To configure the resources in WebSphere, please see the corresponding chapters in this book.

You can install the Trader application from **Applications** → **Install New Application**. Specify `TraderXXXXEAR.ear` (where `XXXX` is the application of choice: MQ, IMS, CICS, or DB2) in the Server path field and proceed through all of the installation steps (there will be eight of them) without changing the default values to the last panel. You might have to change the target application server to deploy. When you get to the last panel, click **Apply** and **Save** to the Master Configuration.

3.3.2 Running Trader application

You can access the Trader application from a Web browser. The logon page of Trader is as follows, where `XXXX` is the name of the connect, that is, CICS:

```
http://servernameTraderXXXXWeb/Logon.html
```

The *servername* is the name of the server that you are deploying, and port is port number of HTTP transfer port of server. You can see the logon screen shown in Figure 3-20. For demonstration purposes, we are going to use the CICS example, although all of the scenarios were designed to operate the same.



Figure 3-20 Trader application logon screen for the CICS Trader application

Enter a valid user ID and click **Go**.

This will take you to the CICS Trader Servlet, as shown in Figure 3-21, where you can buy, sell, or get a quote on four fictitious companies.

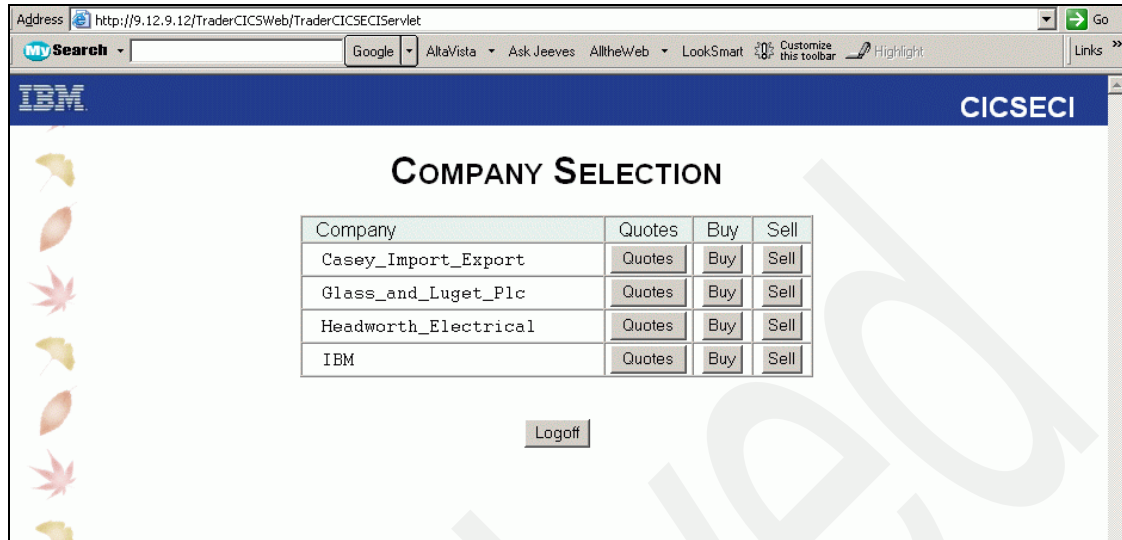


Figure 3-21 CICS Trader servlet

We are going to buy 100 shares of IBM stock, so we are going to click the **Buy** button. In Figure 3-22, we entered 100 in the number of shares to buy field, and clicked the **Buy** button.

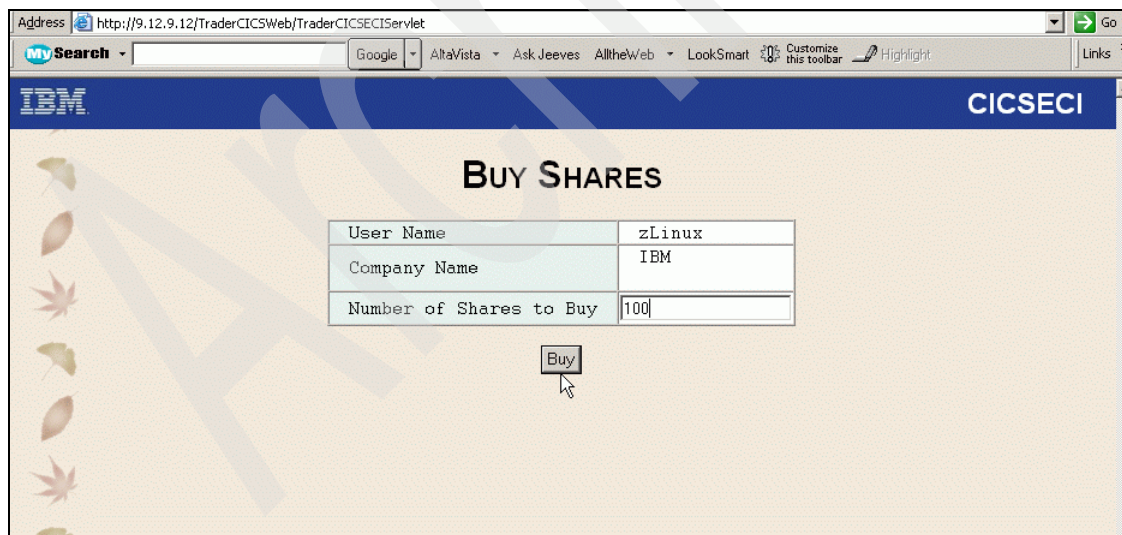


Figure 3-22 Buy 100 shares of IBM stock

We will then be presented with the original CICS Trader Servlet screen. From here we click **Quotes** and can see our purchase, as shown in Figure 3-23.

User		Comission Cost	
zLinux	IBM	For Selling	015
Company	IBM	For Buying	010
Share Values			
Now	00163.00		
1 week ago	00157.00		
6 days ago	00156.00		
5 days ago	00159.00	Number of shares held	0100
4 days ago	00161.00	Value of shares held	000016300.00
3 days ago	00160.00		
2 days ago	00162.00		
1 day ago	00163.00		

Go to companies list

Figure 3-23 Quotes - Summary of shares held

3.4 Trader DB2 table definitions

Figure 3-24 on page 64 shows how the Trader sample application is accessing DB2. The TraderDB2 JDBC application has the business logic, and the data is stored in DB2 tables. The CMP component is similar, with the only difference being the use of a Container-Managed Persistence (CMP) entity bean to access DB2.

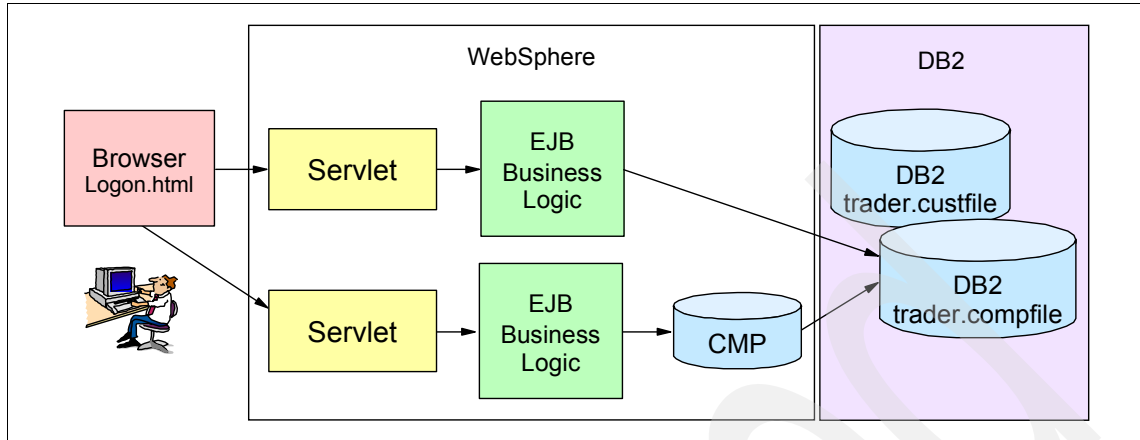


Figure 3-24 Trader application using DB2

The Trader application uses two DB2 tables. The first table is the company table and the second is the customer table. Example 3-5 shows the commands used to create the Trader database, tablespace, tables and unique indices.

Example 3-5 Trader sample database and table creation

```

DROP DATABASE TRADERDB ;
COMMIT;
CREATE DATABASE TRADERDB ;
CREATE TABLESPACE TRADERTS IN TRADERDB ;
COMMIT;
CREATE TABLE TRADER.COMPANY
( COMPANY CHAR(20) NOT NULL PRIMARY KEY,
  SHARE_PRICE REAL,
  UNIT_VALUE_7DAYS REAL,
  UNIT_VALUE_6DAYS REAL,
  UNIT_VALUE_5DAYS REAL,
  UNIT_VALUE_4DAYS REAL,
  UNIT_VALUE_3DAYS REAL,
  UNIT_VALUE_2DAYS REAL,
  UNIT_VALUE_1DAYS REAL,
  COMM_COST_SELL INT,
  COMM_COST_BUY INT )
IN TRADERDB.TRADERTS ;

CREATE, TABLE TRADER.CUSTOMER
( CUSTOMER CHAR(60) NOT NULL,
  COMPANY CHAR(20) NOT NULL,
  NO_SHARES INT,
  PRIMARY KEY (CUSTOMER,COMPANY))
IN TRADERDB.TRADERTS ;
  
```

```

CREATE TYPE 2 UNIQUE INDEX TRADER.COMPANYX1
ON TRADER.COMPANY (COMPANY) ;

CREATE TYPE 2 UNIQUE INDEX TRADER.CUSTOMERX1
ON TRADER.CUSTOMER (CUSTOMER,COMPANY) ;

COMMIT;

```

3.5 Trader VSAM file definitions

Example 3-6 shows the IDCAMS delete/define of the TraderCICS VSAM files. It also copies the example data into the file. These jobs are available in Appendix B, “Additional material” on page 337.

Example 3-6 Trader sample VSAM files delete/define

```

/???????? JOB USER=???????,CLASS=?,NOTIFY=???????,MSGCLASS=?
/*****
/*
/* THIS JOB WILL:
/*
/* 1. DELETE AND DEFINE THE TRADER 'COMPFILE'.
/* 2. REPRO THE REQUIRED DATA INTO THE 'COMPFILE'.
/*
/*****
//STEP01 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DELETE h1q.SAMPLE.COMPFILE CLUSTER
SET MAXCC = 0
DEFINE CLUSTER -
    (NAME(h1q.SAMPLE.COMPFILE) -
    INDEXED -
    TRACKS(1) -
    SHAREOPTIONS(3 4)) -
DATA -
    (NAME(h1q.SAMPLE.COMPFILE.DATA) -
    KEYS(20 0) -
    RECORDSIZE(90 90) -
    CONTROLINTERVALSIZE(4096)) -
INDEX -
    (NAME(h1q.SAMPLE.COMPFILE.INDEX))
/*
/*
//STEP02 EXEC PGM=IDCAMS

```

```

//OUTFILE DD DSN=h1q.SAMPLE.COMPFILE,DISP=OLD
//INFILE DD DSN=tradercodata.txt,DISP=OLD
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
REPRO INFILE(INFILE) OUTFILE(OUTFILE)
/*

```

3.6 Trader CICS Transaction Gateway usage

Figure 3-25 provides an overview of how our Trader application uses CTG. The figure shows a remote connection to a CICS system running on z/OS.

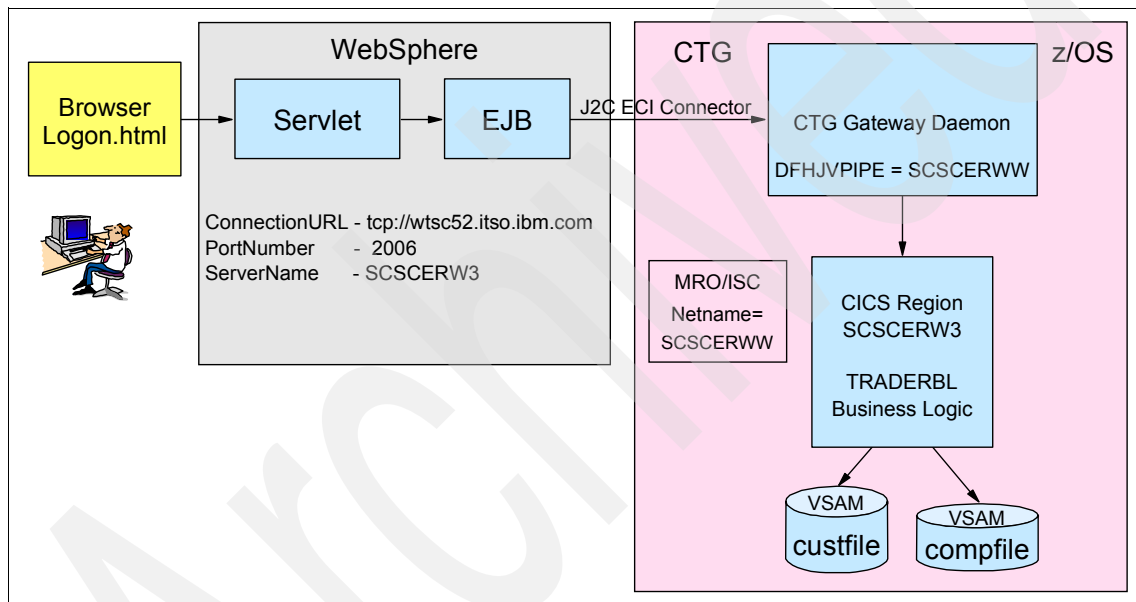


Figure 3-25 Overview of Trader application using CTG

3.7 WebSphere Studio Integration Edition hints and tips

For developing an application with IMS connectors, you have to update WebSphere Studio Integration Edition to Version 5.0.1, and you also have to apply Interim fix 004 (which is known as the IMS Connector fix).



WebSphere Application Server setup

In this chapter we describe the WebSphere Application Server setup including the prerequisite product WebSphere MQ.

We describe the following:

- ▶ Planning for WebSphere Application Server setup
- ▶ Installing WebSphere MQ client for Linux for zSeries
- ▶ Installing WebSphere Application Server

4.1 Planning for WebSphere Application Server setup

IBM WebSphere Application Server Version 5 offers a world-class infrastructure for the next phase in open e-business platforms. As the foundation of the WebSphere software platform, WebSphere Application Server provides a rich, e-business application deployment environment with a complete set of application services including capabilities for transaction management, security, clustering, performance, availability, connectivity and scalability. Version 5 offers full J2EE specification support (Servlet 2.3, JSP 1.2, EJB 2.0, and others), as well as a variety of extensions.

After installing the Linux for zSeries system, we needed to configure the system to test our connectors. First we made sure we had enough storage to install WebSphere Application Server, then we installed the full WebSphere Application Server product and its prerequisites, as follows.

4.1.1 Space requirements for installation

Table 4-1 shows the space requirements for the WebSphere Application Server.

Table 4-1 *WebSphere Application Server space requirements*

	Base product	Network deployed product	IBM HTTP Server	Tivoli® Global Security Kit	Temp space
Install Dir	/opt/WebSphere/AppServer	/opt/WebSphere/DeploymentManager	/opt/WebSphere/DeploymentManager	/opt/ibm/gsk5	/tmp
Min free space	512 MB	512 MB	20.4 MB	13.2 MB	

4.1.2 WebSphere MQ client for Linux on zSeries

When installing the IBM WebSphere Application Server, you can install the embedded messaging feature (selected by default) for use as the Java Message Service (JMS) provider. However, in order to install the embedded messaging feature successfully, you need to complete several other actions first. These actions include considering whether you also want to use WebSphere MQ on the same host, and defining the groups and users needed for embedded messaging. Because one of our scenarios is to test the MQ connector, we needed to install WebSphere MQ locally and define the groups and users needed for embedded messaging. So we will describe the installation steps for WebSphere MQ.

4.2 Installing WebSphere MQ Client for Linux on zSeries

Following are the steps required to install WebSphere MQ Client. We also chose to install the WebSphere MQ Server so that we could test the local puts and gets to and from the local queue.

1. Log in as root. We set up a secure environment to log in, so we telnetted in to a user called user1, then issued the `su - root` command to switch to root, or we `ssh`d to root. This allowed us to pick up the effective groups that root was defined to. This was important because when we telnetted directly to root, although the `mqm` and `mqbrkrs` groups were defined, they were not the root's effective group list.
2. Go to the directory containing the WebSphere MQ for Linux for zSeries software:

```
cd /mnt/sw/mq53
```

3. Before you can install the MQSeries components you need to read and accept the license agreement.

Run the `mqlicense.sh` script and accept the license agreement:

```
./mqlicense.sh
```

Enter the number that corresponds to the language you prefer (1 for English).

Press Enter to see the agreement.

Press 1 to accept.

If you are using an x-11 window, then instead of performing the previous steps, you will see a panel to which you can click **Accept**, as shown in Figure 4-1.

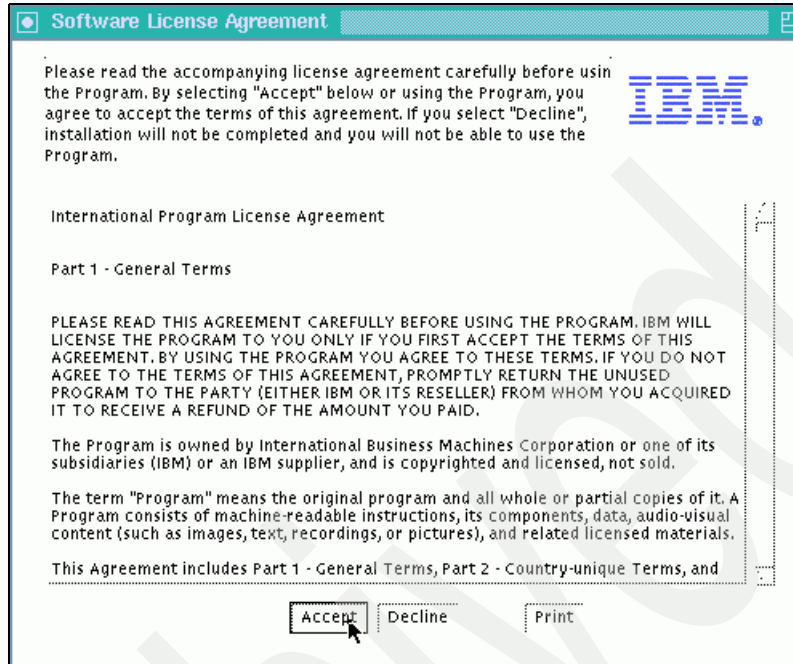


Figure 4-1 MQ license agreement

4. Use the `rpm` command to install each MQ component. The runtime component must be installed first, but the other packages can be installed in any order.

```
rpm -Uvh MQSeriesRuntime-5.3.0-4.s390.rpm
rpm -Uvh MQSeriesSDK-5.3.0-4.s390.rpm
rpm -Uvh MQSeriesServer-5.3.0-4.s390.rpm
rpm -Uvh MQSeriesClient-5.3.0-4.s390.rpm
rpm -Uvh MQSeriesSamples-5.3.0-4.s390.rpm
```

5. Run the `setmqcap` command, inputting the number of processors that you have paid for.

```
setmqcap 2
```

4.2.1 Verifying the MQ Client installation

We verify the MQ Client installation by testing if we can create a default queue, start the Queue Manager, and define a local queue as follows:

1. Change the effective user ID to `mqm`, making the shell a login shell, then create a default Queue Manager called `venus.queue.manager` from `mqm`:

```
su -l mqm
```



```
crtmqm -q venus.queue.manager
```

2. Start the Queue Manager that is going to run the commands and wait for the started message:

```
strmqm
```

3. Start using the MQ Series commands (MQSC) facility interactively by using the **runmqsc** command. A Queue Manager name has not been specified, therefore the MQSC commands will be processed by the default Queue Manager we created previously:

```
runmqsc
```

4. Define a local queue called ORANGE.QUEUE:

```
define qlocal (orange.queue)
```

5. End the interactive input of MQSC:

```
end
```

Our output is shown in Example 4-1.

Example 4-1 Verify the MQ client installation

```
linux11:/mnt/sw/mq # su -l mqm
mqm@linux11:~> crtmqm -q venus.queue.manager
WebSphere MQ queue manager created.
Creating or replacing default objects for venus.queue.manager.
Default objects statistics : 31 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
mqm@linux11:~> strmqm
WebSphere MQ queue manager 'venus.queue.manager' started.
mqm@linux11:~> runmqsc
5724-B41 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
Starting MQSC for queue manager .
define qlocal (orange.queue)
  1 : define qlocal (orange.queue)
AMQ8006: WebSphere MQ queue created.
endd
  2 : end
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

4.2.2 Testing the MQ installation

To test the Queue Manager and queue, use the amqsput sample program to put a message on the queue, and the amqsget sample program to get the message back from the queue as follows:

1. Change into the /opt/mqm/samp/bin directory, which contains the sample programs:

```
cd /opt/mqm/samp/bin
```

2. Put a message on the queue:

```
./amqsput ORANGE.QUEUE
```

Type some message text, on one or more lines, followed by a blank line.

3. Get the message from the queue:

```
./amqsget ORANGE.QUEUE
```

Press Enter to end.

Example 4-2 shows this procedure.

Example 4-2 Testing the MQ installation commands

```
mqm@linux11:~> cd /opt/mqm/samp/bin
mqm@linux11:/opt/mqm/samp/bin> ./amqsput ORANGE.QUEUE
Sample AMQSPUTO start
target queue is ORANGE.QUEUE
Say it aint so
beauty

Sample AMQSPUTO end
mqm@linux11:/opt/mqm/samp/bin> ./amqsget ORANGE.QUEUE
Sample AMQSGETO start
message <Say it aint so>
message <beauty>
end
no more messages
Sample AMQSGETO end
MQJE001: Completion Code 2, Reason 2085
```

4.3 Installing WebSphere Application Server

There are two ways to install the WebSphere Application Server: By using the GUI mode or by using the silent mode. We chose to use the GUI mode for our installation.

Interactive - GUI mode

This method requires an X-server on your workstation. We chose to install using this method. For our installation we used Hummingbird's Exceed product.

Automated - silent mode

This process uses a default response file. A default response file, named `install.script`, is supplied with the WebSphere Application Server.

4.3.1 Installing with the installation wizard GUI

To do this:

1. Log in as root and set up our graphical environment.
2. Launch the installation wizard.

Note: The `launchpad.sh` script does not work on the zSeries product. You have to execute the install script instead.

Change to the installation directory `cd /mnt/sw/was502/linuxs390`.

Execute the install script `./install.sh&`.

3. Accept the default language (English). Click **Next**.
4. The installation wizard opens and a welcome page appears. Click **Next**, as shown on Figure 4-2.

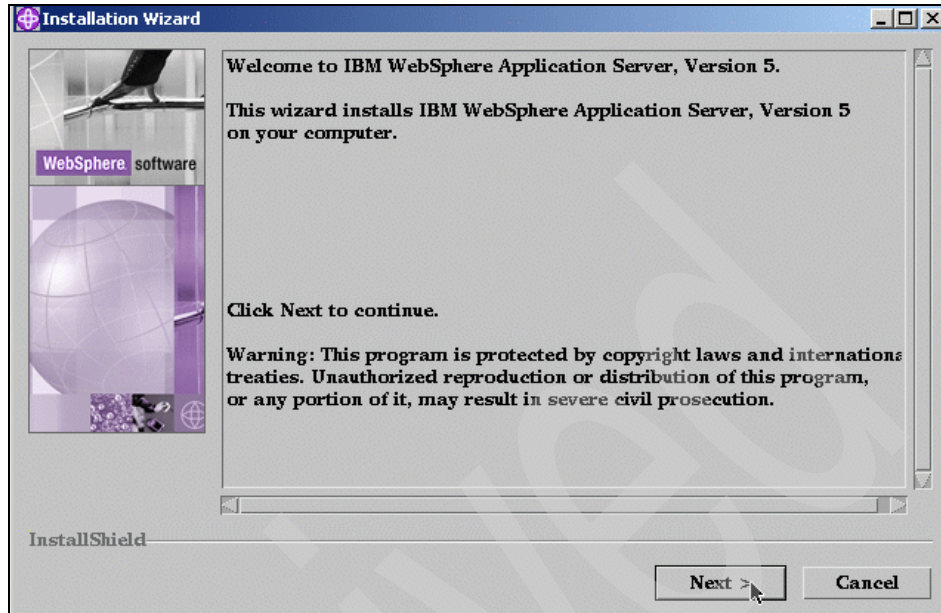


Figure 4-2 WebSphere Application Server welcome page

5. The license agreement appears for you to read. Accept the agreement and click **Next**.
6. The installation Wizard checks for system prerequisites at this time. If all requirements have been met, another panel will appear asking you to choose the setup type that best suits your needs, Custom or Full. We chose the Full install. Click **Next**.
7. A panel then appears with the installation directories for IBM WebSphere Application Server Version 5 and IBM HTTP Server Version 1.3.26, as shown in Figure 4-3. You may customize these directories or accept the defaults. We accepted the defaults. Click **Next**.

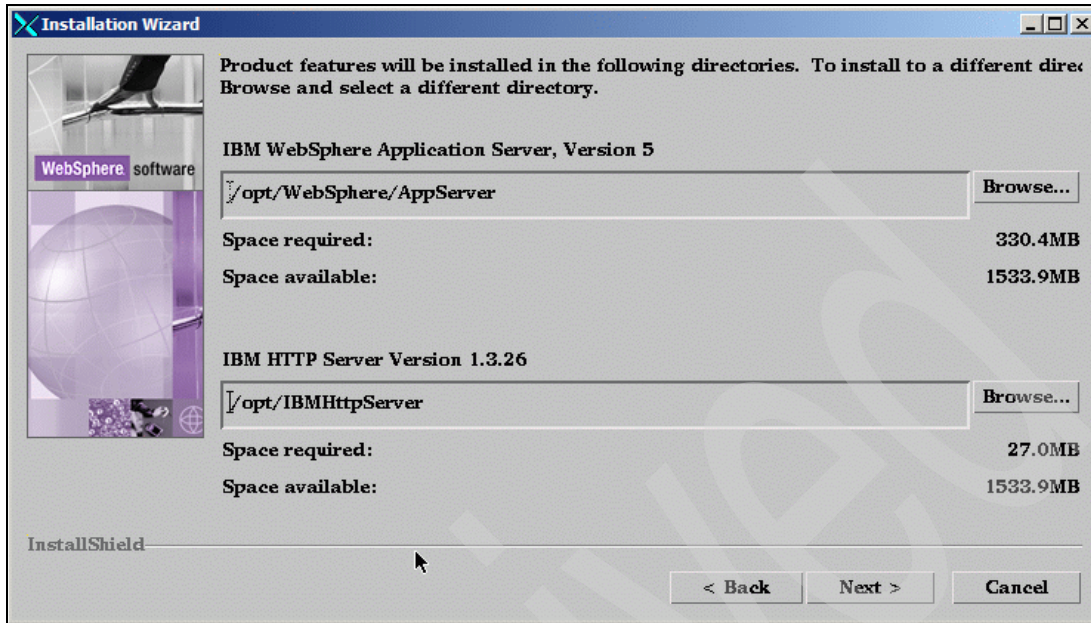


Figure 4-3 WebSphere Application Server and HTTP server directories

8. The next panel shows the node name and host name or IP address. You may override this information or accept the defaults as shown in Figure 4-4 on page 76. Accept the defaults. Click **Next**.

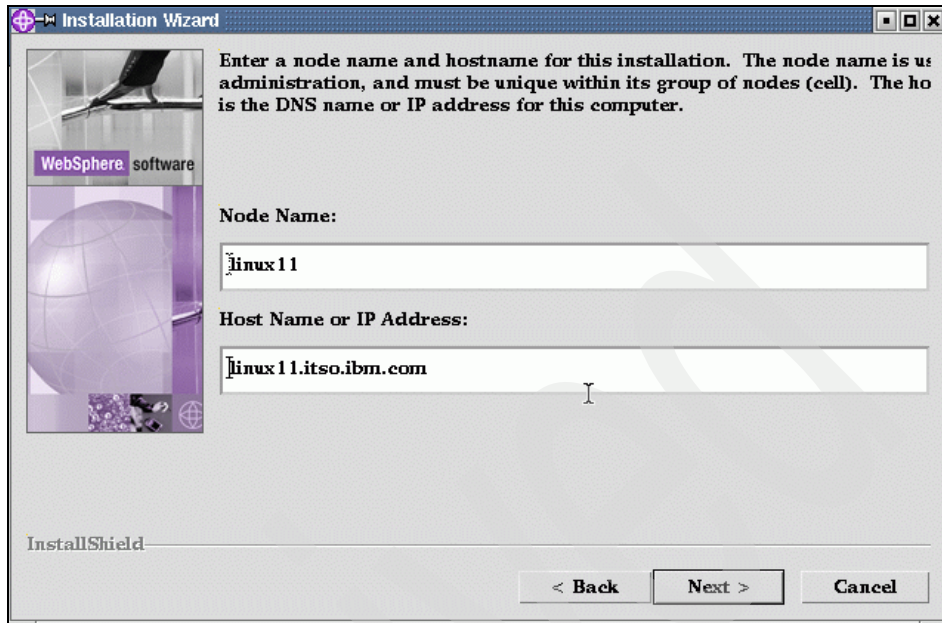


Figure 4-4 Node name and host name

9. The next panel shows you a summary of the features that you have chosen to be installed, as shown in Figure 4-5. Click **Next** to install the product(s) if you have no changes.

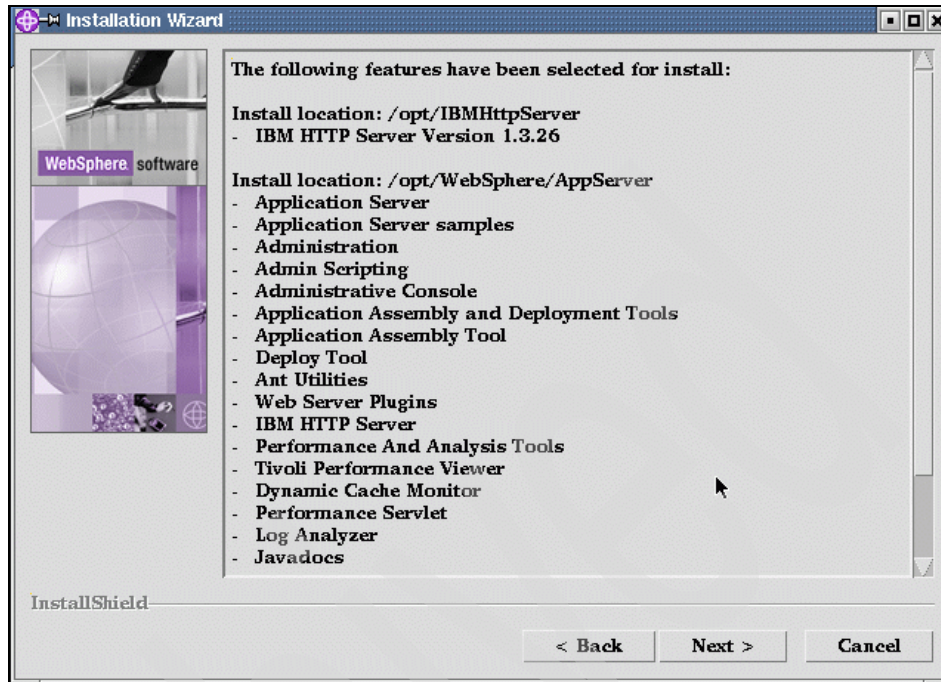


Figure 4-5 Summary of products selected for installation

10. When the installation is complete, the wizard displays the registration form. You can register now or deselect the registration form to register later. We deselected the button. Click **Next**.
11. Click **Finish** to close the installation wizard.
12. After the installation completes, the First Steps application should be launched for you, as shown in Figure 4-6.

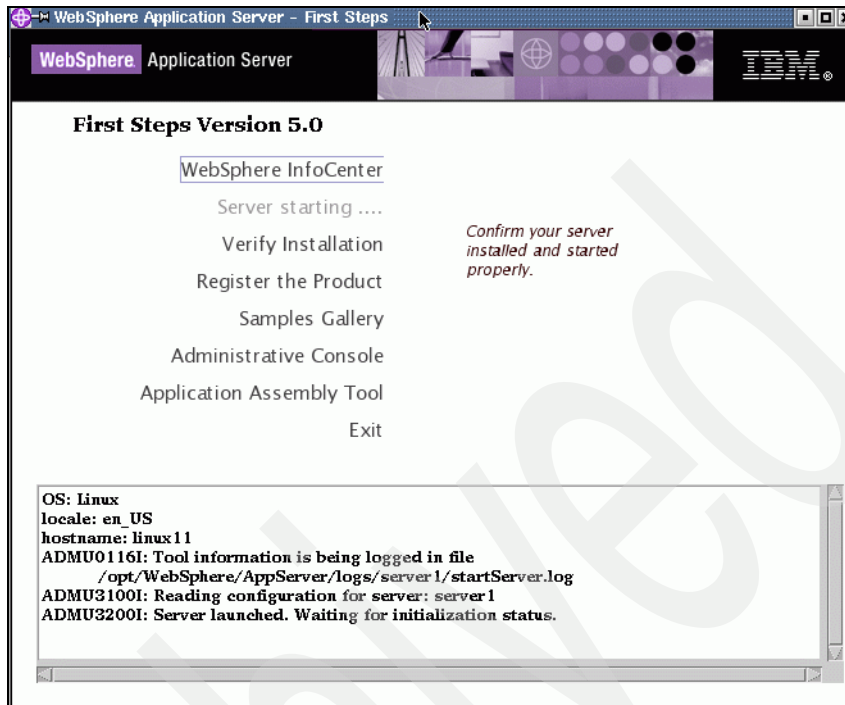


Figure 4-6 WebSphere Application Server - First Steps

To restart the First Steps interface, you can execute the first steps script:

```
/opt/WebSphere/AppServer/bin/firststeps.sh&
```

We are going to perform the steps manually so that we know how to execute them again later. Exit out of first steps.

4.3.2 Getting started

To get started:

1. Start the application. Back on the Linux shell execute the startServer script to start the application server. Server1 is the name of the configuration directory of the server you want to start.

```
/opt/WebSphere/AppServer/bin/startServer.sh server1
```

You will see the following messages when the server is started:

```
Server server1 open for e-business; process id is 18000
```


2. Explore the sample applications you just installed. Samples are installed by default. You can point your browser directly to `http://9.12.9.12:9080/WSamples`, as shown in Figure 4-7.

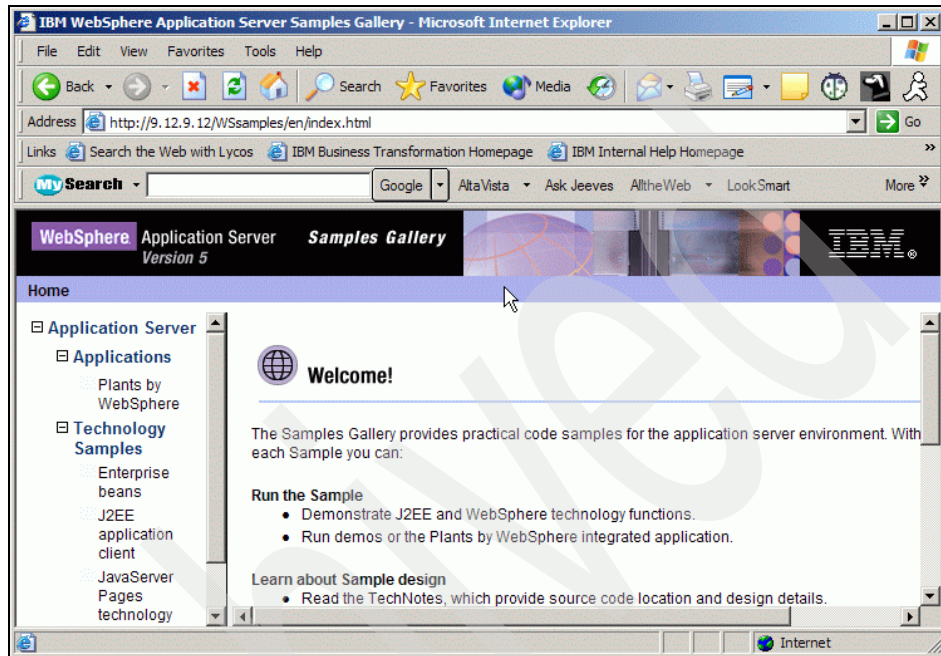


Figure 4-7 Samples

The Samples Gallery includes the following samples:

- The Plants by WebSphere application, which demonstrates several J2EE functions, using an online store that specialized in plant and garden tool sales.
 - Technology samples, which showcase enterprise beans, servlets, JavaServer Pages technology, message-driven beans, and J2EE application client.
 - The Java Pet Store Application, which demonstrates J2EE technology using an online pet store.
 - The message-driven beans samples demonstrate message-driven beans receiving messages from the Point-to-Point and Publish Subscribe messaging models. It also demonstrates Java Message Service (JMS) inside the client container.
3. Launch the Administrative Console.

The Administrative Console is a confirmation editor that runs in a Web browser. It lets you work with application server configuration files encoded in XML. Changes made using the Administrative Console take effect the next time you start the application server. Point your browser to:

`http://linux11.itso.ibm.com:9090/admin`

A login window pops up. Enter a user and click **OK**. We used the user `ITSUser`. You will now see the WebSphere Application Server's Administration Console, as shown in figure Figure 4-8 on page 80.

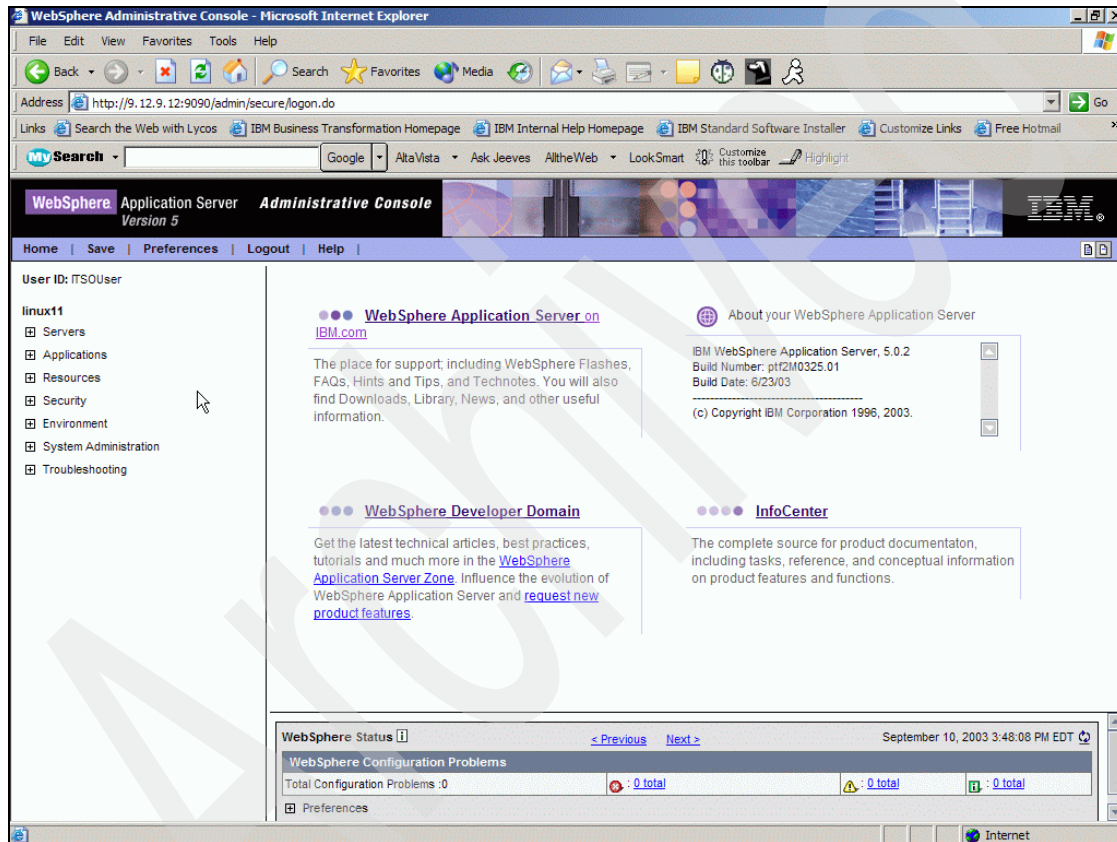


Figure 4-8 WebSphere Application Server's Administration Console

4. Start the IBM HTTP server:
`/opt/IBMHttpServer/bin/apachectl start`
5. Test the HTTP server as shown in Figure 4-9. Point your browser to `http://linux11.itso.ibm.com`. You will see the welcome page from the HTTP server.



Figure 4-9 HTTP home page

6. Test the HTTP server plug-in for the WebSphere Application Server. Point your browser to <http://linux11.itso.ibm.com/WSamples>. You will once again see the samples gallery page (Figure 4-10). The difference is that the request is now coming from the HTTP server instead of directly from the browser.

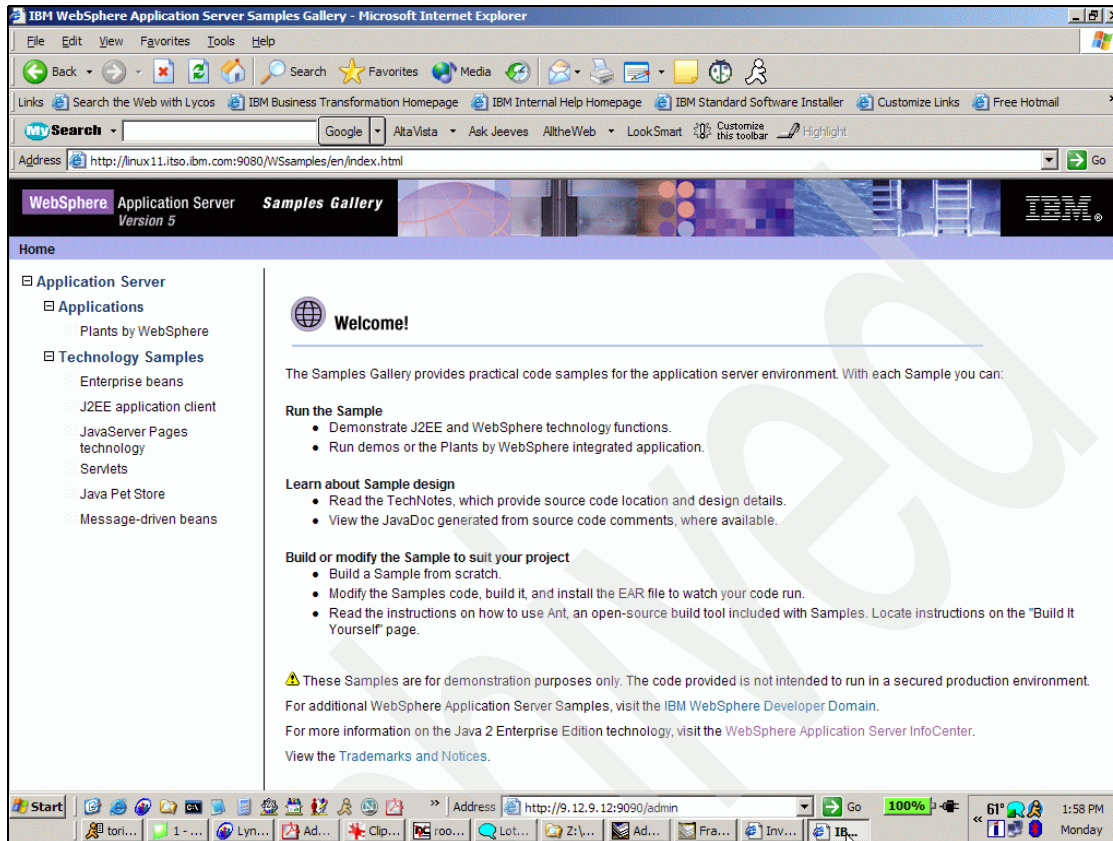


Figure 4-10 The samples gallery page

CICS J2EE connectors

This chapter describes how we configured WebSphere Application Server to work with the CICS Transaction Gateway on Linux for zSeries to communicate with CICS regions on VSE and z/OS.

We describe the following:

- ▶ Overview of the test environment
- ▶ Setting up the CTG on Linux for zSeries
- ▶ Setting up the CICS regions on VSE and z/OS
- ▶ Testing the connectivity from Linux for zSeries
- ▶ Configuring WebSphere for CICS connections
- ▶ Problem determination

5.1 Overview of the test environment

We had two test scenarios for our CICS connections, one for z/OS and one for VSE (Figure 5-1). The first thing we need to do is to install the CICS Transaction Gateway onto our Linux system. After the code is installed, we configured our CTG connections and tested them with a sample Java application. Finally, we configured WebSphere Application Server to allow us to communicate with CICS TS and CICS/VSE®, as shown in Figure 5-2 and Figure 5-3, from the WebSphere Application Server. For our testing purposes, we used the same CICS J2EE Trader application for both of these scenarios.

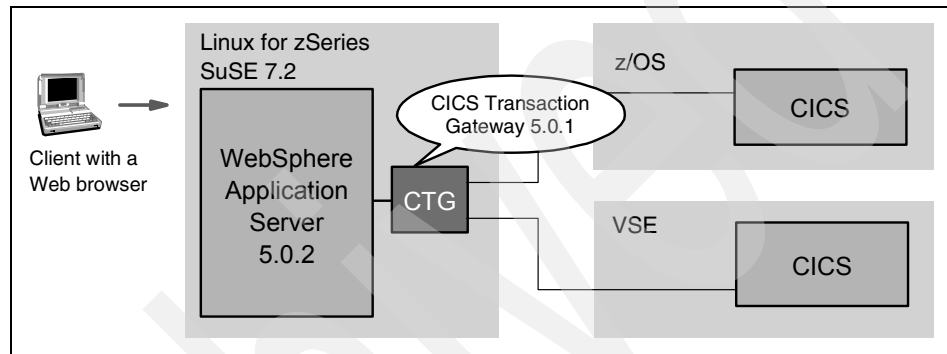


Figure 5-1 Overview of CICS connections to z/OS and VSE

To validate the two scenarios there were minimal unique definitions that we had to make. In WebSphere, we defined one resource adapter, but created two different J2C connection factories, one pointing to the z/OS environment and one pointing to the VSE environment. In the CTG, we defined two server definitions, one for z/OS and one for VSE. We will describe both scenarios.

5.1.1 CICS connection to z/OS

Figure 5-2 shows the environment for our CICS connection from Linux to z/OS. We have configured the WebSphere Application Server for Linux for zSeries to use the CICS resource adapter. We defined a J2C connection factory associated with the CICS resource adapter that the application references. The J2C connection factory holds a list of configuration properties, including the JNDI name and customer properties, such as the the server name defined to the CTG and the port to which the CTG is listening on. It is through the CICS J2EE connector definition that we can access the program running on the CICS TS V2.2 region on z/OS.

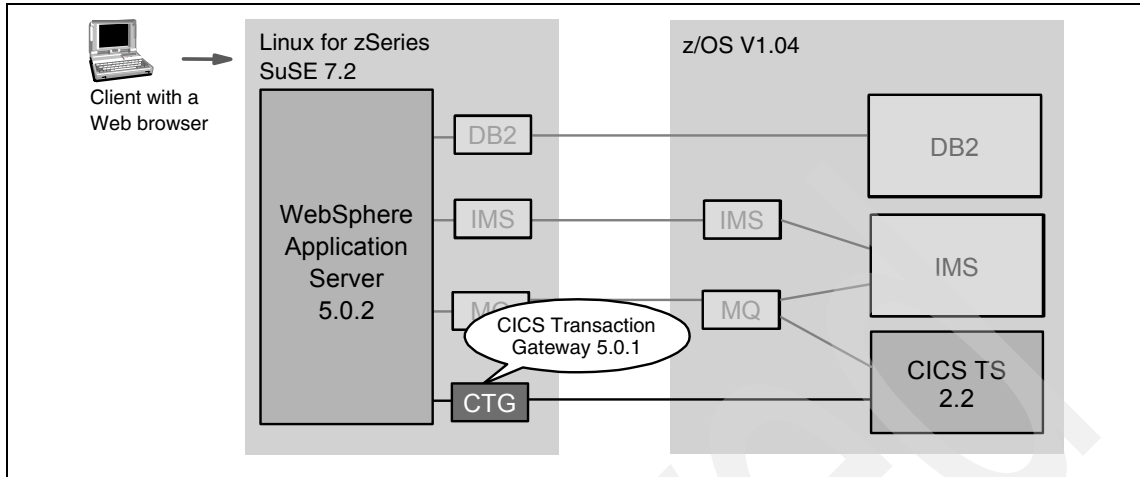


Figure 5-2 CICS connection to z/OS

5.1.2 CICS connection to VSE

Figure 5-3 shows the environment for our CICS connection from Linux to VSE. We have configured the WebSphere Application Server for Linux for zSeries to use the CICS resource adapter. We defined a J2C connection factory associated with the CICS resource adapter that the application references. The J2C connection factory holds a list of configuration properties including the JNDI name and customer properties, such as the server name defined to the CTG and the port to which the CTG is listening on. It is through the CICS J2EE connector definition that we can access the program running on the CICS VSE region.

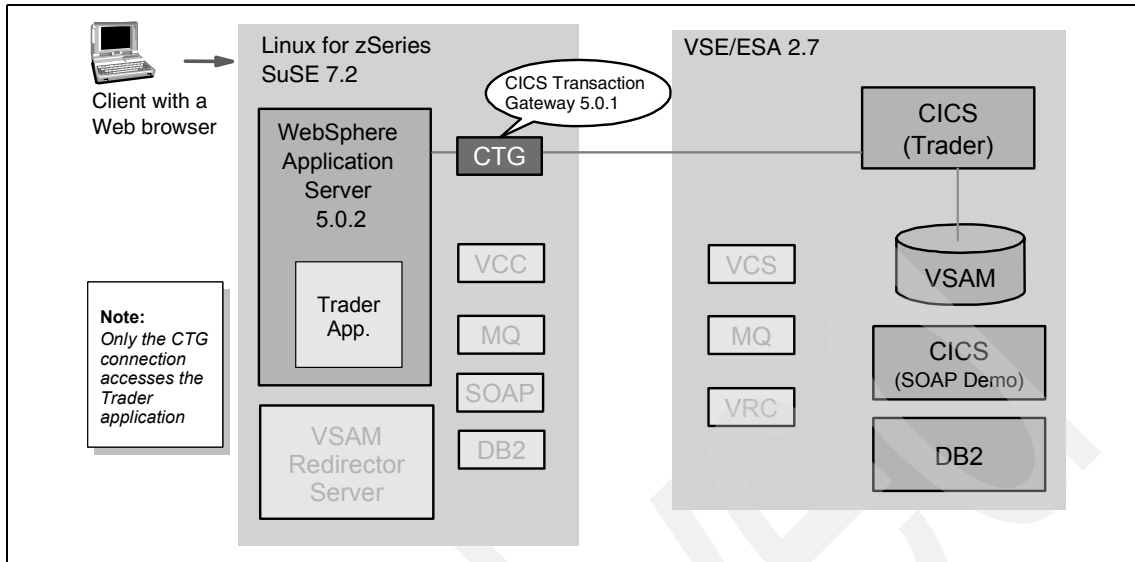


Figure 5-3 CICS connection to VSE

5.2 Setting up the CTG on Linux for zSeries

The CICS Transaction Gateway (CICS TG) is a set of client and server software components that allow a Java application to invoke services in a CICS region. The Java application can be an applet, a servlet, an enterprise bean, or any other Java application. The CTG is supported for use with CICS/ESA® V4.1, CICS/VSE 2.3 and CICS TS for VSE/VSE/ESA V1, but only if the CTG runs on a distributed platform. For use with CICS TS V1 for OS/390 or CICS TS V2 for z/OS, the CTG can run on z/OS, OS/390, or a distributed platform. There are many ways to set up the CICS TG for communication, which are thoroughly described in *CICS Transaction Gateway V5.0: Unix Administration*, SC34-6190, and in the redbook *CICS Transaction Gateway V5 The WebSphere Connector for CICS*, SG24-6133. We are going to describe one of the configurations here that we used to test our CICS Trader application.

5.2.1 Installing the CTG on Linux for zSeries

The first thing we need to do is to install the CICS Transaction Gateway onto our Linux system. After the code is installed, we can configure our connections and test them with a sample Java application. Finally, we can test the J2EE resource adapter using the CICS Trader application. Following are the steps we needed to take to install the CTG:

1. Log in as root.

2. The CICS Transaction Gateway is distributed as a compressed tar file containing an executable script file. When the file is uncompressed and untarred, there will be a customization guide (in PDF format), an rpm file, a list of changes for this edition (in various formats), and two zip files. Uncompress and untar the files:

```
gunzip c51t3m1CTG502.tar.Z
tar xvf c51t3m1CTG502.tar
```

3. Install the CICS Transaction Gateway package with the `rpm` command. This will create a directory, `/opt/ctg`. In that directory there is a command, `ctginstall`, which will be used to install the CICS Transaction Gateway.

```
rpm -ivh ctg-5.0.1-1.s390.rpm
```

4. Start the installation of the CICS Transaction Gateway by entering:

```
ctginstall
```

5. Read and accept the license agreement (Figure 5-4) to continue the installation. If the agreement does not display properly, restart the installation using the command `ctginstall 40`. This sets your screen width to 40 characters during installation.

To view the license agreement after installation, issue the command `ctgbrowse`. If the agreement does not display properly, issue the command `ctgbrowse 40`.

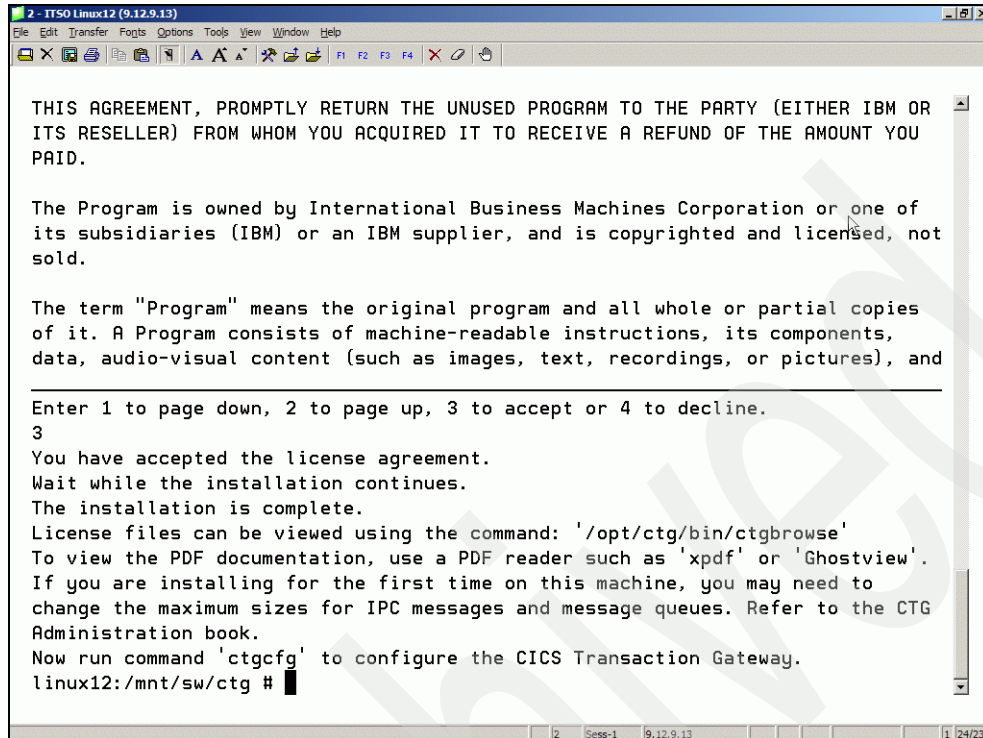


Figure 5-4 CTG license agreement

5.2.2 Configuring the CTG on Linux for zSeries

We needed to define two server definitions in the CTG that we would use to connect to CICS TS and CICS/VSE. We configure the CICS Transaction Gateway to define the CICS server running on the z/OS, followed by configuring the CTG to define the CICS server running on VSE. Here are the steps that we took:

1. Enter the **ctgcfg** command. A panel will appear asking if you wish to use the Task Guide for configuration, as shown in Figure 5-5. Click **Yes**. A Configuration Taskguide information panel will appear. Click **Next**.

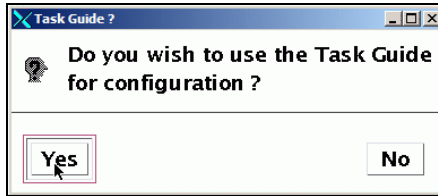


Figure 5-5 CICS Transaction Gateway Task Guide

2. The Task Guide will ask if you wish to create a CICS server definition for the clients to connect to. The default is Yes. Click **Next**.

We defined the CICS TS definitions first.

3. You need to name your CICS server definition. Although this name is arbitrary, this will be the server name that is used when testing the Java application and that will be referenced by the WebSphere Application Server. We chose to use the same name as the name of our CICS TS region, SCSCERW3, as shown in Figure 5-6 on page 89. Click **Next**.

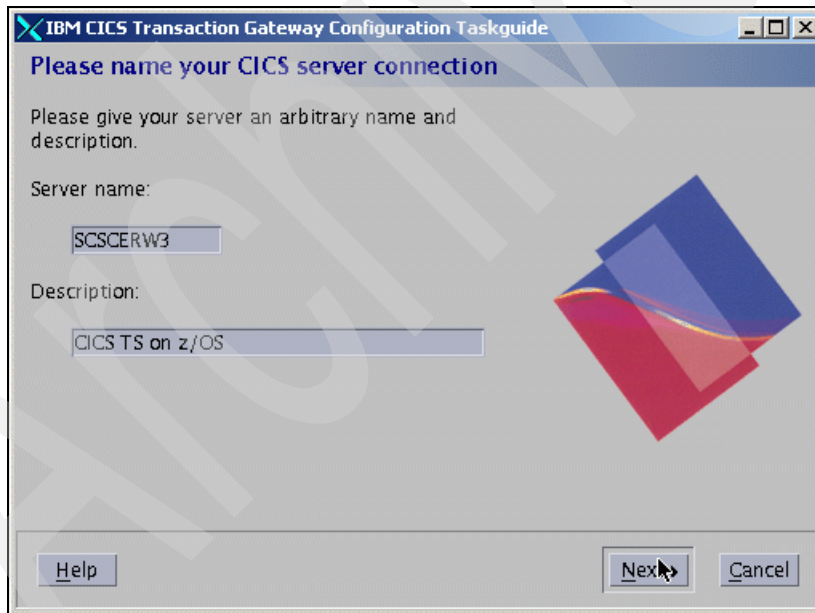


Figure 5-6 CICS Server connection name for z/OS

4. Select the protocol to use to connect to the CICS server of either TCP/IP or TCP62, as shown in Figure 5-7, and click **Next**. We chose TCP/IP, as we are connecting the directory to a CICS TS server that has the TCP/IP capability. A TCP/IP configuration is straightforward to configure, flowing over an IP

network, and does not incur the protocol conversion overhead of TCP62, eliminating the need for Anynet on the z/OS side.

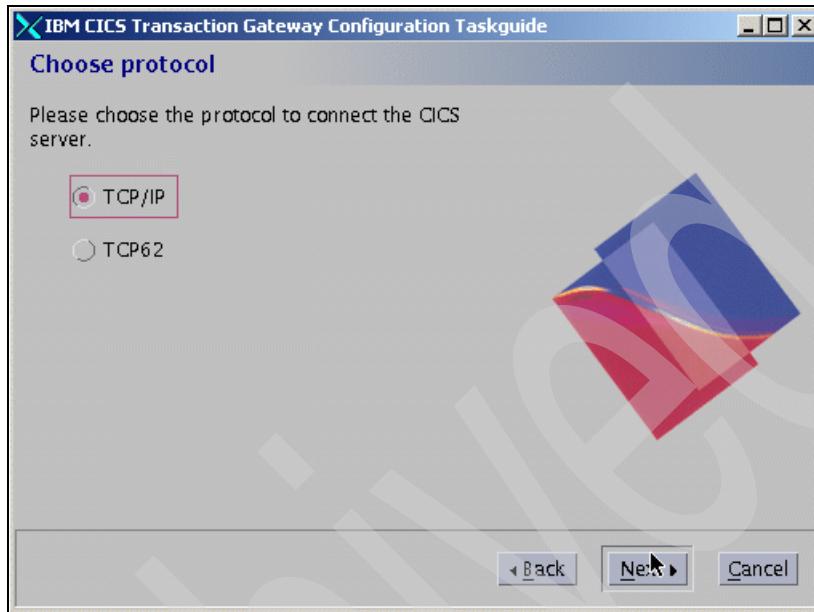


Figure 5-7 CTG protocol selection

5. Enter the host name or IP address and the port number that CICS TS is listening on, as shown in Figure 5-8, and click **Next**. In our case, the first server that we defined was for the CICS TS V2.2 region on z/OS. Remember, our goal in this redbook is not to test all of the potential connection possibilities in the CTG, but to test the J2EE resource adapters from WebSphere Application Server.

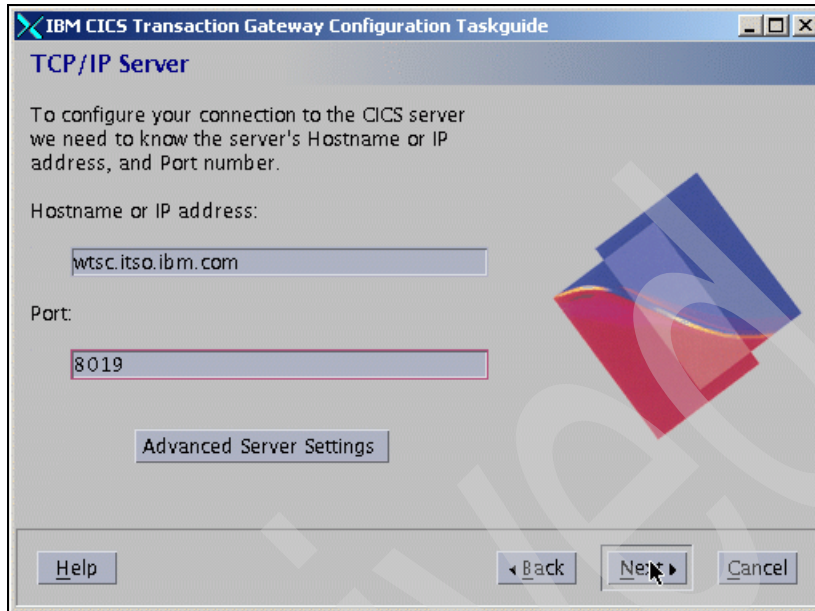


Figure 5-8 CICS host and port number selection for z/OS

6. The taskguide then asks if you wish to define another server. If you have no other servers to define, you may select **No**. We selected **Yes** since we also wanted to define the CICS VSE server definition at this time. Click **Next**. If you selected No, you may proceed to step 12 on page 93. Since we had another server to define, we were presented with the CICS Server Connection panel again.

Now we need to define our CICS/VSE server definition. We will be presented with the same panels as we were for the previous z/OS definitions. This time, however, we will enter the information for our VSE environment.

7. We needed to enter the name of the CICS/VSE server definition. Although this name is arbitrary, this will be the server name that is used when testing the Java application and that will be referenced by the WebSphere Application Server. We chose to use the same name as the name of our CICS/VSE region, VSE270, as shown in Figure 5-9. Click **Next**.

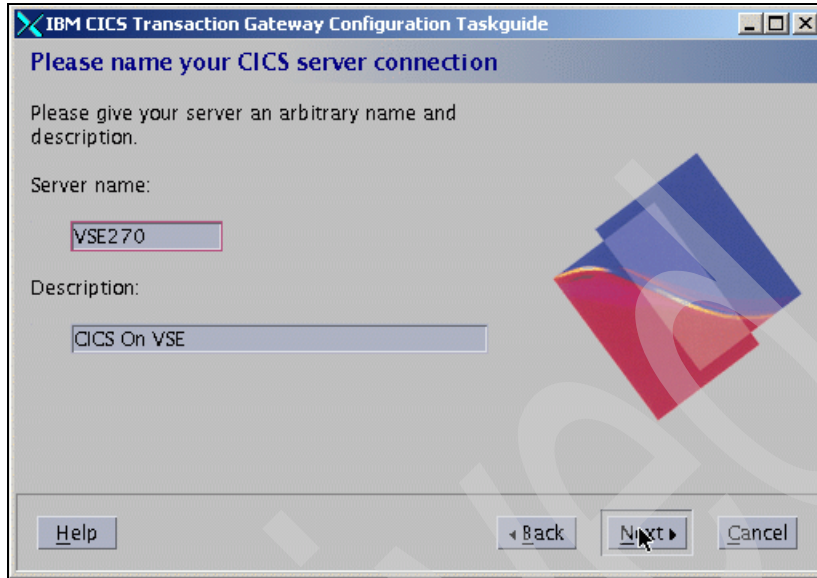


Figure 5-9 CICS Server connection for VSE

8. You are asked to select the protocol to use to connect to the CICS/VSE server of either TCP/IP or TCP62. Chose TCP/IP and click **Next**.
9. Enter the host name or IP address and the port number that CICS/VSE is listening on, as shown in Figure 5-10, and click **Next**.

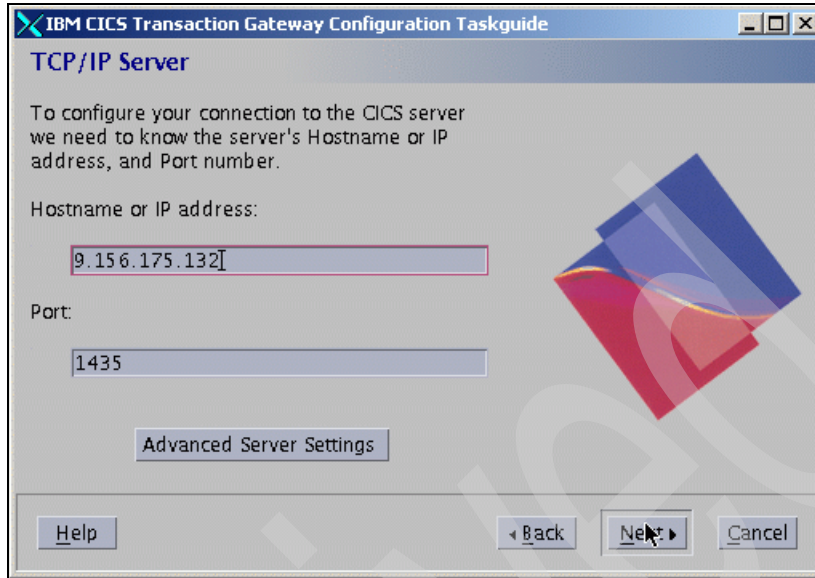


Figure 5-10 CICS host and port number selection for VSE

10. The Task Guide then asks if you wish to define another server. Since we did not have any more servers to define, we selected **No** and clicked **Next**.
11. You are asked which protocols you would like to use with the Java Gateway. We chose **TCP** and clicked **Next**.
12. You will see a panel that says Finished. Click **Finish**.

Congratulations. You have successfully defined two CICS server definitions. At this point, the IBM CICS Transaction Gateway Configuration Tool shows all of the Server definitions that the client can connect to and the protocols that the CTG can use, as shown in Figure 5-11 on page 94.

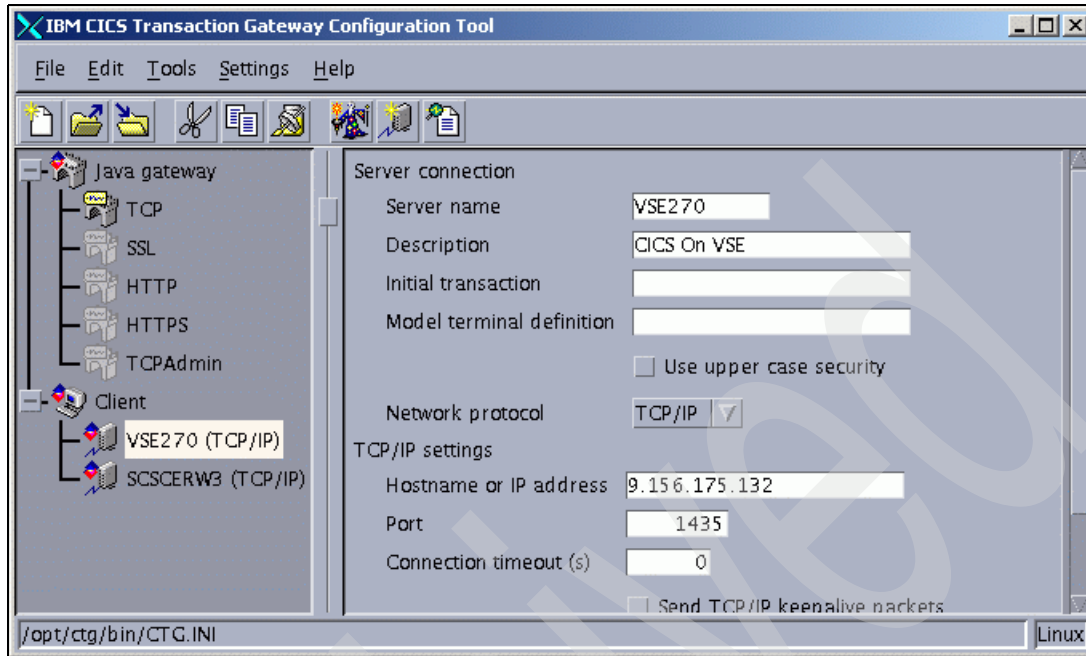


Figure 5-11 CTG definitions on the Configuration Tool

You should save your configuration file. Click **File**, then **Save**. The default is in CTG.INI, which resides in the /opt/ctg/bin directory (Example 5-1).

Example 5-1 Example CTG.INI file

```
SECTION GATEWAY
  closetimeout=10000
  ecigenericreplies=off
  uowvalidation=on
  msgqualvalidation=on
  connectionlogging=off
  initconnect=1
  initworker=1
  maxconnect=100
  maxworker=100
  noinput=off

  nonames=on
  notime=off
  workertimeout=10000
ENDSECTION

SECTION CLIENT = *
```



```
CPIPADDRESSMASK=00000000
LOGFILE=CICSCLI.LOG
TCP62PORT=0
MAXBUFFERSIZE=32
MAXREQUESTS=256
MAXSERVERS=10
MAXWRAPSIZE=0
REMOTENODEINACTIVITYPOLLINTERVAL=60
SRVRETRYINTERVAL=60
TERMINALEXIT=EXIT
TRACEFILE=CICSCLI.BIN
ENDSECTION
```

```
SECTION SERVER = SCSCERW3
DESCRIPTION=CICS TS on z/OS
UPPERCASESECURITY=N
PROTOCOL=TCPIP
NETNAME=wtsc52.itso.ibm.com
PORT=8019
CONNECTTIMEOUT=30
TCPKEEPALIVE=N
ENDSECTION
```

```
SECTION SERVER = VSE270
DESCRIPTION=CICS on VSE
UPPERCASESECURITY=N
PROTOCOL=TCPIP
NETNAME=9.156.175.131
PORT=1435
CONNECTTIMEOUT=30
TCPKEEPALIVE=N
ENDSECTION
```

```
SECTION DRIVER = TCPIP
DRIVERNAME=CCLIBMIP
ENDSECTION
```

You may use the default or save it to another file. We chose to use the default name, as shown in Figure 5-12.

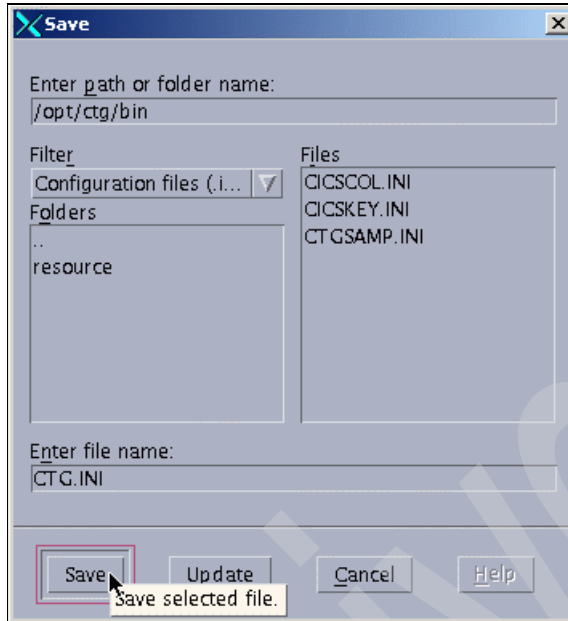


Figure 5-12 Save the CTG configuration information

You may now exit the configuration tool.

5.3 Setting up the CICS regions on z/OS and VSE

ECl over TCP/IP is a communications mechanism that provides the ability to connect from the CTG on Linux for zSeries to a CICS region over an IP network. It utilizes the support for ECl over TCPIP in CICS TS for z/OS V2.2 and CICS TS for VSE/ESA V1.1.

We will discuss the necessary TCP/IP definitions in CICS TS for zOS and CICS TS for VSE for our CTG on Linux for zSeries connection.

5.3.1 Setting up the CICS TS for z/OS environment

Before the CTG on Linux could connect to the CICS TS application on z/OS we needed to make sure that our CICS TS environment was configured.

TCP/IP definitions for CICS TS for z/OS

On z/OS, we have a CICS TS 2.2 region called SCSCERW3. To configure the CICS TCP/IP listener to handle ECI requests, we first had to confirm that the following definitions were installed into SCSCERW3:

- ▶ SIT parameter: TCPIP=YES
- ▶ CICS-supplied Transaction CIEP in group DFHIPECI installed
- ▶ CICS-supplied Program DFHIEP in group DFHIPECI installed

We then had to create a TCP/IP listener to CICS. We did this by defining a TCPIP SERVICE called ERW3TCP in the group ERW3GRP with the following definitions with the CEDA DEF TCPIPS(ERW3) GROUP(ERW3GRP) command:

- ▶ The port number on which the TCP/IP service is listening (8019)
- ▶ The protocol of the service (ECI)
- ▶ The transaction that handles incoming ECI requests (CIEP)
- ▶ The level of Attach-time security required for TCP/IP connections to CICS Clients (local or verify). We used local, as we were in a test environment.

These definitions are shown in Figure 5-13.

```
OVERTYPE TO MODIFY OR PRESS ENTER TO EXECUTE          CICS RELEASE = 0620
CEDA DEFINE TCPIPService( ERW3TCP )
TCPIPService ==> ERW3TCP
GRoup        ==> ERW3GRP
DEscription  ==>
URm          ==>
PORtnumber   ==> 08019          1-65535
STatus       ==> Open         Open | Closed
PRotocol     ==> Eci          Iiop | Http | Eci
TRansaction  ==> CIEP
Backlog      ==> 00001        0-32767
TSqprefix    ==>
Ipaddress    ==> 9.12.6.1
SOcketclose  ==> No          No | 0-240000 (HHMMSS)
SECURITY
SSL          ==> No          Yes | No | Clientauth
Certificate  ==>
AUthenticate ==>            No | Basic | Certificate | AUTORegister
                               | AUTOMatic
+
                                           SYSID=ERW3 APPLID=SCSCERW3
PF 1 HELP 2 COM 3 END          6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

Figure 5-13 TCP/IP service definition

We installed the TCPIPService with the following command:

```
CEDA INS TCPIPService(ERW3TCP) GROUP(ERW3GRP)
```

We then used the CICS CEMT I TCPIPS command to display the active service.

Application definitions for CICS TS for z/OS

We defined two COBOL programs in SCSCERW3. One was called EC01, which was used for our simple ECI connection test from our Linux system. The other was called TRADERBL, which was used for the J2EE CICS Trader application running on WebSphere Application Server for Linux for zSeries.

z/OS verification commands

After we configured the definitions in CICS, we wanted to verify that they were being picked up by the z/OS system. We issued the `netstat -a` command to verify that TCP/IP is listening on that PORT and looked for the CICS definition, as shown in Figure 5-14.

```
SCSCERW3 00072290 9.12.6.1..8019          9.12.9.12..33063      Establish
```

Figure 5-14 netstat command

5.3.2 Setting up the CICS TS for VSE environment

On VSE CICS TS runs in a partition. There is a COBOL program called TRADERBL defined to this CICS. This program will be called by the Trader application running on the WebSphere Application Server for Linux on zSeries. The connection to CICS TS is established via the CTG and TCP/IP on the Linux side addressing the port CICS TS is listening on.

Before the CTG on Linux can connect to the CICS TS application on VSE/ESA we needed to make sure that our CICS TS environment supports TCP connections.

TCP/IP definitions for CICS TS for VSE

To configure the CICS TCP/IP listener to handle ECI requests, we first have to make sure the following definitions are active for the CICS TS:

- ▶ SIT parameter: TCPIP=YES
- ▶ CICS-supplied Transaction CIEP in group DFHIPECI is installed
- ▶ CICS-supplied Program DFHIEP in group DFHIPECI is installed

We then have to create a TCP/IP listener to CICS. We do this by defining a TCPIP SERVICE called ECI in group CTG. The following command brings up the following screen, which you have to complete with additional information.

```
CEDA DEF TCPIPS(ECI) GROUP(CTG)
```

Example 5-2 Define the IP listener for the CTG

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0411
CEDA ALTER TCpipservice( ECI                      )
```

```

TCpipservice   : ECI
Group          : CTG
Description    ==>
Urm           ==>
Portnumber     ==> 01435           1-65535
Certificate    ==>
SStatus       ==> Open           Open | Closed
SSI           ==> No             Yes | No | Clientauth
Attachsec     ==> Verify        Local | Verify
TRansaction   ==> CIEP
Backlog       ==> 00005         0-32767
TSqprefix     ==>
Ippaddress    ==>
S0cketclose   ==> No           No | 0-240000

```

SYSID=CIC5 APPLID=A0006CI2

PF 1 HELP 2 COM 3 END 6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL

The parameters are:

TCPipservice ECI: This is only a logical name, and can be any other 8-character string.

Group CTG: The name of the group can be new or an already existing group.

Description Put in whatever makes sense.

Urm Leave it free.

Portnumber 1435: The port that the Linux MQ must connect to can be any free port.

SStatus Open: The port will be serviced immediately after CICS starts.

SSI No we do not use Secured Socket Layer.

Attachsec Verify that incoming requests have to provide user ID and Password.

TRansaction CIEP: This transaction code will invoke the program DFHIPECI to handle the request.

Backlog 0005: The max number of unserved requests.

The rest of the parms you can leave with the default.

We install the new group containing the TCPIPService with the following command:

```

CEDA INS TCPIPService(ECI) GROUP(CTG)

```

Do not forget to add the new group to the CICS TS configuration list.

We then use the CEMT | TCPIPS command to display the active service.

Example 5-3 CEMT | TCPIPS

```
I TCPIPS
STATUS: RESULTS - OVERTYPE TO MODIFY
  TcpiPs(ECI      ) Bac( 00005 ) Con(0000) Por(01435)   Ope
    Tra(CIEP)                Ipa(9.156.175.132 )       Wai
```

In addition we verify that tcp is active with CEMT | TCP.

Example 5-4 CEMT | TCP

```
I TCP
STATUS: RESULTS - OVERTYPE TO MODIFY
  Tcp Ope
```

5.3.3 Testing the CTG to VSE CICS/TS connectivity

The CTG for Linux on zSeries includes some programming examples we can use to test our environment.

The host-based programs can be found in /opt/ctg/samples/server; see Example 5-5.

Example 5-5 Host-based program examples

```
linux11:/opt/ctg/samples/server # ls -l
total 76
drwxr-xr-x  2 root  root    4096 Oct 31 14:21 .
drwxr-xr-x  8 root  root    4096 Oct 31 14:21 ..
-rw-r--r--  1 root  root    6475 Jul 16 00:01 ec01.ccp
-rw-r--r--  1 root  root    7983 Jul 16 00:01 ec02.ccp
-rw-r--r--  1 root  root    7971 Jul 16 00:01 ep01.ccp
-rw-r--r--  1 root  root   12838 Jul 16 00:01 ep02.ccp
-rw-r--r--  1 root  root    2079 Jul 16 00:01 ep02map.bms
-rw-r--r--  1 root  root    4060 Jul 16 00:01 ep03.ccp
-rw-r--r--  1 root  root    8494 Jul 16 00:01 epinq.ccs
-rw-r--r--  1 root  root    6449 Jul 16 00:01 mapinq.bms
linux11:/opt/ctg/samples/server #
```

The Linux-based Java programs can be found in /opt/ctg/samples/java/com/ibm/ctg/samples/eci; see Example 5-6.

Example 5-6 Java-based ECI program example

```
linux11: # ls -l
total 96
drwxr-xr-x  2 root    root      4096 Nov 20 16:32 .
drwxr-xr-x  7 root    root      4096 Nov 19 14:08 ..
-rw-r--r--  1 root    root     18695 Jul 16 00:01 EciA1.java
-rw-r--r--  1 root    root      5184 Nov 19 14:06 EciB1.class
-rw-r--r--  1 root    root     12273 Jul 16 00:01 EciB1.java
-rw-r--r--  1 root    root     22810 Jul 16 00:01 EciB2.java
-rw-r--r--  1 root    root     23944 Jul 16 00:01 EciI1.java
linux11:/opt/ctg/samples/java/com/ibm/ctg/samples/eci #
```

For testing the ECI interface first upload the host program EC01.CPP to your VSE system. EC01.CCP is a small CICS COBOL online program. Called by the Java servlet program EciB1 using ECI, it returns his commarea to the ECI interface. The commarea is returned to the Java servlet and displayed on the screen.

First compile and install program EC01 as a CICS online program.

Next compile the Java servlet to do that change to subdirectory /opt/ctg/samples/java, and compile EciB1.java (see Example 5-7). The new EciB1.class file is placed in the subdirectory:

```
/opt/ctg/samples/java/com/ibm/ctg/samples/eci/
```

Example 5-7 Compile EciB1.java

```
linux11:/opt/ctg/samples/java # javac ./com/ibm/ctg/samples/eci/EciB1.java
linux11:/opt/ctg/samples/java #
```

Now we can run our test by starting the servlet. The servlet must be started *exactly* from the /opt/ctg/samples/java directory.

Example 5-8 Test ECI connection with host program EC01

```
linux11:/opt/ctg/samples/java # java com.ibm.ctg.samples.eci.EciB1
CICS Transaction Gateway Basic ECI Sample 1
Usage: java com.ibm.ctg.samples.eci.EciB1 [Gateway URL]
                                         [Gateway Port Number]
                                         [SSL Classname]
                                         [SSL Password]
```

To enable client tracing, run the sample with the following Java option:
-Dgateway.T.trace=on

The address of the Gateway has been set to local: Port:2006

CICS Servers Defined:

1. SCSCERW3 -CICS TCP/IP
2. VSE260 -
3. **VSE270** -
4. ERW3REM -Using CTG On z/OS
5. SCSCERWX -CICS TS on z/OS

Choose Server to connect to, or q to quit:

3

Program EC01 returned with data:-

Hex:

```
ffffff2ffffff561ffffff1ffffff161ffffff0ffffff340ffffff2ffffff17afffffff
2ffffff27affffff0ffffff10
```

ASCII text: ??a??a??@??z??z??

```
linux11:/opt/ctg/samples/java #
```

Now we disable the CICS program with `cemt s prog (EC01)` disabled and see what happens.

Example 5-9 Test ECI connection with host program EC01 disabled

```
linux11:/opt/ctg/samples/java # java com.ibm.ctg.samples.eci.EciB1
CICS Transaction Gateway Basic ECI Sample 1
Usage: java com.ibm.ctg.samples.eci.EciB1 [Gateway URL]
                                           [Gateway Port Number]
                                           [SSL Classname]
                                           [SSL Password]
```

To enable client tracing, run the sample with the following Java option:
`-Dgateway.T.trace=on`

The address of the Gateway has been set to local: Port:2006

CICS Servers Defined:

1. SCSCERW3 -CICS TCP/IP
2. VSE260 -
3. VSE270 -
4. ERW3REM -Using CTG On z/OS
5. SCSCERWX -CICS TS on z/OS

Choose Server to connect to, or q to quit:

3

You are not authorised to run this transaction.

ECI returned: ECI_ERR_TRANSACTION_ABEND
Abend code was AEIO

5.4 Testing the connectivity from Linux for zSeries

After you have installed and configured the CTG on Linux for zSeries and set up your CICS servers on the host systems for TCP/IP communication, the next step is to test the communication links between the CICS Transaction Gateway and your CICS servers.

5.4.1 Testing the connections

Log on to Linux as root and ping the IP address of the image that CICS is running on:

```
ping 9.12.6.1 -c2
```

You should have no packets lost, as shown in Example 5-10.

Example 5-10 Test ping to TCP/IP zOS

```
linux11:~ # ping 9.12.6.1 -c2
PING 9.12.6.1 (9.12.6.1): 56 data bytes
64 bytes from 9.12.6.1: icmp_seq=0 ttl=63 time=3.003 ms
64 bytes from 9.12.6.1: icmp_seq=1 ttl=63 time=1.175 ms
--- 9.12.6.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 1.175/2.089/3.003 ms
linux11:~ #
```

5.4.2 Creating a simple TestECI program on Linux for zSeries

We found that by testing the connection using a traditional servlet-based application that only uses the CICS TG ECI Request class to call a program in a CICS region and invoked by a Linux line command was helpful in assuring that the connection from the CTG on Linux to the CICS application on VSE or z/OS was set up correctly. Once we had verified the connection from our simple connection test, we were able to focus our attention on configuring the WebSphere connector for CICS.

For testing the connections to CICS TS on both z/OS and on VSE, we used a Java program called TestECI. This program was designed to allow you to make calls to CICS via the CTG. It allows you to specify all the relevant call parameters, and one or more programs to run as an extended LUW. It is a

variation of EciB1.java, which is supplied with the CTG. It sends a simple commarea over to a program running under CICS. The program, in turn, received the COMMAREA and returned a date and timestamp. In order to test the program we did the following:

- ▶ We put the following definitions in /etc/profile so that each time the user logs on to Linux, he will have them in his profile:

```
export JAVA_HOME=/opt/IBMJava2_s390-131
export CLASSPATH=./opt/classes:/opt/ctg/classes/ctgclient.jar:$CLASSPATH
export PATH=/opt/IBMJava2_s390-131/bin:/opt/IBMJava_s390_131/jre/bin:$PATH
```

- ▶ We made a subdirectory under the CTG sample Java directory on Linux to put our sample Java code in, as the package name used in the application is com.ibm.ctg.test:

```
cd /opt/ctg/samples/java/com/ibm/ctg
mkdir test
cd test
```

- ▶ We put the compiled Java code for the TestECI application into this directory:

```
mv TestECI.class /opt/ctg/samples/java/com/ibm/ctg/test
```

Note: You can use the EciB1.java code to validate your connections. EciB1 is a sample Java program provided with CTG V5 and shows the basic use of CTG API by querying the Client for a list of available servers and launching a transaction on the server chosen. It requires a program, EC01, to be installed on your CICS server. The CICS server must be set up to return the contents of the commarea as ASCII text. If the code page of the application is different from the code page of the server, data conversion must be performed at the server by making use of CICS-supplied resource conversion capabilities, such as the DFHCNV macro definitions. This source for the EciB1.java can be found in:

```
/opt/ctg/samples/java/com/ibm/ctg/samples/eci
```

- ▶ We executed the java code, TestECI, from the /opt/ctg/samples/java directory with the following command:

```
java com.ibm.ctg.test.TestECI jgate=local: server=SCSCERW3 commarea=Hello
commarealength=100 prog0=SAMPLE1
```

Where:

java Command used to execute the java program.

com.ibm.ctg.test.TestECI Package Name to execute.

jate Specifies the IP address of the CICS Gateway that we are using. Since the CTG is on the system we are running on, we enter local:

server	Names the server definition in the CTG.INI file that we used to describe our connection.
commarea	Specifies the contents of the COMMAREA.
commarealength	Specifies the length of the COMMAREA.
prog0	Specifies the he name of the program that is being executed.

Note: If we used the CTG on z/OS, the jgate would be the IP address of the z/OS image (that is, 9.12.9.1) without the colon (:), and the server definition would be the name of the server in the CTG.INI file on the z/OS machine. All else would be the same.

Now that we have verified that the communication between a simple java application on Linux For zSeries using our CTG definition to our CICS TS for z/OS server is working correctly, we are now ready to configure WebSphere for the CICS J2EE connector so that we can test our CICS Trader application on our Linux system.

5.5 Configuring WebSphere for CICS connections

For the WebSphere Application Server to successfully connect to our backend CICS systems, we needed to make several definitions in the WebSphere Administrative Console to define the CICS resources that the WebSphere Application Server will use when executing the CICS J2EE application.

5.5.1 Installing the resource adapter

First we needed to install the CICS ECI Resource Adapter (cicseci.rar). The CICS Resource Adapter is part of the CTG installation package and resides in /opt/ctg/deployable. Two resource adapters are provided with the installation of the CTG: External Call Interface (ECI) and External Presentation Interface (EPI). For our purposes, we only needed to install the CICS ECI Resource Adapter, as we are not performing EPI requests in our application. On the WebSphere Administrative Console, we expanded Resources, clicked **Resource Adapters**, selected the node on which we want to install the resource adapter ,and then clicked **Install RAR** to start the installation process, as shown on Figure 5-15.

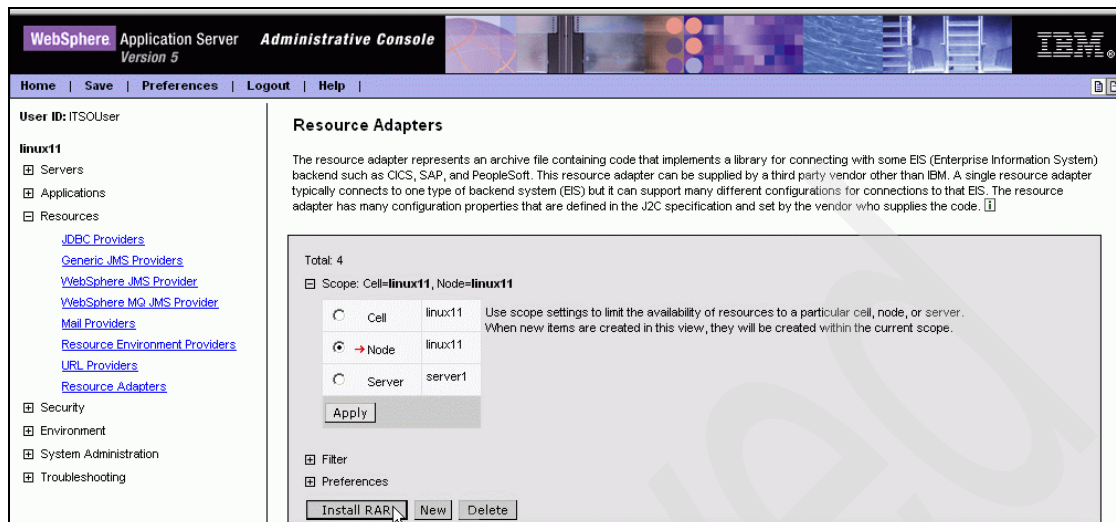


Figure 5-15 WebSphere for z/OS Administrative Console

On the next panel (Figure 5-16 on page 106), we clicked **Server path** for our desired node and typed in the location path on the Linux server where the CICS RAR resides. We then clicked **Next**. As mentioned previously, the RAR file is in the directory `/opt/ctg/deployable`.

Note: The target node cannot be the deployment manager node.

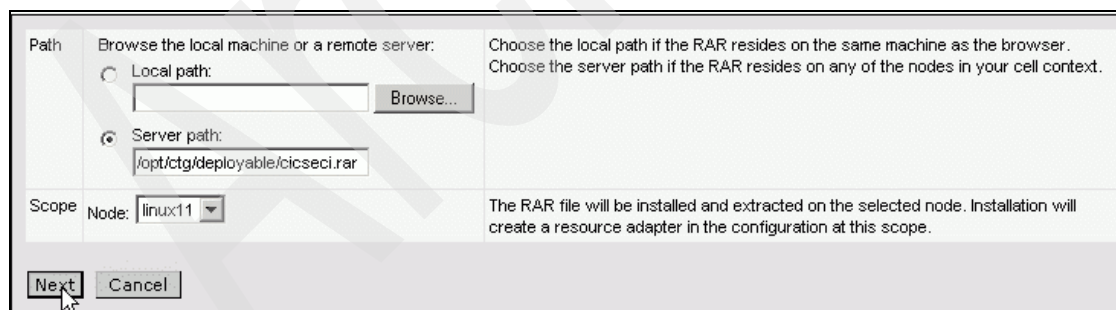


Figure 5-16 Identify the RAR file

Tip: Resource Adapter Archive (RAR) files can be installed from either of two locations. The file can be loaded from a path on the local workstation to a browser or from a path on the server.

We recommend using the RAR file (cicseci.rar) shipped with WSAD.IE. This file contains a JAR file (cicsecitools.jar) that is not included in the RAR shipped with CICS Transaction Gateway for z/OS. If you deployed an application developed with WSAD.IE, which uses WSIF, then this JAR file will be required.

A Resource Adapter configuration panel will be displayed (Figure 5-17 on page 107). We entered ECIconnector as a name for this resource adapter and a description and clicked **OK**.

The screenshot shows a configuration window titled "Configuration" with a "General Properties" section. The fields are as follows:

Property	Value	Description
Scope	* cells:linux11:nodes:linux11	The scope of the configured resource provider. This value indicates the configuration location for the configuration file.
Name	ECIconnector	The name of the resource adapter. If this property is not specified, the value of the display-name in the deployment descriptor will be used.
Description	Resource Adapter for CICS ECI	A text description for the resource provider.
Archive Path		Path to the installed RAR file containing the module for this resource adapter. If this property is not specified, the archive will be extracted to the absolute path represented by the symbolic name "CONNECTOR_INSTALL_ROOT". "CONNECTOR_INSTALL_ROOT" is a variable defined in variables.xml file.
Classpath		A list of paths or JAR file names which together form the location for the resource provider classes. Classpath entries are separated by using the ENTER key and must not contain path separator characters (such as ';' or ':'). Classpaths may contain variable (symbolic) names which can be substituted using a variable map. Check your drivers installation notes for specific JAR file names which are required.
Native Path		An optional path to any native libraries (.dll's, .so's). Native path entries are separated by using the ENTER key and must not contain path separator characters (such as ';' or ':'). Native paths may contain variable (symbolic) names which can be substituted using a variable map.

At the bottom of the window are buttons for "OK", "Reset", and "Cancel".

Figure 5-17 Resource Adapter configuration

This adapter is now displayed as being install on this node (Figure 5-18 on page 108).

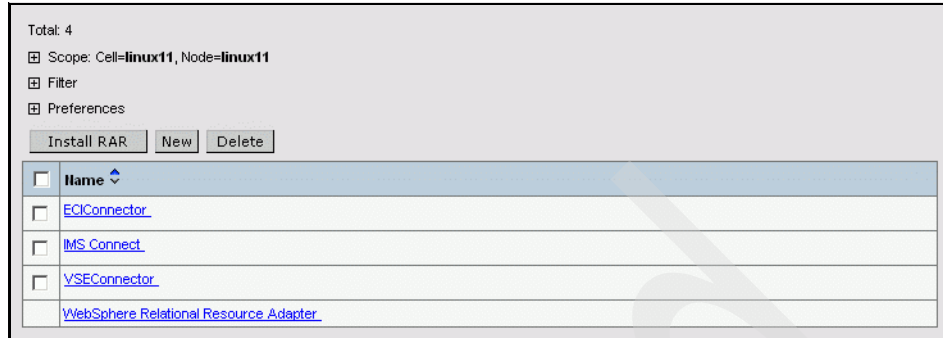


Figure 5-18 List of installed Resource Adapters

We could have saved our configuration changes at this time but we wanted to configure a J2C connection factory for the CICS ECI resource adapter.

5.5.2 Configuring a J2C connection factory for z/OS

We clicked the just installed ECIConnector resource adapter to display its Configuration panel (Figure 5-19 on page 108).

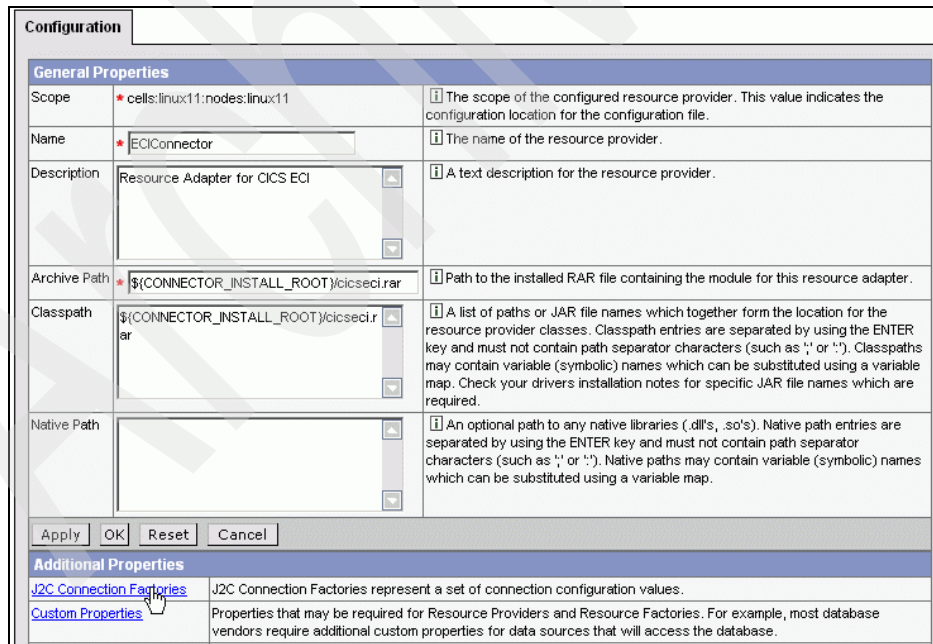


Figure 5-19 CICS Transaction Gateway Configuration pane

We next clicked **J2C Connection Factories**, as shown in Figure 5-19, at the bottom of the ECIconnector - Configuration panel to display the J2C Connection Factories panel (not shown). On this panel we clicked **New** to open the New Configuration panel.

Important: There are two types of CICS ECI J2C connection factories. One directly accesses a CICS region using XCF (LOCAL), which can be used if WebSphere Application Server is running on the same z/OS as CICS, and the other accesses a CICS Transaction Gateway task using TCP/IP (REMOTE).

On the Resource Adapter → CICS ECI → J2C Connection Factories - New panel (Figure 5-20 on page 109), we entered CICS ECI Connection Factory as the name for the factory, and for its JNDI name we entered `its/cics/eci/j2ee/trader/TraderCICSECICCommandCICSECIServiceTraderCICSECICCommandCICSECIPort`. This was the name that the CICS J2EE application, called TraderCICSEAR, used.

Configuration		
General Properties		
Scope	cells:linux11:nodes:linux11	The scope of the configured resource. This value indicates the configuration location for the configuration file.
Name	CICS ECI Connection Factory	The required display name for the resource.
JNDI name	its/cics/eci/j2ee/trader/TraderCICSE	The JNDI name for the resource, including any naming subcontexts. The name is used to link the platform binding information. The binding associates the resources defined the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.
Description	Connection Factory for ECI	An optional description for the resource.
Category		An optional category string which can be used to classify or group the resource.
Authentication Preference	BASIC_PASSWORD	Specifies which of the authentication mechanisms which are defined for the corresponding resource adapter applies to this connection factory. For example, if two auth mechanism entries have been defined for a resource adapter, KerbV5 and Basic Password, this will specify one of those two types. If the auth mechanism preference specified is not an available auth mechanism on the corresponding resource adapter, it is ignored. Default=BASIC PASSWORD
Component-managed Authentication Alias		References authentication data for component-managed signon to the resource.
Container-managed Authentication Alias		References authentication data for container-managed signon to the resource.

Figure 5-20 CICS local general properties

We then clicked **Apply** to display Additional Properties at the bottom of the panel (Figure 5-21 on page 110).

Note: Component-managed Authentication Alias is used to provide a user ID/password when the application's res-auth deployment descriptor specifies Application and there is no explicit user ID/password provided on the connection request.

Container-managed Authentication Alias is used to provide a user ID/password when the application's res-auth deployment descriptor specifies Container.

Additional Properties	
Connection Pool	An optional set of connection pool settings.
Custom Properties	Properties that may be required for Resource Providers and Resource Factories. For example, most database vendors require additional custom properties for data sources that will access the database.
Related Items	
J2C Authentication Data Entries	Specifies a list of userid and password for use by Java 2 Connector security.

Figure 5-21 J2C connection factory Additional Properties panel

We clicked **Custom Properties** to display the Custom Properties panel for this connection factory (Figure 5-22 on page 110). On this panel we entered a ServerName (SCSCERW3), the ConnectionURL (local:), and the PortNumber.

[Resource Adapters](#) > [ECIConnector](#) > [J2C Connection Factories](#) > [CICS ECI Connection Factory](#) >

Custom Properties

Custom properties that may be required for Resource Providers and Resource Factories. For example, most database vendors require additional custom properties for data sources that will access the database. [i]

Total: 12

Filter

Preferences

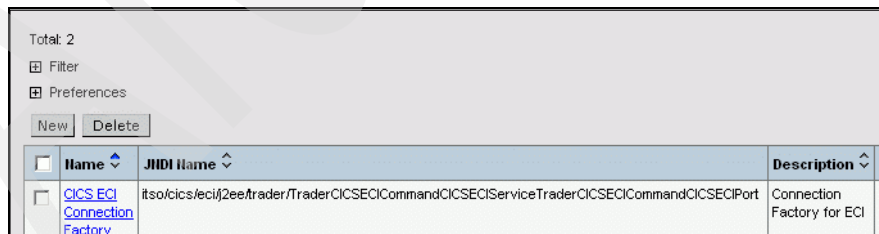
Name	Value	Description	Required
ServerName	SCSCERW3	ServerName	false
ConnectionURL	local:	ConnectionURL	false
PortNumber	2006	PortNumber	false
UserName	linux1	UserName	false
Password	linux1	Password	false
ClientSecurity	-	ClientSecurity	false
ServerSecurity	-	ServerSecurity	false
KeyRingClass	-	KeyRingClass	false
KeyRingPassword	-	KeyRingPassword	false
TranName	-	TranName	false
TPNName	-	TPNName	false
TraceLevel	1	TraceLevel	false

Figure 5-22 CICS J2C Custom Properties panel

The properties are:

- ▶ **ConnectionURL** - Must be set to local: for a local connection factory.
- ▶ **ServerName** - The name of the CICS server to which CTG will connect for all interactions for this connection factory. This is the name of the server definition in the CTG.INI file.
- ▶ **PortNumber** - We are using the default CTG port number of 2006.
- ▶ **TraceLevel**:
 - 0 - No tracing or logging occurs.
 - 1 - Only errors and exceptions are logged.
 - 2 - Errors and exceptions plus the entry and exit of important methods are logged.
 - 3 - Errors and exceptions, the entry and exit of important methods, and the contents of buffers sent and received are logged.
- ▶ **TranName** - The name of the CICS transaction under which the target application program will run. This does not affect the name of the transaction under which the mirror program is initially started.
- ▶ **Userid** - A default user ID to be used for this connection if no other is provided.
- ▶ **Password** - A password to be used with the above default user ID.

Tip: If you need to add additional J2C connection factories you can click **J2C Connection Factories** at the top of the panel to redisplay the list of J2C connection factories (Figure 5-23 on page 111) configured for this adapter and repeat this process; otherwise, save the changes.



<input type="checkbox"/>	Name	JNDI Name	Description	C
<input type="checkbox"/>	CICS ECI Connection Factory	itso/cics/eci/j2ee/trader/TraderCICSECICommandCICSECIServiceTraderCICSECICommandCICSECIPort	Connection Factory for ECI	

Figure 5-23 List of Installed J2C connection factories

After this step we saved our changes in WebSphere.

5.5.3 Configuring a J2C connection factory for VSE

This section shows the steps to define a second J2C connection factory to WebSphere, which is used to connect to the VSE CICS. We added this connection factory to the already defined ECI resource adapter.

1. Click **Resources** → **Resource Adapters**.
2. Click the already defined ECI resource adapter.

<input type="checkbox"/>	Name ▾
<input type="checkbox"/>	ECIConnector
<input type="checkbox"/>	IMS Connect
<input type="checkbox"/>	VSEConnector
	WebSphere Relational Resource Adapter

3. Click **J2C Connection Factories**.

Additional Properties	
J2C Connection Factories	J2C Connection Factories repre
Custom Properties	Properties that may be required vendors require additional custo
View Deployment Descriptor	View the Deployment Descripto

4. Add a new connection factory with the following properties (see below figure).
Then click **Apply**.

General Properties	
Scope	* cells:linux:11:nodes:linux:11
Name	* CICS ECI VSE Connection Factory
JNDI name	/itso/cics/eci/j2ee/trader/TraderCICSE

5. Scroll down and click **J2C Authentication Entries**.

Additional Properties	
Connection Pool	An optional set of connection pool settings.
Custom Properties	Properties that may be required for Resource Manager. For more information, see Additional Custom Properties for Data Sources .
Related Items	
J2C Authentication Data Entries	Specifies a list of user ID and password entries.

6. Add a new entry specifying a valid VSE user ID and its password.

General Properties	
Alias	* linux11/VSE user for ECI
User ID	* HAER
Password	* ****
Description	
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>	

7. In the connection factory, select this J2C authentication entry for component-managed and container-managed authentication alias. Click **Apply**.

Note: Specifying a value for container-managed authentication is not necessary, because only the component-managed authentication entry is used to log on to VSE, but WebSphere issues warnings when it is not specified. So we used the same entry for both definitions.

Component-managed Authentication Alias	linux11/VSE user for ECI
Container-managed Authentication Alias	linux11/VSE user for ECI
Mapping-Configuration Alias	DefaultPrincipalMapping

8. Scroll down and click **Custom Properties**.

Additional Properties	
Connection Pool	An optional set of connection pool settings.
Custom Properties	Properties that may be required for Resource additional custom properties for data source

9. Enter the following values (see figure below).

Note: In our scenario we had to specify local: for the connection URL. Entering the IP address of the Linux system did not work.

Name ▾	Value ▾
ServerName	VSE270
ConnectionURL	local:
PortNumber	2006
UserName	HAER
Password	MYPASSW
ClientSecurity	-
ServerSecurity	-
KeyRingClass	-
KeyRingPassword	-
TranName	-
TPNName	-
TraceLevel	1

10. Save your definitions. We had to restart WebSphere at this point to make the changes active.

5.5.4 Deploying the application in WebSphere

The JNDI name specified in the application needed to be resolved to a JNDI name of one of our J2C connection factories.

During the installation of the TraderCICS application, a panel was presented that listed the JNDI names found in the application. Selecting the pull-down displayed a list of the available J2C connection factories. We selected the desired factory and the desired modules and clicked **Apply**.

Each EJB reference defined in your application must be mapped to an Enterprise bean.

Module	EJB	URI	Reference Binding	Class	JNDI
TraderCICSWeb		TraderCICSWeb.war/WEB-INF/web.xml	itso/cics/eci/j2ee/trader/TraderCICSECHome	itso.cics.eci.j2ee.trader.TraderCICSECI	itso/

OK Cancel

Figure 5-24 Map resource references to resources

We then continued with the installation and successfully executed the application.

Tip: The J2C connection factory can be modified later without re-installing the application.

5.5.5 Implementing application security in WebSphere

Each application that accesses a J2C resource has a resource deployment descriptor (res-auth) that determines the authentication behavior when accessing the resource. There are two options:

- ▶ Container authentication

When container authentication is specified, the user ID and password used on the connection are provided by the container. The user ID and password combination is provided to the container by the J2C connection factory in the Container-managed Authentication Alias entry in the Configuration - General Properties panel of the J2C connection factory.

- ▶ Application authentication

When application authentication is specified the user ID and password used on the connection are provided explicitly by the application or by the J2C connection factory in the Component-managed Authentication Alias entry in the Configuration - General Properties panel of the J2C connection factory.

We wanted the application to provide the user ID and password, so we change the authentication deployment descriptor from container to application.

5.6 Problem determination

To perform problem determination we recommend enabling the following WebSphere traces:

- ▶ `com.ibm.ejs.j2c.*=all=enabled`
- ▶ `com.ibm.connector2.*=all=enabled`

And setting the trace level in the J2C connection factory custom property to 3.

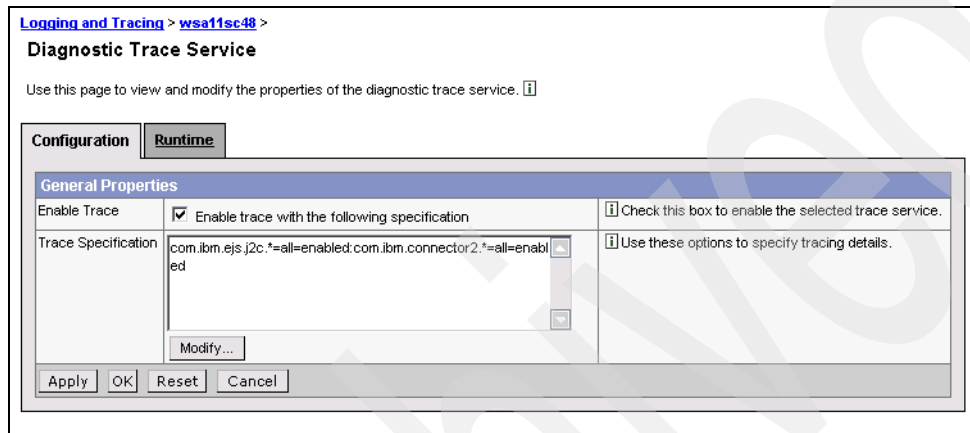


Figure 5-25 Diagnostic traces

5.6.1 Common errors

The following are some common errors:

- ▶ CTG9630E: IOException occurred in communication with CICS.
 - Explanation: A configuration error has occurred.
 - Usual cause: Review the JNI trace for the root cause. Some possibilities are:
 - RRS register return code 0x300 - CTG has not been properly configured to use RRS.
 - EXCI reason code 403 - CTG is unable to contact the target CICS server because of an invalid pipe name.
- ▶ CTG9631E: Error occurred during interaction with CICS. Error Code=ECL_ERR_NO_CICS minor code: 0 completed:
 - Explanation: CTG is unable to contact the target CICS server.

- Usual cause: EXCI reason code 203 - the target CICS server is not active or has not opened IRC communicatoin.

CTG9631E and AZI4 are typically the same problem:

▶ AZI4:

- Explanation: An error occured when executing a program on CICS. Although the message and codes book says that IRC has not been enabled, we found that an AZI4 presented to us on LINUX corresponds to ECI_ERR_NO_CICS.
 - Check the TCPIPService and make sure it is open.
 - Check the TCPIP in z/OS and make sure it is listening on that port. Possibly recycle the TCPIPService in CICS.
 - Make sure your port in WebSphere (or line command) when executing the program uses the correct port in CICS or the CTG.
 - Make sure the port you specified in the CTG uses the correct PORT.

▶ CTG9631E: Error occurred during interaction with CICS. Error Code=ECI_ERR_SECURITY_ERROR.

- Explanation: An error occur either validating a user ID, or a RACF® authorization failure has occurred.
- Usual cause: Review the JNI trace for the root error. Some possibilities are:
 - EXCI reason code 423 - RACF surrogate checking has failed.
 - RACF return code 143 - The user ID is unknown or not defined to RACF or does not have an OMVS RACF segment.

▶ Return code - 22:

Explanation: The connection to the gateway was successful, however, the program does not exist. See Example 5-26 on page 118.

▶ Hang after connect to gateway message:

- Explanation: The port of the remote gateway cannot be found.
- Check you port definitions.

▶ Return code - 3:

Explanation: The server definition that you used does not exist.

```

=== Connect to Gateway ===
Successfully created JavaGateway
=== Available Servers ===
System : SCSCERW3, Description : CICS TCP/IP
System : VSE260, Description :
System : VSE270, Description :
System : ERW3REM, Description : Using CTG On z/OS
System : SCSCERWX, Description : CICS TS on z/OS

=== Call Programs ===
About to call : EC220
  Commarea   : Hello
  Extend_Mode : 0
  Luv-Token  : 0
Commarea    : Hello
Return code : -22
Abend code  : null
Successfully closed JavaGateway
linux11:/opt/ctq/samples/java #

```

Figure 5-26 Example of TestECI or EciB1 test invocation

For more information on the CTG you can refer to the following documentation:

- ▶ Chapters 5 and 10 of *CICS Transaction Gateway V5 The WebSphere Connector for CICS*, SG24-6133
- ▶ *CICS Transaction Gateway V5.0: Unix Administration*, SC34-6190
- ▶ *CICS TS V2.2 External Interfaces Guide*, SC34-6006
- ▶ *UNIX System Services Messages and Codes*, SA22-7807

Using SOAP to communicate with CICS

Simple Object Access Protocol (SOAP) for CICS is an XML-based connectivity solution that enables CICS applications to provide and request services independently of platform, environment, application language or programming model.

This chapter gives an overview of SOAP support in CICS TS 2.2 for z/OS and CICS TS 1.1 for VSE/ESA. It also describes an implementation example of a stand-alone Java application using SOAP to access CICS TS 1.1 on VSE/ESA.

We discuss the following:

- ▶ SOAP overview
- ▶ Configuring CICS on VSE for SOAP support
- ▶ Compiling the SOAP service on VSE
- ▶ Testing the SOAP communication
- ▶ Writing your own SOAP programs on VSE
- ▶ Considerations on using SOAP in WebSphere

6.1 SOAP overview

The SOAP for CICS feature on z/OS and VSE/ESA provides a mechanism that allows CICS applications, written in any supported programming language, to communicate via SOAP. Transports are provided over Hypertext Transfer Protocol (HTTP) and WebSphere MQ (z/OS only). Both inbound and outbound function is provided. The implementation supports SOAP 1.1.

On z/OS, the SOAP for CICS feature delivers an enhanced level of the function already available as a Technology Preview in SupportPac CA1M, as a fully supported product for use in production.

On VSE/ESA 2.7, the SOAP for CICS feature is part of the operating system. This includes:

- ▶ A SOAP server that is running in CICS on the basis of CICS Web Support (CWS). It allows to call a WebService that is implemented as a CICS program from any kind of WebService client (for example Apache, AXIS, Microsoft® .Net (C#), and so on).
- ▶ A SOAP client that can be used from a CICS program to invoke a WebService that is running on any kind of WebService provider (for example WebSphere with Apache or AXIS, and so on).

Besides the SOAP client and server parts, there is an XML parser, which can also be used in user written CICS programs.

Figure 6-1 shows the SOAP scenario.

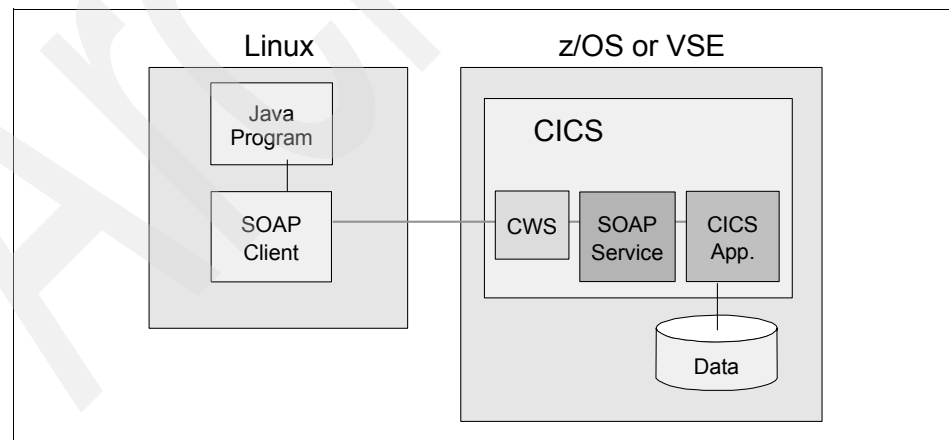


Figure 6-1 Using SOAP to communicate with CICS

In this scenario, a stand-alone Java program uses SOAP-related class libraries to call a CICS application on the backend host, which can be either z/OS or VSE. The call goes over TCP/IP to a CICS Web Support (CWS) based listener, which calls the SOAP service, which in turn calls a CICS application.

The SOAP for CICS feature enables existing or new CICS applications, written in any supported programming language, to communicate outside of the CICS environment via the Simple Object Access Protocol (SOAP). Message transport is provided over Hypertext Transfer Protocol (HTTP), using functions of CICS Web Support (CWS), and includes support for Secure Sockets Layer (SSL) via HTTPS. Message transport is also provided over WebSphere MQ (z/OS only). The SOAP protocol is supported at the Version 1.1 level.

The SOAP for CICS feature enables a user-written application layer to map the XML-based SOAP message into a COMMAREA, thus enabling access to COMMAREA-based applications using SOAP messages. SOAP for CICS not only permits existing CICS COMMAREA-based applications to be driven via XML-formatted SOAP messages, it can also be used for new applications driven via SOAP messages, and enables CICS applications to issue outbound SOAP messages targeted via SOAP or XML messaging to remote applications.

This feature will help to maximize the reuse of enterprise assets via standard interfaces, enhancing the value of existing applications in the CICS environment.

More details about SOAP can be found in the *WebSphere Version 5 Web Services Handbook*, SG24-6891.

6.1.1 SOAP on z/OS

SOAP for CICS requires CICS Transaction Server Version 2.2 or 2.3, and has the same hardware requirements as the base product. Please refer to IBM Software Announcements 201-354 and 203-296.

6.1.2 SOAP on VSE

SOAP for CICS on VSE requires CICS Transaction Server Version 1.1 and VSE/ESA 2.7 or VSE/ESA 2.6 with PTF UQ81044.

On the VSE side, the SOAP server is based on CICS Web Support (CWS). A detailed description of how to activate the SOAP server and an overview of the IBM-provided SOAP sample is contained in the manual *VSE/ESA e-business Connectors User's Guide*, SC33-6719. The VSE Connector Client provides the same information as the online documentation, including the source code of all

Java and C-sources, compile and link jobs. You can download the latest version of the VSE Connector Client also from:

<http://www-1.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/>

The following software prerequisites must be met in order to get SOAP on VSE to run:

- ▶ If running VSE 2.6, the following VSE PTF must be applied:
APAR PQ78973/PTF UQ81044
- ▶ If running VSE 2.6 or VSE 2.7, the following CICS PTFs must be applied:
 - APAR PQ51395/PTF UQ57278
 - APAR PQ60964/PTF UQ75840
 - APAR PQ73753/PTF UQ76951

6.2 Configuring CICS on VSE for SOAP support

CICS has to be configured in order to accept incoming TCP/IP traffic destined for a SOAP server running under the control of CICS. This section describes the necessary setup steps on VSE. The z/OS-related configuration is similar.

The following configuration steps have to be done:

1. Specify TCP/IP=YES in CICS setup.
2. Define the symbolic name of the VSE system in TCP/IP.
3. Define a TCP/IP service.
4. Activate the ASCII to EBCDIC converter.

The following sections describe each of these steps.

6.2.1 Step 1: Specify TCP/IP=YES in CICS setup

Make sure that TCPIP=YES is defined either in your DFHSIT or in your CICS startup job.

Example 6-1 Specify TCPIP=YES in DFHSIT or CICS startup job

```
// EXEC DFHSIP,SIZE=DFHSIP,PARM='APPLID=&XAPPLF2.,START=&XMODEF2.,EDSAL*  
IM=&ELIM.,SI',DSPACE=2M,OS390  
SIT=SP,STATRCD=OFF,MXT=20,NEWSIT=YES,TCPIP=YES,
```

Put the LIBDEF to your VSE library, where you have cataloged the SOAP application, into the CICS job if necessary.

Example 6-2 Make your private VSE library known to CICS

```
* $$ JOB JNM=CICSICCF,CLASS=2,DISP=L
* $$ LST CLASS=A,DISP=D,RBS=100
// JOB CICSICCF          CICS/ICCF STARTUP
// OPTION SADUMP=5
// OPTION SYSDUMPC
// UPSI 11100000
// LIBDEF *,SEARCH=(PRD2.CONFIG,PRD1.BASED,PRD1.BASE,PRD2.PROD, X
                    PRD2.SCEEBASD,PRD2.SCEEBASE,PRD2.DBASE,PRD1.MACLIBD, X
                    PRD1.MACLIB,PRIMARY.JSCH),PERM
...

```

6.2.2 Step 2: Define the symbolic name of VSE to TCP/IP

You have to make the symbolic name of your VSE system known to CICS, because CICS internally references this name when starting its TCP/IP service. You can make this definition either online at the VSE console via the TCP/IP command.

```
DEFINE NAME,NAME=VSEDEMO,IPADDR=9.152.82.82
```

Or put this statement in your TCP/IP startup member (IPINITxx.L).

Another way of enabling CICS to look up the symbolic name of the VSE system is to have the symbolic name defined in a name server. In this case you would just make this name server known to TCP/IP for VSE with the TCP/IP command:

```
SET DNS1=ip addr or name of name server
```

6.2.3 Step 3: Define the TCP/IP service

The SOAP server needs a TCP/IP listener in order to receive data from a client. This listener is defined as a TCP/IP service as shown in Example 6-3.

Example 6-3 Define the TCP/IP service to CICS

```
CEDA DEFine TCpipservice(          )
TCpipservice ==> SOAP
Group        ==> VSESPG
Description  ==> TCP/IP SERVICE FOR SOAP SERVER
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    ==>
Ippaddress   ==>
S0cketclose  ==> No          No | 0-240000
```

Now check if the TCP/IP service is open, as shown in Example 6-4.

Example 6-4 Check if TCP/IP is open

```
msg f2,data=cent i tcpip
AR 0015 1I40I  READY
F2 0112
      Tcpip
      Openstatus( Open )
      RESPONSE: NORMAL TIME: 22.56.52  DATE: 10.30.03
      SYSID=CIC1  APPLID=DBDCCICS
```

6.2.4 Step 4: Activate the ASCII to EBCDIC converter

An ASCII to EBCDIC converter is provided in ICCF library 59 as skeleton DFHCNV. It is used by CICS Web Support (CWS) to convert incoming XML data from ASCII to EBCDIC and outgoing data from EBCDIC to ASCII.

Just submit the skeleton DFHCNV, which catalogs a VSE phase DFHCNVBA into PRD2.CONFIG to be used as the character conversion table.

6.3 Compiling the SOAP service on VSE

Before testing the SOAP communication you have to compile the IBM-provided SOAP server sample. The source code is included in the VSE Connector Client. See the subdirectory /samples/soap.

The source code for the sample SOAP service consists of the following files:

- ▶ GETQUOTE.C, the program code
- ▶ IESSOAPH.H, the header file

The header file is shipped with VSE in PRD1.BASE.

For the complete source code of both files, refer to “IBM-provided SOAP service C program” on page 325 and “IBM-provided include file for SOAP” on page 330.

To compile and link the C source, you can use the following two jobs.

Compile job

Example 6-5 shows the compile job you can use for compiling the SOAP service. Replace the placeholder *XXXX* with your VSE sublibrary, where you want to catalog the phase. If it is a private library, you have to put it into the LIBDEF of the CICS startup job.

Example 6-5 Compile job for IBM-provided SOAP service

```
* $$ JOB JNM=GETQUOTE,DISP=D,CLASS=4
* $$ LST DISP=D,CLASS=Q,PRI=3
* $$ PUN DISP=I,DEST=*,PRI=9,CLASS=4
// JOB GETQUOTE CICS PRE TRANSLATE GETQUOTE
// ASSGN SYSIPT,SYSRDR
* *****
// EXEC IESINSRT
$ $$ LST DISP=D,CLASS=Q,PRI=3
$ $$ PUN DISP=I,DEST=*,PRI=9,CLASS=4
// JOB GETQUOTE COMPILE GETQUOTE
// ASSGN SYSIPT,SYSRDR
* *****
// EXEC IESINSRT
$ $$ LST DISP=D,CLASS=Q,PRI=3
// JOB GETQUOTE CATALOG OBJECT GETQUOTE
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.DBASE,PRIMARY.XXXX)
// EXEC LIBR
    ACCESS SUBLIB=PRIMARY.XXXX
    CATALOG GETQUOTE.OBJ REPLACE=YES
$ $$ END
* *****
// ON $CANCEL OR $ABEND GOTO ENDJ3
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.DBASE,PRIMARY.XXXX)
// OPTION ERRS,SXREF,SYM,CATAL,DECK,LISTX
// EXEC EDCCOMP,SIZE=EDCCOMP,PARM='LONGNAME SS SOURCE RENT TEST'
* $$ END
* *****
// ON $CANCEL OR $ABEND GOTO ENDJ2
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRD2.DBASE,PRIMARY.XXXX)
// OPTION ERRS,SYM,DECK,NOXREF
// EXEC DFHEDP1$,SIZE=512K,PARM='SP'
* $$ SLI MEM=GETQUOTE.C,S=PRIMARY.XXXX
/*
/. ENDJ2
* *****
// EXEC IESINSRT
/*
/. ENDJ3
* *****
// EXEC IESINSRT
/*
```

```

#&
$ $$ EOJ
$ $$ END
* *****
#&
$ $$ EOJ
$ $$ END
* *****
/&
* $$ EOJ

```

Link job

Example 6-6 shows the link job you can use to link the IBM-provided SOAP service. Replace the placeholder *XXXX* with your VSE sub library, where you want to catalog the phase. If it is a private library, you have to put it into the LIBDEF of the CICS startup job.

Example 6-6 Link job for the IBM-provided SOAP sample

```

* $$ JOB JNM=LINKGETQ,DISP=D,CLASS=4
* $$ LST DISP=D,CLASS=Q,PRI=3
// JOB LINKGETQ
// LIBDEF *,SEARCH=(PRD2.SCEEBASE,PRIMARY.XXXX)
// LIBDEF PHASE,CATALOG=PRIMARY.XXXX
// OPTION ERRS,SXREF,SYM,CATAL,NODECK,LISTX
    PHASE GETQUOTE,*,SVA
    INCLUDE GETQUOTE
    INCLUDE DFHELII
/*
// EXEC EDCPRLK,SIZE=EDCPRLK,PARM='UPCASE MAP'
/*
// EXEC LNKEDT,SIZE=256K
/*
/&
* $$ EOJ

```

6.4 Testing the SOAP communication

To test the SOAP communication, you need a SOAP client that uses the SOAP service on VSE to retrieve some data. The next section shows how to implement a simple Java client. The Java sample is taken from the following Web site and has been slightly modified to work with VSE.

<http://www.apache.org>

In this example, the Java client requests the stock quote of a given company from the SOAP service running on VSE. The SOAP service is a LE/VSE C program, which just returns a hard-coded value instead of really accessing a database or VSAM file.

6.4.1 Software prerequisites for the Java SOAP client

The Java SOAP client program is based on a couple of Java class libraries that you can download from the Web.

What you need are the following packages:

- ▶ soap.jar, which implements the SOAP-related functions. Download this from:
<http://xml.apache.org/soap>
- ▶ activation.jar, which contains the Sun JavaBeans Activation Framework (JAF). Download this from:
<http://java.sun.com/products/javabeans/glasgow/jaf.html>
- ▶ mail.jar, which implements the Sun Java Mail API. Download this from:
<http://java.sun.com/products/javamail>
- ▶ xerces.jar, which implements XML-related functions. Download the latest Xerces-J-bin package from:
<http://xml.apache.org/xerces-j/index.html>

6.4.2 Implementing a Java-based SOAP client

The Java SOAP client sample is the popular GetQuote.java sample taken from the Apache SOAP distribution. You can find the original file in the Apache SOAP directory soap-2_3_1\samples\stockquote\GetQuote.java.

Only one single line was changed to call the IBM-supplied getQuote SOAP service.

Example 6-7 IBM-supplied getQuote SOAP service

```
...
// Build the call.
Call call = new Call ();
call.setTargetObjectURI ("urn:iessoapd:getquote"); // CHANGED
call.setMethodName ("getQuote");
call.setEncodingStyleURI(encodingStyleURI);
Vector params = new Vector ();
params.addElement (new Parameter("symbol", String.class, symbol, null));
call.setParams (params);
```

...

You can find the complete source code of the java client in Appendix A, “VSE/ESA code samples” on page 319.

6.4.3 Running the Java-based SOAP client

The Java client can be run with the following Windows batch file. Here it is assumed that all necessary jar files are copied to the current directory. The example calls the CWS service IESSOAPS with parameter “IBM” to get the IBM stock quote.

Example 6-8 Running the SOAP client

```
set CLASSPATH=j2ee.jar;soap.jar;xerces.jar;mail.jar;activation.jar;%classpath%
java GetQuote http://9.152.82.82:1080/cics/CWBA/IESSOAPS IBM
pause
```

On the VSE side, you will see the following output on the operator console when running the SOAP client (Example 6-9).

Example 6-9 Output on operator console

```
F2 0103 SSRV: SOAPSERVER called
F2 0103 SSRV: Method = getQuote
F2 0103 SSRV: method getQuote
F2 0103 SSRV: stock symbol = IBM
F2 0103 SSRV: rc = 0
F2 0103 SSRV: SOAPSERVER finished
```

On the client side, the following output is shown (Example 6-10). The sample has been slightly modified to show some input parameters on the Java console.

Example 6-10 Client-side output

```
C:\SOAP\client>set
CLASSPATH=j2ee.jar;soap.jar;xerces.jar;mail.jar;activation.jar;.

C:\SOAP\client>java GetQuote http://9.152.82.82:1080/cics/CWBA/IESSOAPS IBM

url          = http://9.152.82.82:1080/cics/CWBA/IESSOAPS
symbol       = IBM
80.10

C:\SOAP\client>pause
Press any key to continue . . .
```

6.5 Writing your own SOAP programs on VSE

There is a detailed description of how to implement a host-based SOAP server and client in the *VSE/ESA e-business Connectors User's Guide*, SC33-6719, which you can download from:

<http://www-1.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/>

All referenced coding samples, compile and link jobs are provided with the *VSE Connector Client*, which is downloadable from the same Web page.

6.6 Considerations for using SOAP in WebSphere

There is a lot of information about SOAP and how it can be used in a WebSphere environment in the IBM Redbook *WebSphere Version 5 Web Services Handbook*, SG24-6891.

Archived

DB2 connectors

This chapter describes how to run a J2EE application on Linux for zSeries using DB2 connectors to access data on DB2 for z/OS and DB2 for VSE/ESA.

We describe the following:

- ▶ DB2 Connect scenario
- ▶ Installing DB2 Connect V8.1
- ▶ Customizing WebSphere Application Server for DB2 Connect
- ▶ Deploying TraderDB in WebSphere Application Server

7.1 DB2 Connect scenario

Figure 7-1 shows the DB2 Connect scenario with z/OS and VSE/ESA.

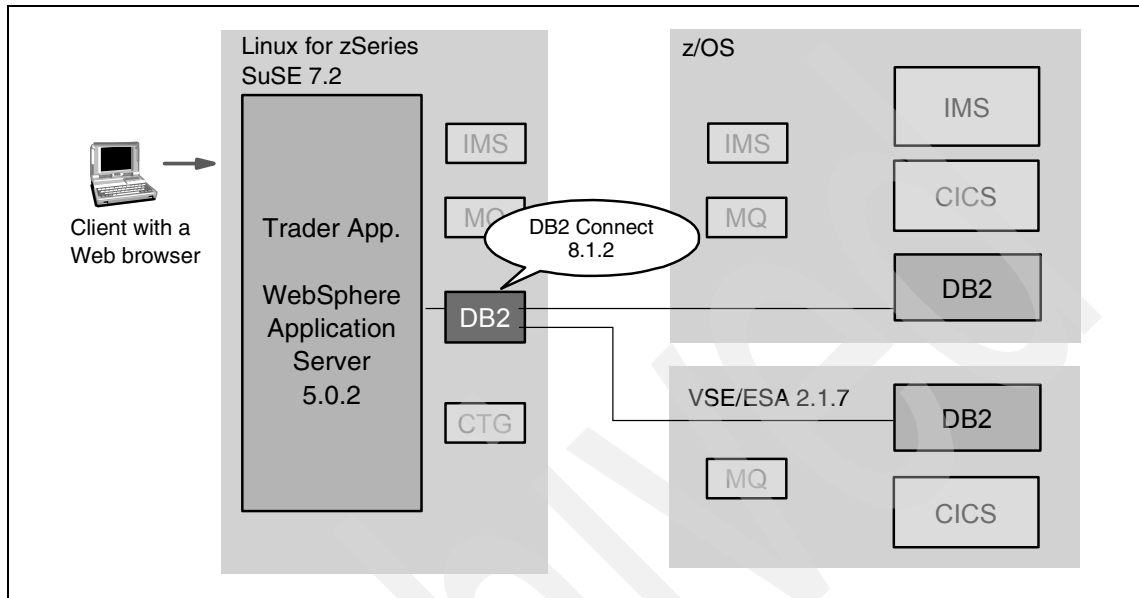


Figure 7-1 DB2 connect overview

You see the DB2 Connect component on Linux interfacing to the WebSphere Application Server on the left side. On the right side two database connections using TCP/IP are defined for data access, one for DB2 on z/OS and the other for DB2 on VSE/ESA. DB2 Connect uses the Distributed Relational Database Access (DRDA®) protocol.

7.2 Installing DB2 Connect V8.1

IBM DB2 Connect Enterprise Edition enables local and remote client applications to create, update, control, and manage DB2 databases on host systems using Structured Query Language (SQL), DB2 APIs, Open Database Connectivity (ODBC), Java Database Connectivity (JDBC), Embedded SQL for Java (SQLJ) or DB2 Call Level Interface (*CLI). In addition, DB2 connect supports Microsoft Windows data interfaces such as ActiveX Data Objects (ADO), Remote Data Objects (RDO), and Object Linking and Embedding (OLE) DB.

DB2 Connect Enterprise Edition is often installed on a dedicated intermediate server to allow multiple DB2 clients to connect to the host. This way the clients

are relieved from the DRDA overhead. In our scenario we installed the DB2 Connect on Linux to give multiple DB2 client applications running under the control of WebSphere access to the remote DB2 databases.

DB2 Connect for Linux on zSeries installs as follows:

1. Log in as root. You will need a graphical environment. Make sure your display is set up properly and your Windows X-server is started.
2. Unpack the DB2 Connect installation code and change the directory (cd) to the directory where you packed the DB2 Connect into.
3. Start the DB2 Connect setup by executing the db2setup script:

```
./db2setup
```

Click **Install Products**, as shown in Figure 7-2, to start the installation.

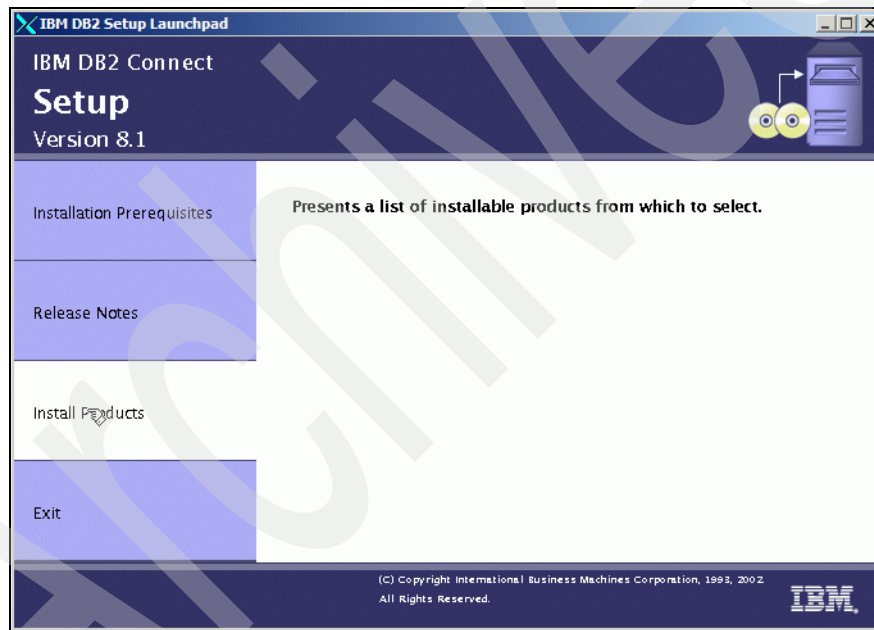


Figure 7-2 IBM DB2 Connect setup launchpad

4. Select the product(s) to install.
Select the **DB2 Connect Enterprise Edition**, as shown in Figure 7-3, and click **Next**.

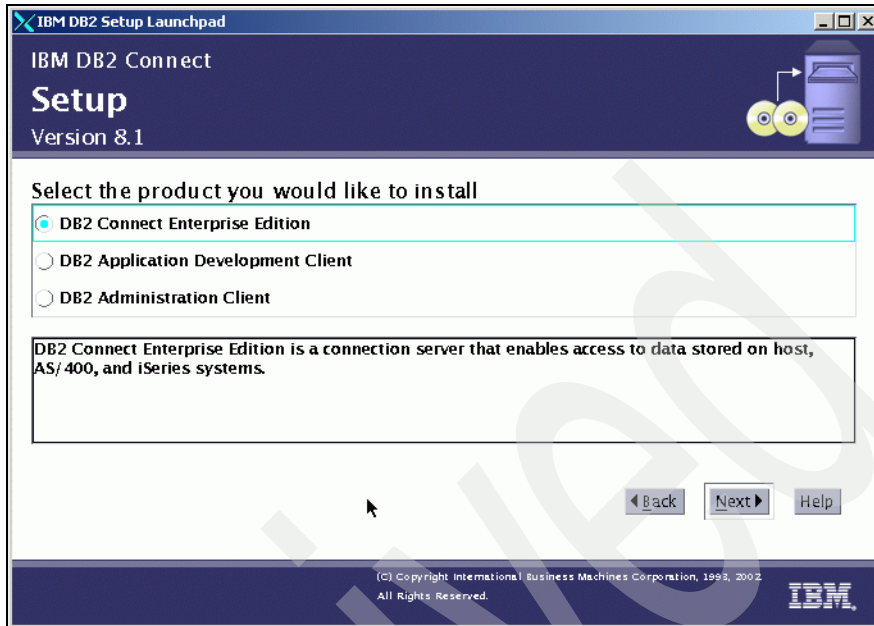


Figure 7-3 DB2 products to install

You will see an installation wizard introductory screen. Click **Next**. Review and accept the license. Click **Next** again.

5. Set up the DB2 Administration server user ID as shown in Figure 7-4.

Enter a new password twice for the administration server user dasusr1. Click **Next**.

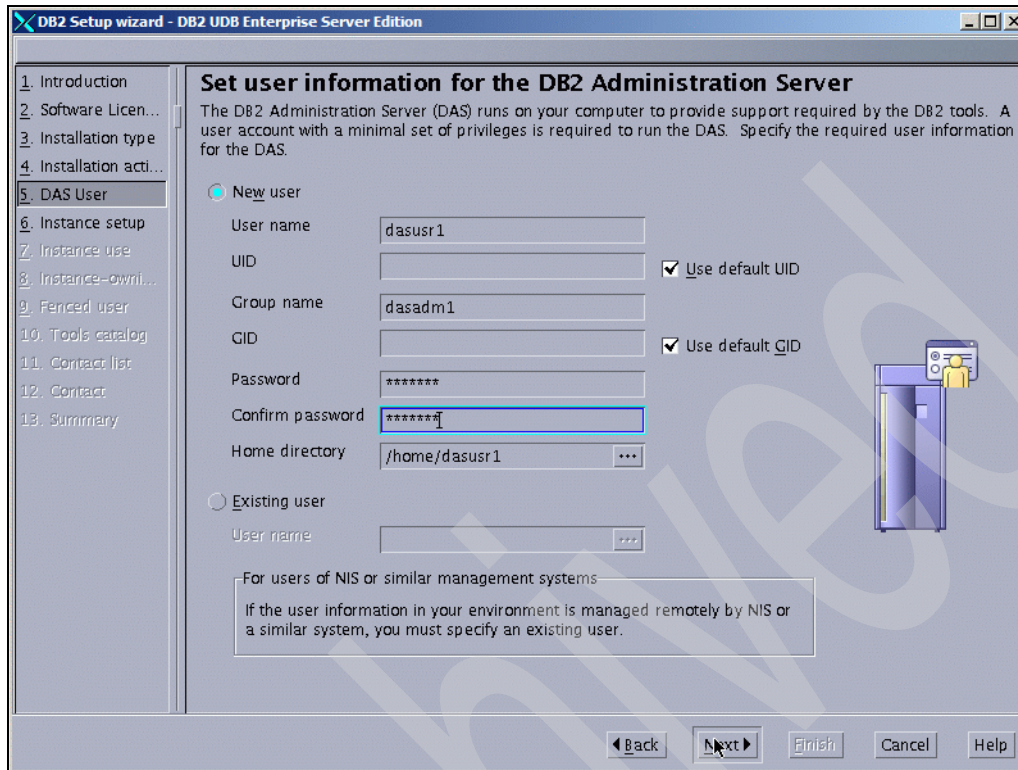


Figure 7-4 DB2 Administration Server user information

6. Create a DB2 instance.
7. Select the option to create a DB2 instance. Click **Next**.
8. Specify and confirm the password for the instance owner, db2inst1 as shown in Figure 7-5. Click **Next**.

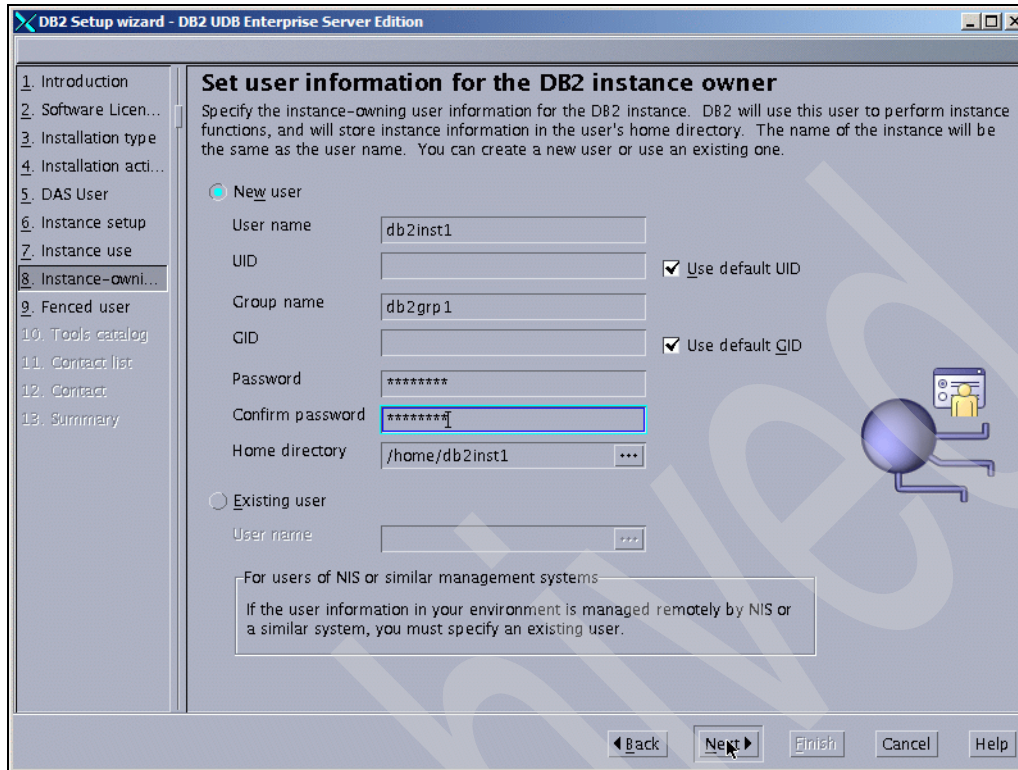


Figure 7-5 DB2 instance owner user information

- Specify and confirm the password for the fenced user, db2fenc1 as shown in Figure 7-6. Click **Next**.

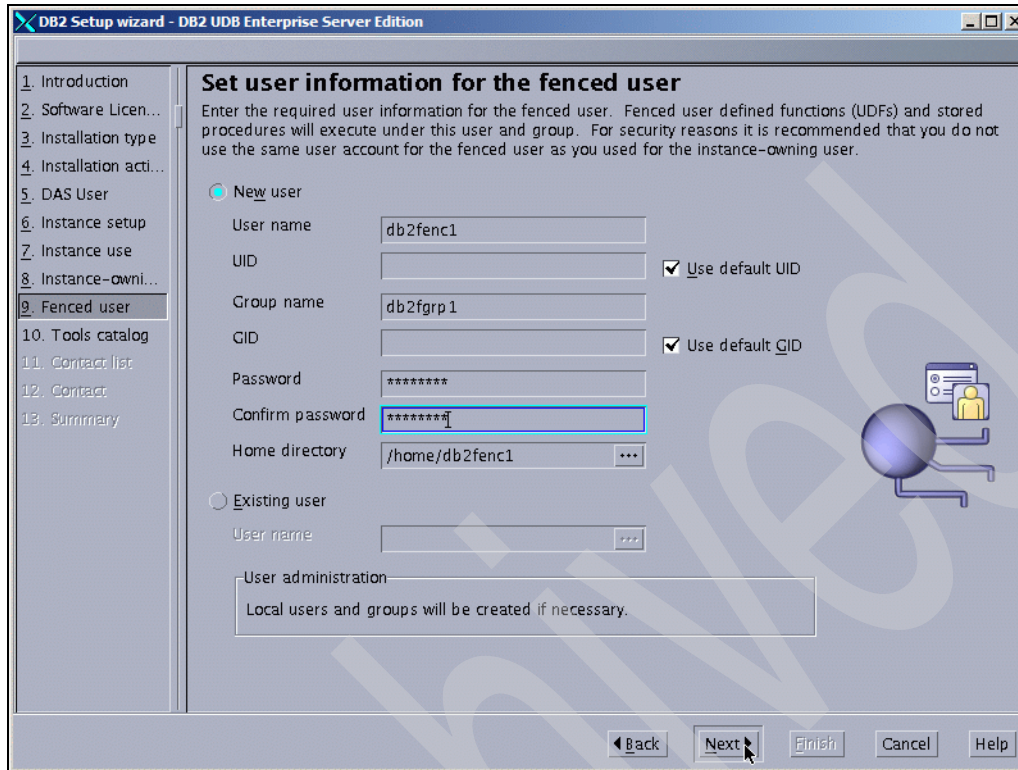


Figure 7-6 DB2 fenced user information

10. Prepare the DB2 tools catalog, as shown in Figure 7-7.

Accept the defaults to not prepare the DB2 tools catalog. Click **Next**.

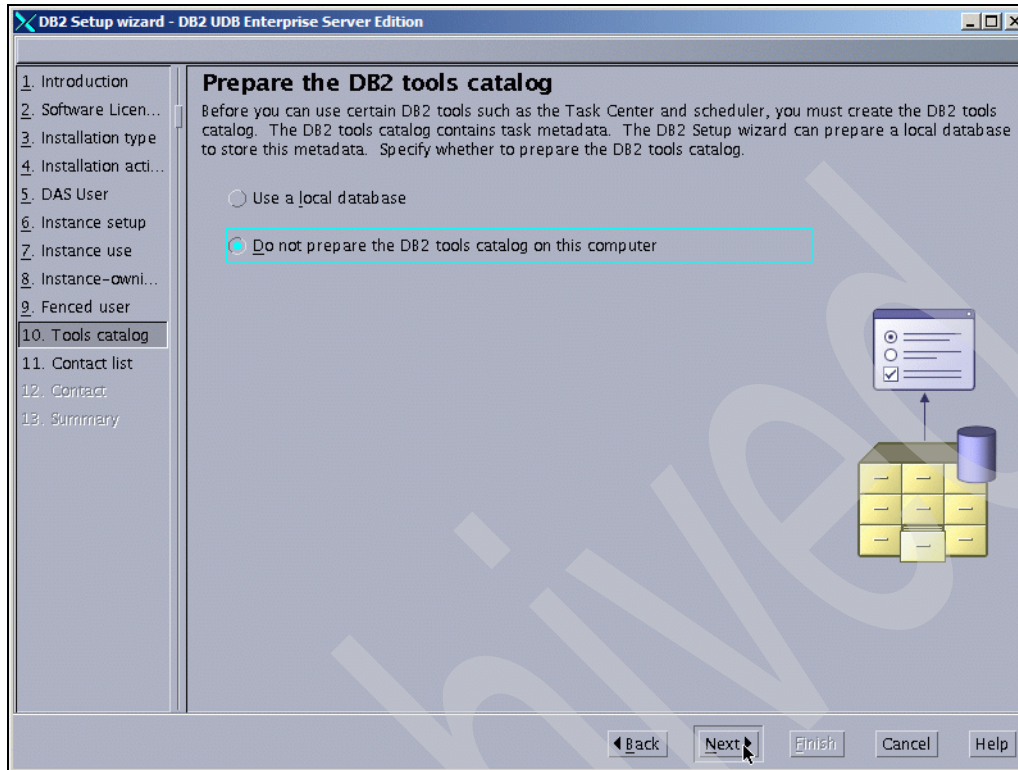


Figure 7-7 Prepare the DB2 tools catalog

11. You will get a chance to specify a user ID for notification. Since we did not have an SMTP server running at this time, we were still able to set up the contact list, but we disabled the Enable notification until later, as shown in Figure 7-8.

Accept the defaults to create a contact list on this system. Do not enable notification if you do not have an SMTP server running. Click **Next**.

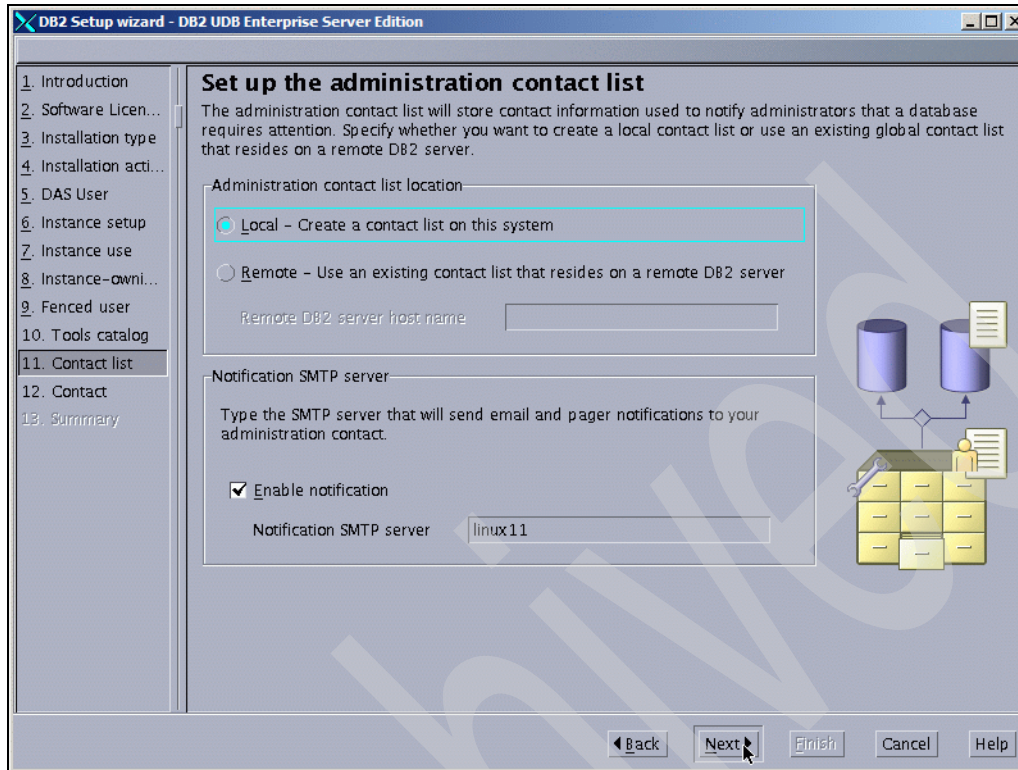


Figure 7-8 Administration contact list set up

Ignore the warning about the notification SMTP server. Click **OK**.

Accept the defaults for an ID to receive health notifications, as shown in Figure 7-9. Click **Next**.

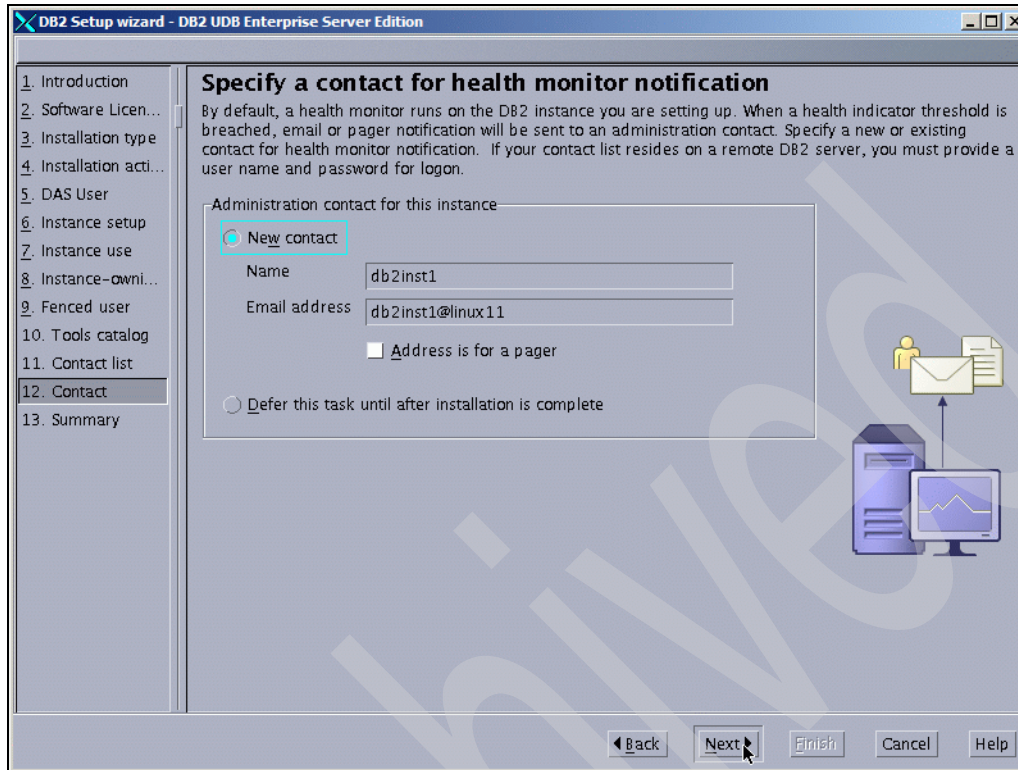


Figure 7-9 Specify a contact for health monitor notification

12. You should now see a summary of your choices. Our choices are summarized in Figure 7-10. Review your choices and click **Finish** to start copying files and setting up your system.

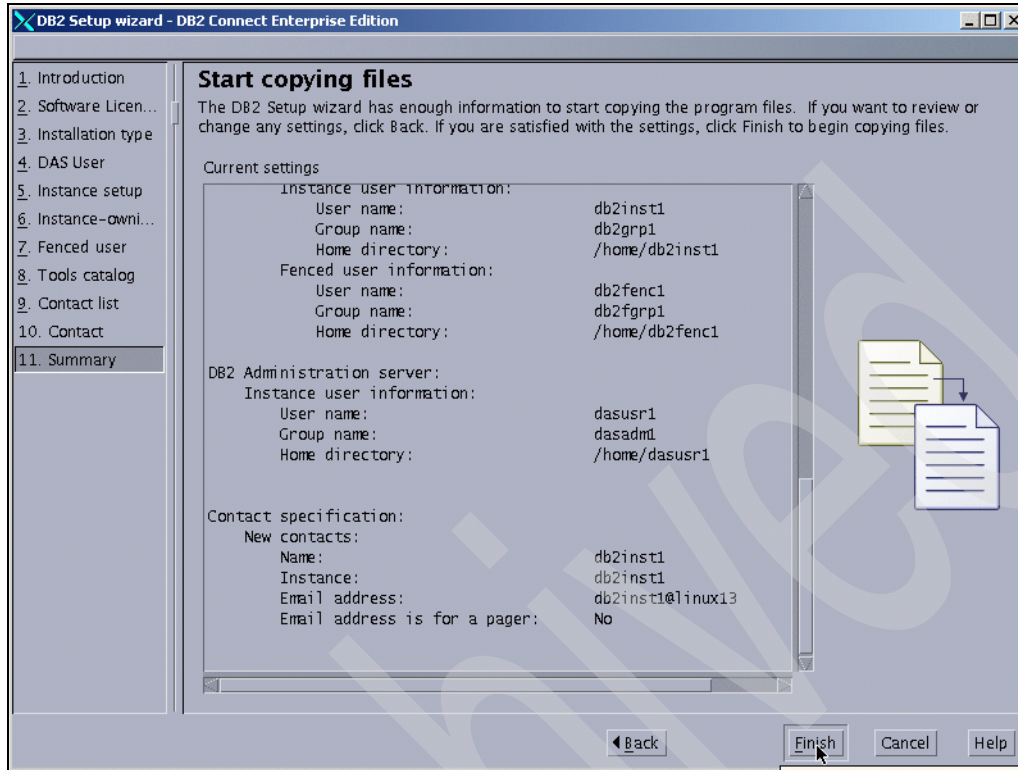


Figure 7-10 DB2 summary information

13. While the installation is completing, you will see a status bar indicating the progress of the install. When the installation completes, you should see a post-install summary.

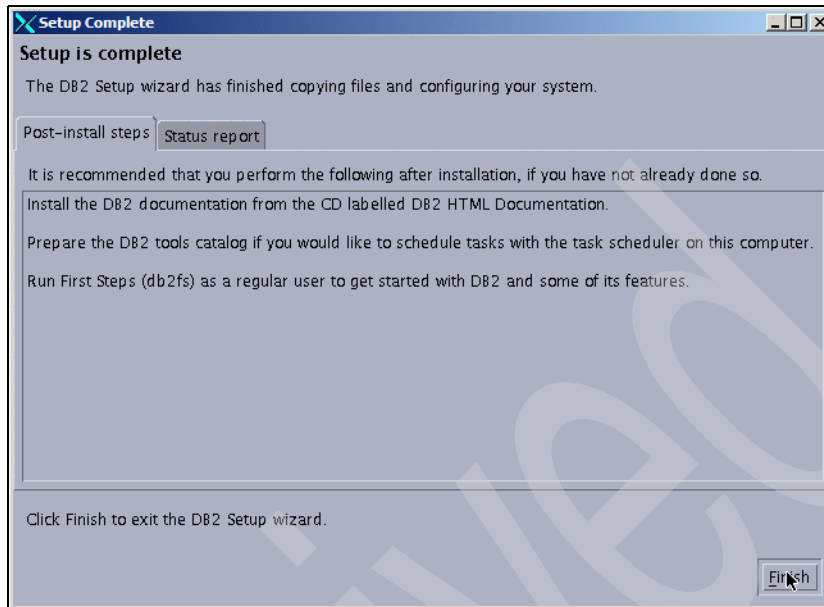


Figure 7-11 DB2 setup is complete

At this point you have installed DB2 Connect and its Administration Server and created an instance of DB2 and a DB2 fenced user. If you need to make additional changes later, you can always rerun `db2setup`.

7.2.1 Simple connect to DB2 for z/OS

The following line commands are what we used to test the connection to the Trader DB2 database on z/OS before we set up our WebSphere Application Server:

1. Catalog the remote node of the z/OS system:

```
db2 catalog tcpip node node_name remote ip_addr server port_number
```

- Where *node_name* is any name you want. You may use the DB2 remote location name for consistency.
- Where *ip_addr* is the IP address of the system where the DB2 subsystem resides.
- Where *port_number* is the TCP/IP port that DB2 is listening to (default is 446). This port number can be found in the TCPIP.PROFILES member for your TCP/IP procedure (search for DB2).

2. Catalog the data connection services:

```
db2 catalog dcs db node_name
```

Where *node_name* is the DB2 location name that you get from the syslog when DDF is started.

3. Map the remote DB2 database to a local database alias:

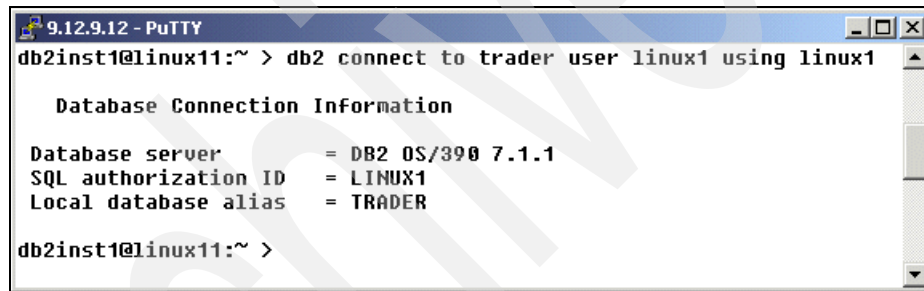
```
db2 catalog db node as alias at node node authentication dcs
```

- Where *node* equals *node_name* in step 2.
- Where *alias* is any name. We suggest that you chose *trader* for the Trader application.

4. Test the connection to the remote database using the alias:

```
db2 connect to trader user userID using password
```

Where *userID* and *password* are a user ID and a password known to DB2 on the z/OS server. You should then see something like Figure 7-12.



```
9.12.9.12 - PuTTY
db2inst1@linux11:~ > db2 connect to trader user linux1 using linux1

Database Connection Information

Database server      = DB2 OS/390 7.1.1
SQL authorization ID = LINUX1
Local database alias = TRADER

db2inst1@linux11:~ >
```

Figure 7-12 Connection to the Trader database has been tested

7.2.2 Simple connect to DB2 for VSE

The steps to connect to DB2 for VSE are very similar to those used for z/OS.

1. If necessary, switch the user to DB2INST1 for authorization purposes:

```
linux11:/ # su db2inst1
db2inst1@linux11:/ >
```

2. Catalog the remote node of the VSE system:

```
db2inst1@linux11:/ > db2 catalog tcpip node db2vse remote 9.152.82.82
server 446
DB20000I The CATALOG TCPIP NODE command completed successfully.
```

3. Catalog the data connection services:

```
db2inst1@linux11:/ > db2 catalog dcs db db2vse
DB20000I The CATALOG DCS DATABASE command completed successfully.
```

4. Map the remote DB2 database to a local database alias:

```
db2inst1@linux11:/ > db2 catalog db sqlds as tradedb at node db2vse
DB20000I The CATALOG DATABASE command completed successfully.
db2inst1@linux11:/ > db2 catalog dcs database tradedb as sqlds
DB20000I The CATALOG DCS DATABASE command completed successfully.
```

5. Test the connection to the remote database using the alias. The output should look similar to Example 7-1.

Example 7-1 Testing the connection to the sample database on VSE

```
db2inst1@linux11:/ > db2 connect to tradedb user mydbuser using mydbpw
```

Database Connection Information

```
Database server      = SQL/DS VSE 7.3.0
SQL authorization ID = MYDBUSER
Local database alias = TRADEDB
```

```
db2inst1@linux11:/ >
```

7.3 Customizing WebSphere Application Server for DB2 Connect

When DB2 Connect is installed and able to make a connection to the remote DB2 database, you need to customize WebSphere Application Server to be able to communicate with DB2 Connect, and you need to define in WebSphere Application Server the data sources.

7.3.1 Updating the WebSphere Application Server startup script

In order to avoid a `java.lang.UnsatisfiedLinkError` when WebSphere Application Server connects to the database, you have to make a few changes to the WebSphere startup script:

- ▶ Ensure that the `db2profile` has been run in the shell starting WebSphere.
- ▶ The shared library path environment variable, `LD_LIBRARY_PATH`, must be extended to include `<DB2 instance home>/sqllib/java12` and `<DB2 instance home>/sqllib/lib`.

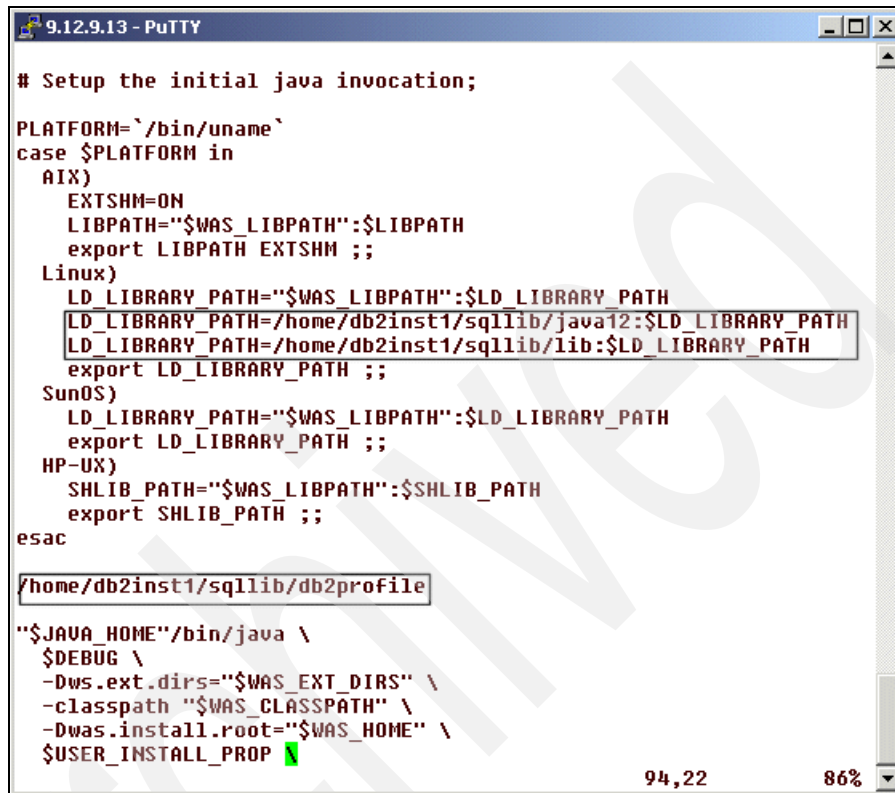
The changes are described in a Technote, which can be found at:

<http://www-1.ibm.com/support/docview.wss?uid=swg21110708>

1. Make the changes to the `startServer.sh` file. In Figure 7-13 on page 145 the changes are marked up.

2. Restart WebSphere Application Server in order to activate the changes:

- . /opt/WebSphere/AppServer/bin/stopServer.sh server1
- . /opt/WebSphere/AppServer/bin/startServer.sh server1



```
9.12.9.13 - PuTTY

# Setup the initial java invocation;

PLATFORM=`/bin/uname`
case $PLATFORM in
  AIX)
    EXTSHM=ON
    LIBPATH="$WAS_LIBPATH":$LIBPATH
    export LIBPATH EXTSHM ;;
  Linux)
    LD_LIBRARY_PATH="$WAS_LIBPATH":$LD_LIBRARY_PATH
    LD_LIBRARY_PATH=/home/db2inst1/sqllib/java12:$LD_LIBRARY_PATH
    LD_LIBRARY_PATH=/home/db2inst1/sqllib/lib:$LD_LIBRARY_PATH
    export LD_LIBRARY_PATH ;;
  SunOS)
    LD_LIBRARY_PATH="$WAS_LIBPATH":$LD_LIBRARY_PATH
    export LD_LIBRARY_PATH ;;
  HP-UX)
    SHLIB_PATH="$WAS_LIBPATH":$SHLIB_PATH
    export SHLIB_PATH ;;
esac

/home/db2inst1/sqllib/db2profile

"$JAVA_HOME"/bin/java \
  $DEBUG \
  -Dws.ext.dirs="$WAS_EXT_DIRS" \
  -classpath "$WAS_CLASSPATH" \
  -Dwas.install.root="$WAS_HOME" \
  $USER_INSTALL_PROP
```

Figure 7-13 Necessary changes to the startServer.sh file

7.3.2 Configuring a WebSphere data source

To configure:

1. Log in to WebSphere by typing `http://myserver:9090/admin/` in a browser.
2. In the WebSphere Administrative Console left panel menu choose **Resources** → **JDBC Providers**.
3. In the right panel, click **New** to create a new JDBC provider. Register a new JDBC provider
4. In the **Configuration** panel, select the DB2 driver that you want to use from the drop down list and click **OK**. We choose the “DB2 Legacy CLI-based type 2 JDBC Driver” as the one named “DB2 JDBC Driver” is now deprecated and

may be obsolete in future versions of DB2.

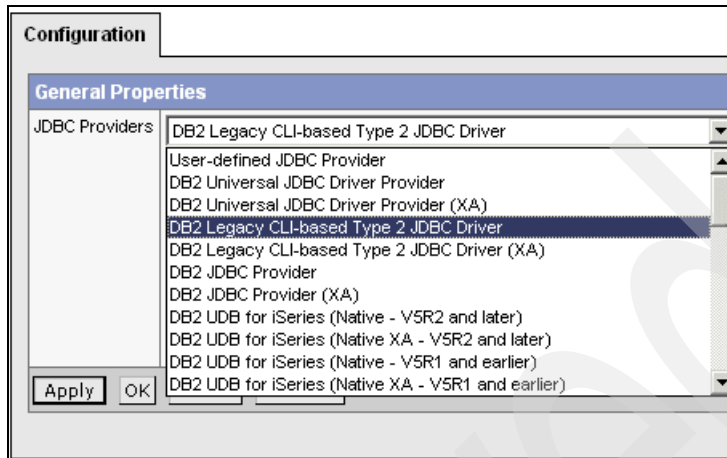


Figure 7-14 Choose a type of JDBC provider to create

5. Enter the location of the DB2 driver in the **Classpath** field. The location of the driver will usually be `/home/db2inst1/sqllib/java12/db2java.zip`.

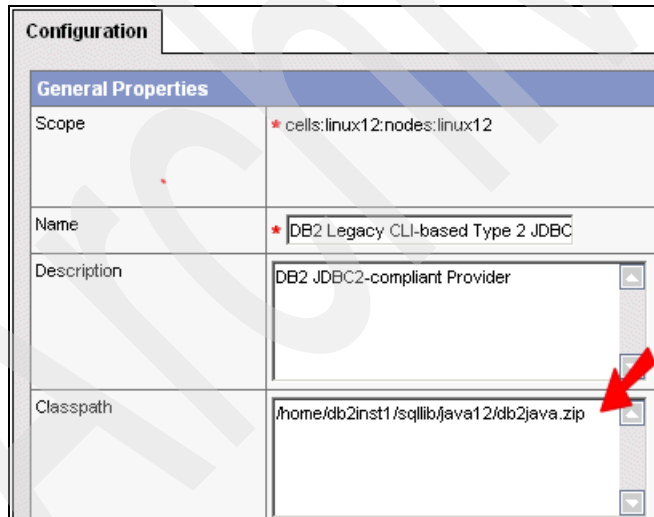


Figure 7-15 Enter the location of the DB2 driver in the Classpath field

6. Go to the bottom of the **Configuration** panel. Under **Additional Properties**, click **Data Sources** and then click **New**.
7. In the new Configuration panel, enter `TraderDB2` in the Name field and `jdbc/TraderDB2` in the JNDI Name field. Make sure that the "Use this Data

Source in container-managed persistence (CMP)" box is checked. Then scroll down to the bottom of the panel and click **Apply**.

General Properties	
Scope	* cells:linux11:nodes:linux11
Name	* TraderDB2
JNDI Name	jdbc/TraderDB2
Container managed persistence	<input checked="" type="checkbox"/> Use this Data Source in container managed persistence (CMP)
Description	Trader datasource

Figure 7-16 Enter the data source name and properties

8. Again scroll down to the bottom of the Configuration panel and click **Custom Properties**. In Custom Properties you need to do three things:
 - a. Change the `databaseName` to `trader`.
 - b. As shown on Figure 7-17 on page 148, add a new property named `user`. The value is the a user name that has access to the DB2 on z/OS. Click **OK** to add the property.
 - c. Also add a new property named `password`. The value is the password of the user in step b. Click **OK** to add the property.

Configuration		
General Properties		
Scope	• cells:linux12:nodes:linux12	The scope of the configured resource. This value indicates the configuration location for the configuration file.
Required	false	
Name	• <input type="text"/>	Name associated with this property (for example, PortNumber and ConnectionURL).
Value	<input type="text"/>	Value associated with this property in this property set.
Description	<input type="text"/>	Text to describe any bounds or well-defined values for this property.
Type	java.lang.String <input type="text"/>	Fully qualified Java type of this property (java.lang.Integer, java.lang.Byte).
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>		

Figure 7-17 Enter new properties for user and password

- When you have added the new properties, click **Save** in the Messages panel. If a Save to Master Configuration panel comes up, click **Save** in that panel, too.

Message(s)
Changes have been made to your local configuration. Click Save to apply changes to the master configuration.
The server may need to be restarted for these changes to take effect.

Figure 7-18 Click Save to apply changes to the master configuration

- Click **Resources** → **JDBC Providers** in the left panel. In the right panel click **DB2 Legacy CLI-based Type 2 JDBC Driver**, which is the JDBC provider that you have just created.
- Under Additional Resources (you may have to scroll down the right panel) click **Data Sources**.
- Put a checkmark at TraderDatasource, as shown in Figure 7-19.

Total: 1

Filter

Preferences

New Delete Test Connection

<input type="checkbox"/>	Name ▾	JNDI Name ▾	Description ▾
<input type="checkbox"/>	TraderDB2	jdbc/TraderDB2	Trader datasource

Figure 7-19 The data source ready to be tested

13. Click **Test Connection** and the following message should appear (Figure 7-20).

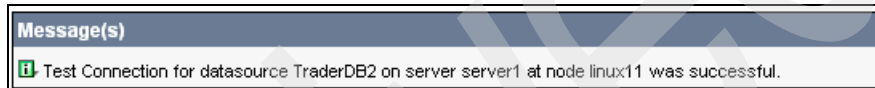


Figure 7-20 The connection for the data source has been verified successfully

7.4 Deploying TraderDB in WebSphere Application Server

In this section we look at the part of Trader that focuses on DB2 connections.

1. Download and unzip the Trader install package into a directory on which you have read/write authority to, for example, /temp.
2. Copy the EAR file to WebSphere Application Server's installable applications:


```
cd /temp
cp TraderDBEAR.ear /opt/WebSphere/AppServer/installableApps/TraderDBEAR.ear
```
3. Open the WebSphere Administrative Console and select **Applications** → **Install New Application**.
4. On the Preparing for the application installation panel select **Server path** and enter /opt/WebSphere/AppServer/installableApps/TraderDBEAR.ear in the field and click **Next**.

Path: Browse the local machine or a remote server:

Local path:

Server path:

Context Root: Used only for standalone Web modules (*.war)

Choose the local path if the ear resides on the same machine as the browser. Choose the server path if the ear resides on any of the nodes in your cell context.

You must specify a context root if the module being installed is a WAR module.

Figure 7-21 Preparing for the application installation

5. On the subsequent panels, click **Next** to accept all defaults. On the last one, click **Finish**.
6. Click **Save** to save the new configuration.
7. Go to **Applications** → **Enterprise Applications** and start the application. Put a checkmark at the TraderDBEAR application and click **Start** (see Figure 7-22).

Start Stop Install Uninstall Update Export Export DDL		
<input type="checkbox"/>	Name	Status
<input type="checkbox"/>	DefaultApplication	
<input type="checkbox"/>	MDBSamples	
<input type="checkbox"/>	PlantsBWWebSphere	
<input type="checkbox"/>	SamplesGallery	
<input type="checkbox"/>	TechnologySamples	
<input type="checkbox"/>	TraderCICSEAR	
<input checked="" type="checkbox"/>	TraderDBEAR	

Figure 7-22 Starting the TraderDBEAR application

8. Enter `http://yourServer/TraderDBWeb/` in a browser to access the TraderDBEAR application, and you will see something like Figure 7-23.



Figure 7-23 The ITSO Trader application using the DB2 connectors

9. Type a user ID, for example, `linux1`, and click **Go**. If you want Trader to use CMP to access the DB2 data instead of hand-coded JDBC, then put a checkmark in the Use CMP box.

Archived

WebSphere MQ connectors

This chapter discusses how to set up the necessary MQ objects needed by the TraderMQ application both on the Linux for zSeries frontend as well as on the z/OS and VSE/ESA backends.

We describe the following topics:

- ▶ Introducing the MQ environment
- ▶ WebSphere MQ setup on Linux for zSeries frontend
- ▶ WebSphere MQ setup on z/OS backend
- ▶ WebSphere MQ setup on VSE backend
- ▶ Configuring VSE for MQ
- ▶ Configuring the MQ connector in WebSphere

8.1 Introducing the MQ environment

Figure 8-1 shows the WebSphere MQ scenario with z/OS.

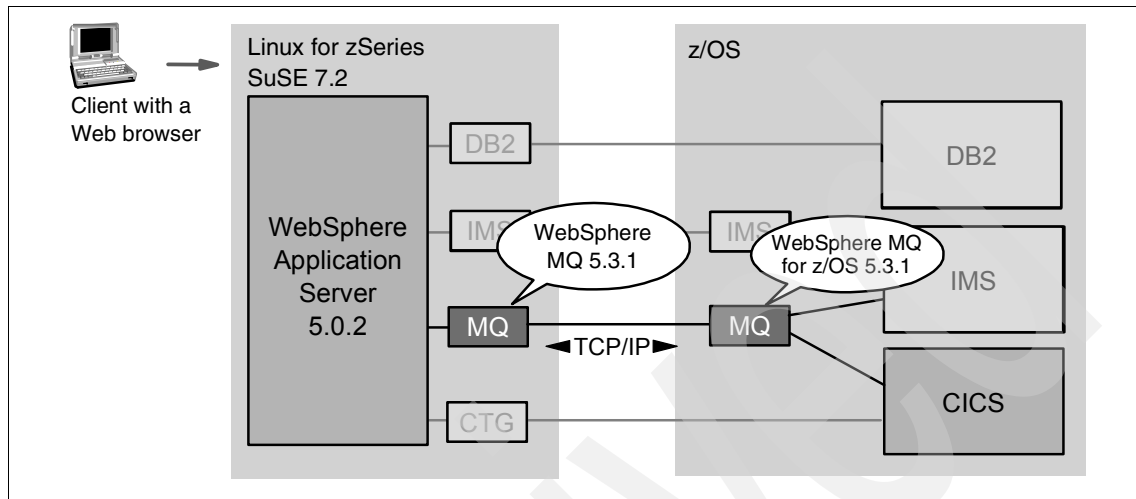


Figure 8-1 WebSphere MQ scenario with z/OS

In this scenario the MQ Queue Manager components are on each side of the TCP/IP connection. On z/OS you can use the MQ-CICS bridge as an interface to the CICS TS application without making any change to these applications.

Figure 8-2 shows the MQ scenario with the VSE backend.

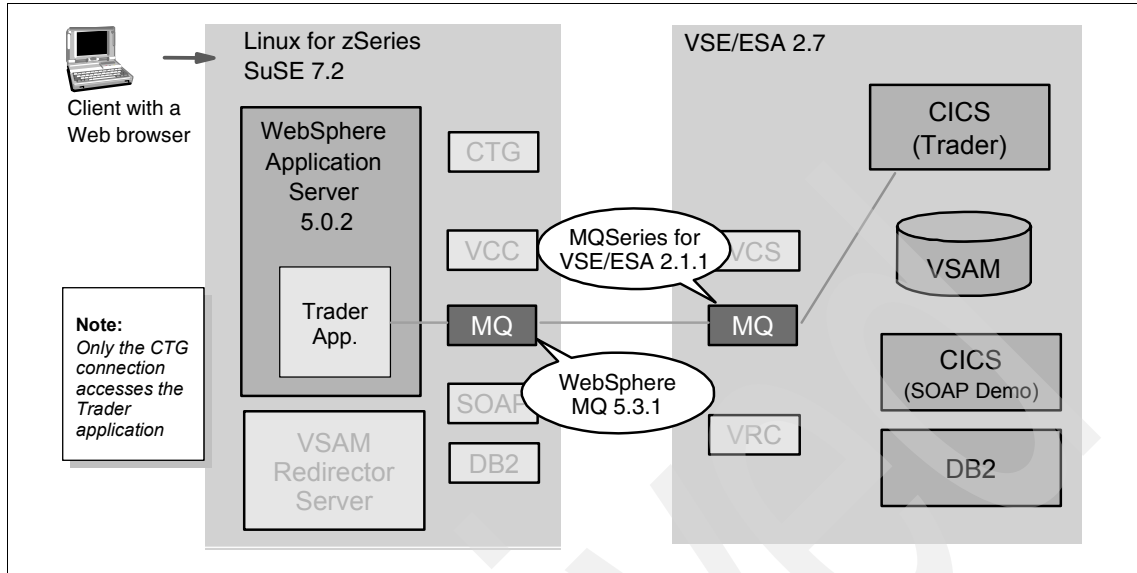


Figure 8-2 WebSphere MQ scenario with VSE/ESA

There is currently no MQ bridge on CICS for VSE. We will, however, show how to set up an MQ connection for VSE. Any installation that wishes to access CICS online transactions using WebSphere MQ must write MQ wrappers for these transactions.

Figure 8-3 shows the MQ setup needed to access the TraderCICS transaction on z/OS via the MQ-CICS bridge. A similar setup is needed to access the TraderIMS transaction.

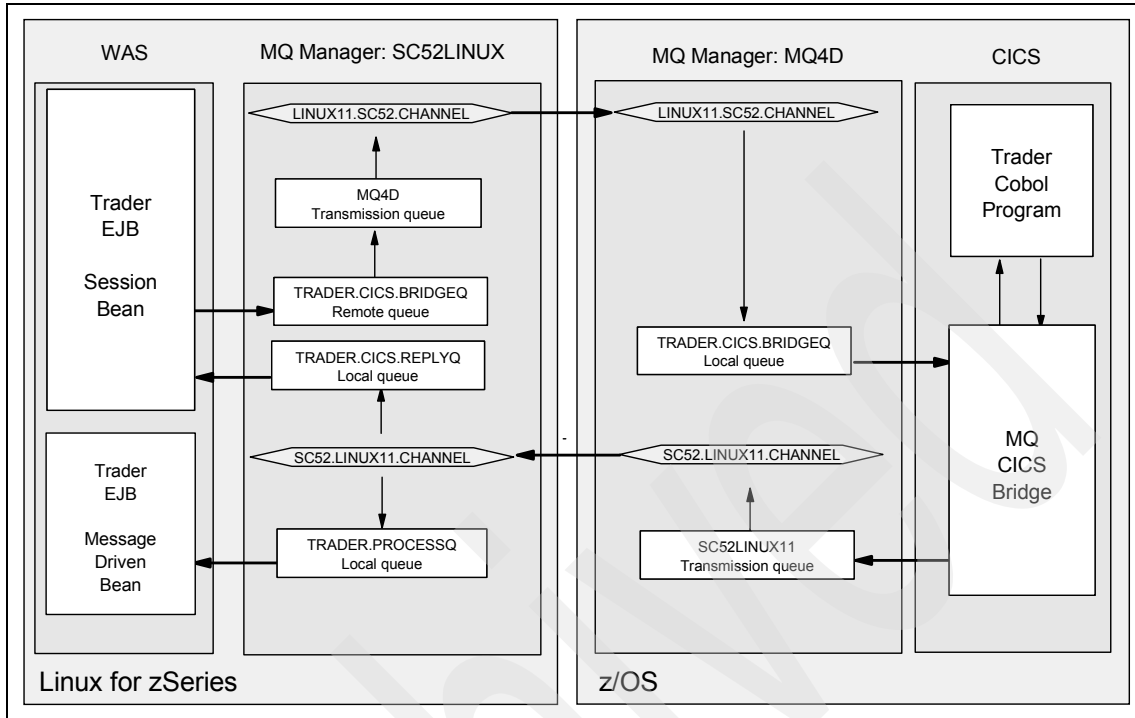


Figure 8-3 MQ setup to access the TraderCICS transaction via the MQ-CICS bridge

8.1.1 MQ-CICS bridge

The MQ-CICS bridge enables an application not running in a CICS environment to run a program or transaction on CICS/ESA and get a response back. You can read more about this feature, as well as download the software and documentation, at:

<http://www6.software.ibm.com/devcon/devcon/docs/male.htm>

8.1.2 MQ-IMS bridge

The bridge is an IMS Open Transaction Manager Access (OTMA) client. In bridge applications there are no MQ calls within the IMS application. The application gets its input using a GET UNIQUE (GU) to the IOPCB and sends its output using an ISRT to the IOPCB. MQ applications use the IMS header (the MQIIH structure) in the message data to ensure that the applications can

execute as they did when driven by nonprogrammable terminals. You can read more about this feature as well as download the support pack at:

<http://www-3.ibm.com/software/integration/support/supportpacs/individual/mal1c.html>

8.2 WebSphere MQ setup on Linux for zSeries frontend

This section describes how to configure WebSphere MQ resources on the frontend Linux on zSeries platform. At this point there are no definitions in WebSphere Application Server involved.

How to find out your default Queue Manager

On a single Linux System you can run more than one Queue Manager. Each Queue Manager will only service its own queues and channels. One of the Queue Managers will be the default manager on the system.

If you do not know the name of your default Queue Manager check out the mqs.ini file in /var/mqm (Example 8-1). This file is the WebSphere MQ Machine-wide Configuration File.

Example 8-1 What is the default Queue Manager in my system

```
linux11:/ # cat /var/mqm/mqs.ini | less
...

AllQueueManagers:
...
    DefaultPrefix=/var/mqm

ClientExitPath:
    ExitsDefaultPath=/var/mqm/exits

LogDefaults:
    LogPrimaryFiles=3
    LogSecondaryFiles=2
    LogFilePages=1024
    LogType=CIRCULAR
    LogBufferPages=0
    LogDefaultPath=/var/mqm/log

QueueManager:
    Name=venus.queue.manager
    Prefix=/var/mqm
    Directory=venus!queue!manager
DefaultQueueManager:
    Name=venus.queue.manager
```

```

QueueManager:
  Name=WAS_linux11_server1
  Prefix=/var/mqm
  Directory=WAS_linux11_server1
QueueManager:
  Name=ITSO_MQ.queue.manager
  Prefix=/var/mqm
  Directory=ITSO_MQ!queue!manager
QueueManager:
  Name=SC52LINUX11
  Prefix=/var/mqm
  Directory=SC52LINUX11
linux11:/

```

How to query all available Queue Managers

The command to use is **dspm**. As we learned, there can be more than one Queue Manager in the system. To find out more about the number, the names, and the status, you can use the **dspm** command (Example 8-2). To do this you need to be logged into the system with root authority.

Example 8-2 Show all Queue Managers in the system

```

linux11:~ # dspm
QMNAME(venus.queue.manager)      STATUS(Ended unexpectedly)
QMNAME(WAS_linux11_server1)      STATUS(Running)
QMNAME(ITSO_MQ.queue.manager)    STATUS(Ended unexpectedly)
QMNAME(SC52LINUX11)              STATUS(Ended unexpectedly)
QMNAME(QM11)                      STATUS(Ended immediately)
QMNAME(MQ11)                      STATUS(Running)
QMNAME(VSELINUX11)               STATUS(Ended unexpectedly)
linux11:~ #

```

How to start the MQ command line interface

The command to use is **runmqsc queue manager name**. In order to use the command line interface you need to log on to the Linux system with root authority.

After you are logged in, first check that the Queue Manager you want to work with is running. To do that use the **dspm** command. If the Queue Manager is not running, execute the **strmqm** command with the name of the Queue Manager you want to work with as a parameter (Example 8-3).

Example 8-3 Starting a Queue Manager

```

linux11:/ # strmqm ITSO_MQ.queue.manager
WebSphere MQ queue manager 'ITSO_MQ.queue.manager' started.

```



```
linux11:/ #
```

Now you are ready to run the **runmqsc** command. As long as you do not want to work with the default Queue Manager, you have to provide the name for the local Queue Manager as a parameter (Example 8-4).

Example 8-4 Start the command-line interface

```
linux11:/ # runmqsc ITSO_MQ.queue.manager
5724-B41 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
Starting MQSC for queue manager ITSO_MQ.queue.manager.
```

Now the MQ command interface is up and running, ready to accept our commands.

Commands available within the MQ command interface

With the **runmqsc** command, use **?**. With the MQ manager started and the command-line interface being active and ready to work, type the following (Example 8-5).

Example 8-5 Query all available MQ definition commands

```
2 : ?
```

```
AMQ8426: Valid MQSC commands are:
```

```
ALTER
CLEAR
DEFINE
DELETE
DISPLAY
END
PING
REFRESH
RESET
RESOLVE
RESUME
START
STOP
SUSPEND
```

A question mark (?) entered at the **runmqsc** prompt prints out all available commands that can be used.

The most important commands will be described in short in the next topics.

How to display MQ resources

With the `runmqsc` command, use *display*. With the MQ manager started and the command-line interface active and ready to work with this Queue Manager, type the following (Example 8-6).

Example 8-6 MQ command interface: Display a resource

```
display
  3 : display
AMQ8405: Syntax error detected at or near end of command segment below:-
display

AMQ8426: Valid MQSC commands are:

DISPLAY AUTHINFO
DISPLAY CHANNEL
DISPLAY CHSTATUS
DISPLAY CLUSQMGR
DISPLAY PROCESS
DISPLAY NAMELIST
DISPLAY QALIAS
DISPLAY QCLUSTER
DISPLAY QLOCAL
DISPLAY QMGR
DISPLAY QMODEL
DISPLAY QREMOTE
DISPLAY QUEUE
DISPLAY QSTATUS
```

Besides the fact that this command on its own is not correct, the command-line interface displays all options of the `display` command. The same happens if parameters or additional options are missing.

You can go ahead and add an option to the `display` command, and as a response you will receive additional information about what is expected next (Example 8-7).

Example 8-7 MQ command interface: Display a resource

```
display channel
  4 : display channel
AMQ8405: Syntax error detected at or near end of command segment below:-
display channel

AMQ8427: Valid syntax for the MQSC command:

DISPLAY CHANNEL(generic_channel_name)
  [ TYPE( ALL | SDR | SVR | RCVR | RQSTR | CLNTCONN | SVRCONN ) ]
```

```

[ ALL      ] [ BATCHHB ] [ BATCHINT ] [ BATCHSZ ] [ CHLTYPE ]
[ CONNAME ] [ CONVERT ] [ DESCR    ] [ DISCINT ] [ HBINT    ]
[ LONGRTY ] [ LONGTMR ] [ MAXMSGL  ] [ MCANAME ] [ MCATYPE  ]
[ MCAUSER ] [ MODENAME ] [ MRDATA   ] [ MREXIT  ] [ MRRTY    ]
[ MRTMR   ] [ MSGDATA ] [ MSGEXIT  ] [ NPMSPEED ] [ PASSWORD ]
[ PUTAUT  ] [ QMNAME  ] [ RCVDATA  ] [ RCVEXIT  ] [ SCYDATA  ]
[ SCYEXIT ] [ SENDDATA ] [ SENDEXIT ] [ SEQWRAP  ] [ SHORTTRY ]
[ SHORTTMR ] [ SSLCAUTH ] [ SSLCIPH  ] [ SSLPEER  ] [ TPNAME   ]
[ TRPTYPE ] [ TYPE     ] [ USERID   ] [ XMITQ    ] [ LOCLADDR ]
[ KAINTE  ]

```

How to define MQ resources

With the `runmqsc` command, use *define*. With the MQ manager started and the command-line interface active and ready to work with this Queue Manager, type the following (Example 8-8).

Example 8-8 MQ command interface: Define a resource

```

5 : define qlocal
AMQ8405: Syntax error detected at or near end of command segment below:-
define qlocal

AMQ8427: Valid syntax for the MQSC command:

DEFINE QLOCAL(q_name)
  [ BOQNAME(string) ] [ BOTHRESH(integer) ]
  [ CLUSNL(name_list_name) ] [ CLUSTER(cluster_name) ]
  [ DEFBIND( NOTFIXED | OPEN ) ] [ DEFPRTY(integer) ]
  [ DEFPSIST( NO | YES ) ] [ DESCR(string) ]
  [ DEFSOPT( EXCL | SHARED ) ] [ DISTL( NO | YES ) ]
  [ GET( ENABLED | DISABLED ) ] [ INITQ(string) ]
  [ LIKE(qlocal_name) ] [ MAXDEPTH(integer) ]
  [ MAXMSGL(integer) ] [ MSGDLVSQ( PRIORITY | FIFO ) ]
  [ NOHARDENBO | HARDENBO ] [ NOREPLACE | REPLACE ]
  [ NOSHARE | SHARE ] [ NOTRIGGER | TRIGGER ]
  [ PROCESS(string) ] [ PUT( ENABLED | DISABLED ) ]
  [ QDEPTHHI(integer) ] [ QDEPTHLO(integer) ]
  [ QDPHIEV( ENABLED | DISABLED ) ] [ QDPLOEV( ENABLED | DISABLED ) ]
  [ QDPMAXEV( ENABLED | DISABLED ) ] [ QSVCIEV( NONE | HIGH | OK ) ]
  [ QSVCIINT(integer) ] [ RETINTVL(integer) ]
  [ SCOPE( QMGR | CELL ) ] [ TRIGDATA(string) ]
  [ TRIGDPTH(integer) ] [ TRIGMPRI(integer) ]
  [ TRIGTYPE( FIRST | EVERY | DEPTH |
NONE ) ] [ USAGE( NORMAL | XMITQ ) ]

```

Again, the `define` command by itself is not sufficient, but it gives an overview of the resources that you can define.

8.3 WebSphere MQ setup on z/OS backend

There is an ISPF panel-driven interface to WebSphere MQ for z/OS. The tool is helpful and easy to use, create, and maintain MQ objects.

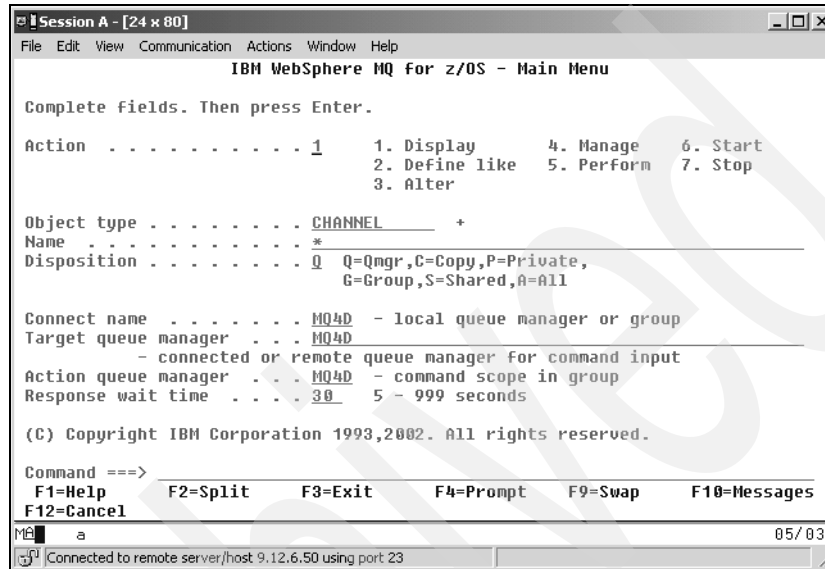


Figure 8-4 The IBM WebSphere MQ for z/OS - Main Menu panel

However, in the process of writing this book, we used the batch API, which we find more suitable to create multiple MQ objects at one time.

8.3.1 Configuring queues and channels on z/OS

We set up an MQ manager on z/OS called MQ4D. This name, of course, may be different at your installation. It will, however, be used throughout this book.

The JCL you need to run in order to create the MQ4D MQ manager and Trader's MQ objects is listed in Figure 8-9.

Example 8-9 JCL for configuring WebSphere MQ on z/OS

```
//COMMAND EXEC PGM=CSQUTIL,PARM=MQ4D
//STEPLIB DD DSN=MQ531.SCSQANLE,DISP=SHR
//CSQUCMD DD DSN=MQ4D.INSTALL.JCL(MQ4DSCXC),DISP=SHR
// DD DSN=MQ4D.INSTALL.JCL(MQ4DCBRG),DISP=SHR
// DD DSN=MQ4D.INSTALL.JCL(MQ4DLINZ),DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
```

* NEXT STATEMENT CAUSES COMMANDS TO BE READ FROM CSQUCMD DDNAME
 COMMAND
 /*

The two script files, MQ4D.INSTALL.JCL (MQ4DCBRG) and MQ4D.INSTALL.JCL (MQ4DSCXC), contain statements to create the following MQ objects (Table 8-1).

Table 8-1 MQ objects

Object type	Target	Object name	Example
Queue	CICS	TRADER.CICS.BRIDGEQ	Example 8-10
Process	CICS	SRTCKBR	Example 8-10
Queue	IMS	TRADER.IMS.BRIDGEQ	Example 8-11
Storage class	IMS	IMSBRG	Example 8-11
Xmit queue	Both	SC52LINUX11	Example 8-12
Receiver channel	Both	LINUX11.SC52.CHANNEL	Example 8-12
Sender channel	Both	SC52.LINUX11.CHANNEL	Example 8-12

If you plan to run a Trader application that does not use WebSphere MQ, such as TraderDB or TraderCICS, you may leave out the MQ definitions because they are not used by programs that do not communicate via MQ.

The MQ4DCBRG member contains the definitions needed for the MQ-CICS bridge to work (Example 8-10).

Example 8-10 MQ4DCBRG: Definitions to handle calls to CICS Trader application via MQ-CICS bridge

```
*****
*                               IBM MQSeries for OS/390
*   Sample for Queue
*   definitions for the CICS Bridge
*****
*   This sample data set contains a set of definitions for
*   the following objects for general use
*   that you can customize as required:
*   - queues
*   - process
*****
* RECOMMENDED NON-SYSTEM OBJECTS
*   CICS Bridge Request Queues for incoming CICS transactions
*   CICS Bridge Reply Queues for returning CICS data
*   Process STRTCKBR to start CICS Bridge Monitor CKBR
*****
```

```

DEFINE QLOCAL('TRADER.CICS.BRIDGEQ') +
    REPLACE +
    QSGDISP(QMGR) +
    DESCR('CICS Bridge request queue') +
    PUT(ENABLED) +
    DEFPRTY(0) +
    DEFPSIST(NO) +
    CLUSTER(' ') CLUSNL(' ') DEFBIND(OPEN) +
    GET(ENABLED) +
    SHARE +
    DEFSOPT(SHARE) +
    MSGDLVSQ(FIFO) +
    RETINTVL(999999999) +
    MAXDEPTH(999 ) +
    MAXMSGL(4194304) +
    NOHARDENBO +
    BOTHRESH(0) +
    BOQNAME(' ') +
    STGCLASS('SYSVOLAT') +
    USAGE(NORMAL) +
    INDXTYPE(NONE) +
    CFSTRUCT(' ')
    QDPMAEV(ENABLED) +
    QDPHIEV(DISABLED) +
    QDEPTHHI(80) +
    QDPLOEV(DISABLED) +
    QDEPTHLO(40) +
    QSVCIIEV(NONE) +
    QSVCIINT(999999999) +
    NOTRIGGER +
    TRIGTYPE(FIRST) +
    TRIGMPRI(0) +
    TRIGDPTH(1) +
    TRIGDATA(' ') +
    PROCESS('STRTCKBR') +
    INITQ('CICS01.INITQ')

DEFINE PROCESS('STRTCKBR') +
    REPLACE +
    QSGDISP(QMGR) +
    APPLTYPE(CICS) +
    APPLICID('CKBR') +
    USERDATA('WAIT=600, AUTH=IDENTIFY ')

*****
* End of MQ4DCBRG
*****

```

Example 8-11 MQ4DSCXC: Definitions to handle calls to Trader IMS transaction via MQ-IMS bridge

```

*****
*                               IBM MQSeries for OS/390

```

```

* Sample for Storage Class definitions using XCF for the IMS Bridge
*****
* This sample data set contains a set of definitions for
* the following objects for general use
* that you can customize as required:
*   - storage classes
*   - queues
*****
* STORAGE CLASSES
*****
* These storage class definitions are needed to match the IMS
* OTMA GRNAME (XCF group) and USERVAR/APPLID1 (XCF member)
* You are recommended not to define any storage classes to map
* to page set 00 where object definitions are kept in order to keep
* messages separate from them.
* Further storage class definitions should
* be added to this sample as required.
*****

*DELETE STGCLASS('IMSBRG')

DEFINE STGCLASS('IMSBRG') +
    QSGDISP(QMGR) +
    PSID(03) +
    XCFGNAME('HAOTMA') +
    XCFMNAME('SCSIMS4D')

*
*****
* RECOMMENDED NON-SYSTEM OBJECTS
*   IMS Bridge Request Queues for incoming IMS transactions
*   IMS Bridge Reply Queues for returning IMS data
*****
DEFINE QLOCAL('TRADER.IMS.BRIDGEQ') +
    REPLACE +
    QSGDISP(QMGR) +
    DESCR('IMS Bridge request queue') +
    PUT(ENABLED) +
    DEFPRTY(0) +
    DEFPSIST(NO) +
    CLUSTER(' ') CLUSNL(' ') DEFBIND(OPEN) +
    GET(ENABLED) +
    SHARE +
    DEFSOPT(SHARE) +
    MSGDLVSQ(FIFO) +
    RETINTVL(999999999) +
    MAXDEPTH(999) +
    MAXMSGL(4194304) +
    NOHARDENBO +
    BOTHRESH(0) +
    BOQNAME(' ') +
    STGCLASS('IMSBRG') +
    USAGE(NORMAL) +
    INDXTYPE(NONE) +

```

```

CFSTRUCT(' ') +
QDPMAXEV(ENABLED) +
QDPHIEV(DISABLED) +
QDEPTHHI(80) +
QDPLOEV(DISABLED) +
QDEPTHLO(40) +
QSVCIIEV(NONE) +
QSVCIINT(999999999) +
NOTRIGGER +
TRIGTYPE(NONE) +
TRIGMPRI(0) +
TRIGDPTH(1) +
TRIGDATA(' ') +
PROCESS(' ') +
INITQ(' ')

```

```

*****
* End of MQ4DSCXC
*****

```

The MQ4DLINZ contains the definitions on z/OS needed to handle the communications with the MQ manager on Linux for zSeries (Example 8-12).

Example 8-12 MQ4DLINZ: Definitions on z/OS to handle communications with the MQ manager on Linux for zSeries

```

*****
* Channels & xmit queue necessary to connect
* to MQ mgr. on Linux for zSeries
*****

*DELETE CHL(LINUX11.SC52.CHANNEL)
*DELETE CHL(SC52.LINUX11.CHANNEL)
*DELETE QLOCAL(SC52LINUX11)

DEF CHL(LINUX11.SC52.CHANNEL) +
  CHLTYPE(RCVR) +
  TRPTYPE(TCP)

DEF CHL(SC52.LINUX11.CHANNEL) +
  CHLTYPE(SDR) +
  TRPTYPE(TCP) +
  CONNAME('9.12.9.12(1414)') +
  CONVERT(YES) +
  XMITQ(SC52LINUX11)

DEF QLOCAL(SC52LINUX11) +
  USAGE(XMITQ)
*****
* End of MQ4DLINZ
*****

```

8.3.2 Configuring queues and channels on Linux for zSeries

On Linux we create an MQ manager called SC52LINUX11 and we set up queues and channels as follows:

1. Create and start the MQ manager:

```
/opt/mqm/bin/crtmqm SC52LINUX11  
/opt/mqm/bin/strmqm SC52LINUX11
```

2. Start the MQ command-line interface:

```
runmqsc SC52LINUX11
```

3. Enter the channel and queue definitions as listed in Example 8-13.

Example 8-13 Channel and queue definitions on Linux

```
DEF CHL(LINUX11.SC52.CHANNEL) +  
  CHLTYPE(SDR) +  
  TRPTYPE(TCP) +  
  CONNAME('wtsc52.itso.ibm.com(1562)') +  
  CONVERT(YES) +  
  XMITQ(MQ4D)  
  
DEF CHL(SC52.LINUX11.CHANNEL) CHLTYPE(RCVR) TRPTYPE(TCP)  
  
DEF QLOCAL(TRADER.CICS.REPLYQ)  
  
DEF QLOCAL(TRADER.IMS.REPLYQ)  
  
DEF QLOCAL(TRADER.PROCESSQ)  
  
DEF QLOCAL(MQ4D) USAGE(XMITQ)  
  
DEF QREMOTE(TRADER.CICS.BRIDGEQ) +  
  RNAME(TRADER.CICS.BRIDGEQ) +  
  RQMNAME(MQ4D) +  
  XMITQ(MQ4D)  
  
DEF QREMOTE(TRADER.IMS.BRIDGEQ) +  
  RNAME(TRADER.IMS.BRIDGEQ) +  
  RQMNAME(MQ4D) +  
  XMITQ(MQ4D)
```

When defining the sender channel, the name of the remote MQ manager and the IP address of the remote z/OS server should be changed from those of the example to reflect the settings of your own shop.

4. Start the MQ manager's TCP/IP listener and the sender channel:

```
runmq1sr -t TCP -m SC52LINUX11 &
```

```
runmqchl -c SC52.LINUX11.CHANNEL -m SC52LINUX11 &
```

The MQ manager is now ready to serve the Trader application.

8.4 WebSphere MQ setup on VSE backend

The MQ part of the Trader application cannot be used with VSE as a backend because there is no MQ Bridge on VSE. So, this section just shows the WebSphere setup for a simple test servlet.

8.4.1 Configuring queues and channels on Linux for zSeries

It is assumed that MQ for Linux on zSeries is already installed on the WebSphere Application Server system.

MQ for Linux on zSeries does not have a graphical user interface as it has on Windows. Due to that fact all MQ-related definitions like defining queues and channels have to be done through the command-line interface.

To give you a good understanding on how things relate to each other, we like to give you a short overview about the most common commands used to administer MQ for Linux on zSeries.

Attention: MQ Series treats some of its definitions in a case-sensitive way.

If you experience any problem make sure you typed the parameters the same way as they are displayed.

8.4.2 Defining MQ resources to Linux for zSeries

First, we create a new Queue Manager. With Linux you can have as many Queue Managers as you want running at the same time. That makes it easier to try new things without disturbing others.

Telnet into Linux with root permission. Type:

```
crtmqm MQ11
```

A new Queue Manager with the name MQ11 is created with a name exactly the way you typed it. Keep that in mind. This Queue Manager already has the default resources defined.

The new Queue Manager is ready for use immediately. Start the Queue Manager by typing in the command `strmqm` with the Queue Manager's name:

```
strmqm MQ11
```

Important: At this point, our recommendation is to use upper-case definitions on both sides for the Queue Manager's name.

Check how many characters of a definition are supported by each side.

If a mixed-case definition is used for addressing or to be addressed, it is very likely that it will end up in a failure.

Trying to start MQ11 with `strmqm mq11` will show the case sensitivity of the Queue Manager's name. Now think about the situation where your VSE has to provide the name of the remote Queue Manager. It will simply not work (Example 8-14).

Example 8-14 Small upper/lower-case test

```
linux11:/ # strmqm mq11
AMQ8118: WebSphere MQ queue manager does not exist.
linux11:/ #
```

8.4.3 Shell script to define the Linux-VSE connection

The next definition steps are placed into a shell script. These are all the statements needed to define a connection to VSE (Example 8-15).

Example 8-15 Definition script was_vse270.sh

```
# ! /bin/sh
runmqsc MQ11 << EOF
    delete qlocal (was.tx.vse270) purge
    delete qlocal (vse270.was)
    delete channel (was.vse270)
    delete channel (vse270.was)
    delete qremote (was.vse270)
    define qlocal (vse270.was)
    define qlocal (was.tx.vse270) usage (xmitq)
    define qremote (was.vse270) rname (WAS.VSE270) rqmname ('QMVSE27')
xmitq (was.tx.vse270)
    define channel (was.vse270) chltype (sdr) conname
('9.156.175.132(1414)') convert (yes) xmitq (was.tx.vse270) trptype (tcp)
seqwrap (999999)
    define channel (vse270.was) chltype (rcvr) trptype (tcp) seqwrap
(999999)
end
```

Let us have a closer look at the contents of Example 8-15.

The first line is only a comment telling that it is a shell script. The second line invokes the command-line processor for the MQ11 Queue Manager, and opens a pipe to pass the definition statement for input until the string 'EOF' is discovered.

Now:

```
delete qlocal (was.tx.vse270) purge
delete qlocal (vse270.was)
delete channel (was.vse270)
delete channel (vse270.was)
delete qremote (was.vse270)
```

In case we made changes on the define statements it is best to delete what we defined before. You can also use the ALTER command for changing properties of a definition, but we prefer the general cleanup.

Defining the local queue

Defining local queues can involve two dozen options, most of them having a default (Example 8-16).

Example 8-16 Complete syntax of the define qlocal command

```
DEFINE QLOCAL(q_name)
  [ BOQNAME(string) ]
  [ CLUSNL(name1ist_name) ]
  [ DEFBIND( NOTFIXED | OPEN ) ]
  [ DEFPSIST( NO | YES ) ]
  [ DEFSOPT( EXCL | SHARED ) ]
  [ GET( ENABLED | DISABLED ) ]
  [ LIKE(qlocal_name) ]
  [ MAXMSGL(integer) ]
  [ NOHARDENBO | HARDENBO ]
  [ NOSHARE | SHARE ]
  [ PROCESS(string) ]
  [ QDEPTHHI(integer) ]
  [ QDPHIEV( ENABLED | DISABLED ) ]
  [ QDPMAXEV( ENABLED | DISABLED ) ]
  [ QSVCIINT(integer) ]
  [ SCOPE( QMGR | CELL ) ]
  [ TRIGDPTH(integer) ]
  [ BOTHRESH(integer) ]
  [ CLUSTER(cluster_name) ]
  [ DEFPRTY(integer) ]
  [ DESCR(string) ]
  [ DISTL( NO | YES ) ]
  [ INITQ(string) ]
  [ MAXDEPTH(integer) ]
  [ MSGDLVSQ( PRIORITY | FIFO ) ]
  [ NOREPLACE | REPLACE ]
  [ NOTRIGGER | TRIGGER ]
  [ PUT( ENABLED | DISABLED ) ]
  [ QDEPTHLO(integer) ]
  [ QDPLOEV( ENABLED | DISABLED ) ]
  [ QSVCIIEV( NONE | HIGH | OK ) ]
  [ RETINTVL(integer) ]
  [ TRIGDATA(string) ]
  [ TRIGMPRI(integer) ]
  [ TRIGTYPE( FIRST | EVERY | DEPTH |
  [ USAGE( NORMAL | XMITQ ) ]
  NONE ) ]
```

```
define qlocal (vse270.was)
```

This statement creates a local queue to which other Queue Managers can send messages. Even if the queue is 'vse270.was', this name is not treated as case sensitive, as it is with the Queue Manager's name.

A display of the queue definition shows the following properties (Example 8-17).

Example 8-17 Detailed view of the queue properties

```
display qlocal (vse270.was) all
  27 : display qlocal (vse270.was) all
AMQ8409: Display Queue details.
  DESCR(WebSphere MQ Default Local Queue)
  PROCESS( )
  INITQ( )
  CLUSTER( )
  QUEUE(VSE270.WAS)
  CRTIME(08.06.56)
  ALTTIME(08.06.56)
  PUT(ENABLED)
  DEFPSIST(NO)
  MAXMSGL(4194304)
  SHARE
  HARDENBO
  RETINTVL(999999999)
  NOTRIGGER
  TRIGDPTH(1)
  QDEPTHHI(80)
  QDPMAXEV(ENABLED)
  QDPLOEV(DISABLED)
  QSVCI EV(NONE)
  DEFTYPE(PREDEFINED)
  SCOPE(QMGR)
  IPPROCS(0)
  CURDEPTH(0)
  BOQNAME( )
  TRIGDATA( )
  CLUSNL( )
  CRDATE(2003-11-18)
  ALTDATE(2003-11-18)
  GET(ENABLED)
  DEFPRTY(0)
  MAXDEPTH(5000)
  BOTHRESH(0)
  DEFSOPT(SHARED)
  MSGDLVSQ(PRIORITY)
  USAGE(NORMAL)
  TRIGTYPE(FIRST)
  TRIGMPRI(0)
  QDEPTHLO(20)
  QDPHIEV(DISABLED)
  QSVCINT(999999999)
  DISTL(NO)
  TYPE(QLOCAL)
  DEFBIND(OPEN)
  OPPOCS(0)
```

Defining the transmit queue

Key in the following:

```
define qlocal (was.tx.vse270) usage (xmitq)
```

Now the local transmit queue (was.tx.vse270) is created. This queue serves the purpose of storing message put into a local queue until the channel manager for this system can deliver them.

qlocal (was.tx.vse270) The name for local transmit queue.

usage (xmitq) This attribute turns a local queue into a transmit queue.

Defining the remote queue

Example 8-18 shows the syntax of the **define qremote** command.

Example 8-18 Complete syntax of the define qremote command

```
DEFINE QREMOTE(q_name)
  [ CLUSNL(name1ist_name) ]           [ CLUSTER(cluster_name) ]
  [ DEFBIND( NOTFIXED | OPEN ) ]     [ DEFPRTY(integer) ]
  [ DEFPSIST( NO | YES ) ]           [ DESCR(string) ]
  [ LIKE(qremote_name) ]             [ PUT( ENABLED | DISABLED ) ]
  [ NOREPLACE | REPLACE ]           [ RNAME(string) ]
  [ RQMNAME(string) ]               [ SCOPE( QMGR | CELL ) ]
  [ XMITQ(string) ]
```

```
define qremote (was.vse270) rname (WAS.VSE270) rqmname ('QMVSE27') xmitq
(was.tx.vse270)
```

Here, we build the first relation between our system and the remote system. First, we declare the remote queue name `qremote (was.vse270)` as a local queue. The definition `rname (WAS.VSE270)` is the queue local to remote MQ that will receive our messages. Do not get confused because the names are both the same. The name of the local queue pointing to the remote queue can be any name you like. The next important information is the name of the remote Queue Manager. Here again this name is case sensitive. Because the system we connect to is a VSE system, the name must be in uppercase, otherwise the VSE MQ manager will not respond. Last, we point to the transmit queue `xmitq (was.tx.vse270)`. This is the linking chain between the local queue `was.vse270` and the local sender channel.

qremote (was.vse270)	Local name for remote queue
rname (WAS.VSE270)	Real name of remote queue
rqmname ('QMVSE27')	Real name of remote QM
xmitq (was.tx.vse270)	Local name of xmit queue

Defining the sender channel

Example 8-19 shows the syntax of the **define channel chtype (sdr)** command.

Example 8-19 Complete syntax of the define channel chtype (sdr) command

```
DEFINE CHANNEL(channel_name)
  CHLTYPE( SDR )                     CONNAME(string)
  XMITQ(string)                      [ BATCHHB(integer) ]
  [ BATCHINT(integer) ]              [ BATCHSZ(integer) ]
```

[CONVERT(NO YES)]	[DESCR(string)]
[DISCINT(integer)]	[HBINT(integer)]
[KAINTE(integer)]	[LIKE(channel_name)]
[LONGRTY(integer)]	[LONGTMR(integer)]
[MAXMSGL(integer)]	[MCANAME(string)]
[MCATYPE(PROCESS THREAD)]	[MCAUSER(string)]
[MODENAME(string)]	[MSGDATA(string)]
[MSGEXIT(string)]	[NOREPLACE REPLACE]
[NPMSPEED(NORMAL FAST)]	[PASSWORD(string)]
[RCVDATA(string)]	[RCVEXIT(string)]
[SCYDATA(string)]	[SCYEXIT(string)]
[SENDDATA(string)]	[SENDEXIT(string)]
[SEQWRAP(integer)]	[SHORTRTY(integer)]
[SHORTTMR(integer)]	[SSLCIPH(string)]
[SSLPEER(string)]	[TPNAME(string)]
[TRPTYPE(LU62 TCP)]	[USERID(string)]
[LOCLADDR(string)]	

```
define channel (was.vse270) chltype (sdr) conname ('9.156.175.132(1414)')
convert (yes) xmitq (was.tx.vse270) trptype (tcp) seqwrap (999999)
```

Now we set up the sender path channel (was.vse270). This channel is monitoring the transmit queue, and if the other side of the connection is available, it delivers what is in the transmit queue. The destination is describe by the host name or IP address. In addition, if we decide not to use port 1414, which is the default, we can provide a different one by putting the port number into parenthesis next to the host definition conname ('9.156.175.132(1414)'). The next important thing is to make a character conversion from ASCII format to the EBCDIC format defining convert (yes), as follows:

channel (was.vse270)	Name of the sender channel.
chltype (sdr)	This is a sender channel.
conname ('9.156.175.132(1414)')	Our target host and port.
convert (yes)	Convert ASCII to EBCDIC.
xmitq (was.tx.vse270)	Monitor this xmit queue.
trptype (tcp)	Use tcp/ip.
seqwrap (999999)	Set the sequence number to what VSE can support.

Defining the receiver channel

Key in the following:

```
define channel (vse270.was) chltype (rcvr) trptype (tcp) seqwrap (999999)
```

The receiver channel is the network interface for the local queue to which a remote system will send its messages.

Important: This definition will not start the active part of the MQ listener.

You have to start the listener with the following command:

```
runmq1sr [-m QMgrName] -t TCP [ProtocolOptions] -p port
```

To stop the listener use the following command:

```
endmq1sr [-m QMgrName]
```

The receiver channel must reflect the definition of the sender side. If the channel is supposed to receive messages from a VSE system there are restrictions you have to take note of. First, do not run the receiver queue with a name containing lower-case characters. Second, the sequence wrap counter has a maximum of 999999; the default on MQ linux is 999 999 999. The attempt to connect from VSE to a MQ Linux channel defined with the default will fail.

channel (vse270.was)	Name of the receiver channel.
chltype (rcvr)	This is a receiver channel.
trptype (tcp)	Use tcp/ip.
seqwrap (999999)	Set the sequence number to what VSE can support.

Starting the listener for the receiver channel

Now, everything to connect MQ VSE and Linux is definend. In order to test the connection, we have to start our listener as well as our sender channel. The MQ manager must be active; otherwise you have to start the task (Example 8-20).

Example 8-20 Start the MQ manager MQ11

```
linux11:/ # dspmq
QMNAME (MQ11)                                STATUS (Ended normally)
linux11:/ #
linux11:/ #
linux11:/ #
linux11:/ # strmqm MQ11
WebSphere MQ queue manager 'MQ11' started.
linux11:/ #
```

After the Queue Manager is up and running, start the listener as shown in Example 8-21.

Example 8-21 Start the listener for MQ11 with port 1414

```
linux11:/ #
linux11:/ # runmq1sr -m MQ11 -t tcp -p 1414
```


Now we can continue with the VSE/ESA side of the MQ connection.

8.5 Configuring VSE for MQ

MQ Series for VSE/ESA runs under control of VSE CICS/ESA 2.x as well as CICS TS 1.1. There is a batch interface available to support MQ Batch applications as well as MQ online applications.

Configuring MQ must be a straightforward approach. This means that you have to decide on a couple of naming definitions that will have an impact on other MQ systems:

- ▶ The name of the sender channel must match the name of the receiver channel on the remote end of your MQ connection.
- ▶ The remote queue name on the sender side must match the local queue name connected to the receiver channel on the receiving MQ system.
- ▶ The definition for the remote queue name must reference the same transmit queue name as the definition for the sender channel.

Figure 8-5 summarizes these naming conventions.

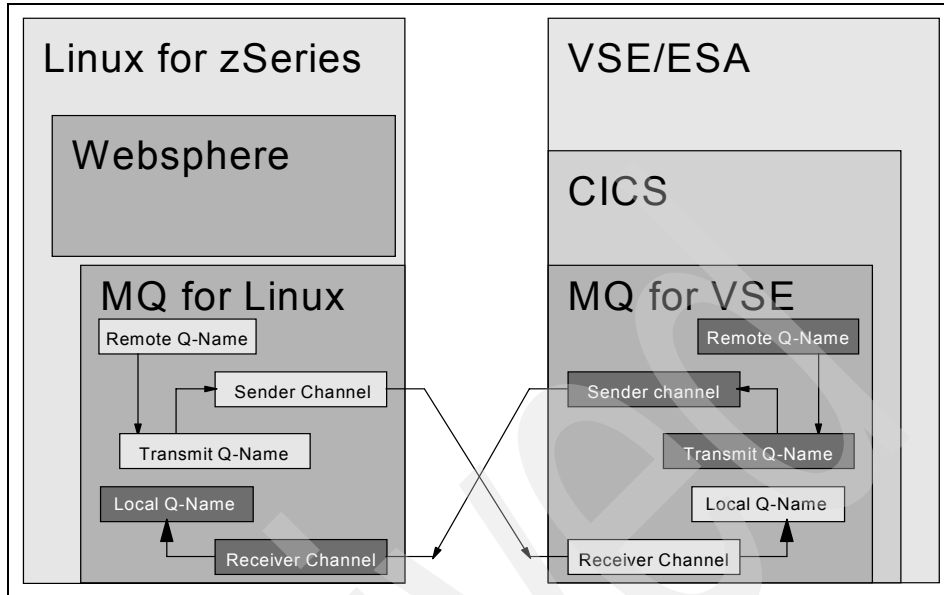


Figure 8-5 MQ naming conventions

MQ Series for VSE uses VSAM data files to store messages to send or to receive. These files are assigned to the local queues that will receive messages from other hosts addressing the target using the name for these queues. MQ messages targeting another MQ system are kept till they are delivered in so-called transmit queues, which again are represented by local VSAM data files.

If you plan to connect from Websphere to VSE using MQ you have to set up the following resources first.

- ▶ VSAM CLUSTER for the LOCAL QUEUE
- ▶ VSAM CLUSTER for the TRANSMIT QUEUE
- ▶ CICS RDO DEFINITIONS for the VSAM CLUSTER
- ▶ MQ SENDER / RECEIVER CHANNEL DEFINITIONS
- ▶ MQ LOCAL QUEUE DEFINITIONS
- ▶ MQ REMOTE QUEUE DEFINITIONS

8.5.1 Defining the VSAM data files

We recommend that you use the Interactive User Interface (IUI). The IUI will also take care of defining the necessary DLBL statements by adding them to the label procedure STDLABUP.PROC in IJSYSRS.SYSLIB.

You now must decide in which of your system VSAM catalogs you want the MQ VSAM data files to reside. A message queue file is required for each queue defined to the MQ (CICS) subsystem. You need one file for the local queue to receive messages and one data file for the queue used for transmitting messages to other systems.

Depending on the size and amount of messages that will be exchanged between the systems you have, you have to make decisions about the following:

- ▶ USER CATALOG
- ▶ USER CATALOG LOCATION
- ▶ PRIMARY ALLOCATION
- ▶ SECONDARY ALLOCATION

Member MQJQUEUE.Z in library PRD2.PROD is the installation job stream that defines the default MQ VSAM data files. Use the values given there to define your own data sets. Provide your own definitions only where you see question marks. Never change any of the other attributes.

You are strongly recommended to define one local queue in each physical file. If you intend to use the automatic VSAM reorganization feature with a queue, that queue must be the only queue in a physical VSAM file.

Tip: Please read the MQ for VSE/ESA System Management Guide Version 2 Release 1 Modification 2 (GC34-5364-03).

After having done this go ahead and define the files using the IUI dialogs. For using the menu item Resource Definition from the entry panel you must have proper privileges.

FAST PATH 22

Start from the entry panel of the IUI (see Example 8-22, IUI Entry Panel), and type 22 at the prompt on the bottom line.

Example 8-22 IUI entry panel

```
IESADMSL.SYSA                VSE/ESA FUNCTION SELECTION                APPLID: A0006CI1
```

Enter the number of your selection and press the ENTER key:

- 1 Installation
- 2 Resource Definition
- 3 Operations
- 4 Problem Handling
- 5 Program Development
- 6 Command Mode


```

ALLOCATION UNIT..... 1          1=Cylinder, 2=Track
PRIMARY ALLOCATION..... 5
SECONDARY ALLOCATION..... 3

CONTROL INTERVAL SIZE.... 4096
AVERAGE RECORD SIZE..... 200
MAXIMUM RECORD SIZE..... 4089

ENABLE DATA COMPRESSION.. 2          1=Yes, 2=No

KEY LENGTH..... 56_
KEY POSITION..... 0          Position 0 starts at the beginning

PF1=HELP      2=REDISPLAY  3=END      4=RETURN

```

Example 8-25 Defining the MQ receiver queue file (part 3)

```

IESFILJOBX          JOB EXECUTION

The control statements for the specified function can be executed
immediately or delayed for later submission.

JOB EXECUTION..... 1  1=Delayed, Submission is handled by user
                     2=Immediate, Job is executed

For delayed execution and job submission, the control statements
are stored in the following ICCF library member.

LIBRARY MEMBER.... vserxq          Existing member is overwritten

PF1=HELP          3=END          4=RETURN

```

On the last panel we save the VSAM IDCAMS job to member VSERXQ in our primary ICCF library for the purpose of review and later reuse.

Because you saved the job you have to submit it to batch from your primary ICCF library.

Repeat the procedure for the transmit queue to the WebSphere Application Server. You can use the same properties except for the file ID, which should be 'VSE.WAS.TRANSMIT.QUEUE', and the filename, which should be 'VSETWAS'. The job streams you generated this way will also add the DLBL

statements to the 'STDLABUP.PROC' and update the ICCF control definitions needed to display the file name in resource definition dialogs.

The file names 'VSEWAS' and WASVSE' will later be used to associate the MQ local and transmit queues to the VSAM data files.

8.5.2 Defining the MQ files to CICS using RDO

Before you can use the VSAM data files you defined in the previous step, you must make them known to CICS. This is done using the Resource Definition Online (RDO) facility, part of CICS TS. With RDO, defining new resources to CICS is a matter of minutes.

RDO is a CICS transaction that updates the CICS Resource Definition File. This file holds groups of definitions you associate with each other. Groups are then added to lists CICS can load during startup. CICS is able to load more than one list at startup time. In case a definition list references the same definition more than once, the last definition in the list will overwrite the preceding definition of the same type and name. Having this in mind, think carefully about where to place your MQ-related definitions.

In order to use RDO you need a SignOn user ID for CICS with the permission to execute the CEDA transaction.

Now go to CICS and execute the CEDA transaction (CEDA def file) to define the MQ receiver queue and the MQ transmit queue.

Example 8-26 shows how to define the MQ receiver queue.

Example 8-26 RDO definition for the receiver queue

```
OVERTYPE TO MODIFY                                CICS RELEASE = 0411
CEDA DEFine File( WASVSE )
  File      : WASVSE
  Group     : MQMITSO
  DEScription ==> THIS IS THE RECEIVER QUEUE FOR MESSAGES FROM WEBSHERE
VSAM PARAMETERS
DSNAme     ==> WAS.VSE.RECEIVER.QUEUE
Password   ==>                                PASSWORD NOT SPECIFIED
Lsrpoolid  ==> 01                               1-15 | None
Catname    ==> ESCAT10
DSNSharing ==> Noreqs                           Noreqs | Allreqs | Modifyreqs
STRings    ==> 001                              1-255
Nsrgroup   ==>
SHr4access ==> Key                             Key | Rba
REMOTE ATTRIBUTES
REMOTESystem ==>
```

```

REMOTENAME ==>
RECORDSize ==> 1-32767
Keylength ==> 056 1-255
INITIAL STATUS
STatus ==> Enabled Enabled | Disabled | Unenabled
Opentime ==> Firstref Firstref | Startup
BUFFERS
Databuffers ==> 00002 2-32767
Indexbuffers ==> 00001 1-32767
DATATABLE PARAMETERS
Table ==> No No | Cics | User
Maxnumrecs ==> 16-16777215
DATA FORMAT
RECORDFormat ==> V V | F
OPERATIONS
Add ==> Yes No | Yes
Browse ==> Yes No | Yes
DElete ==> Yes No | Yes
READ ==> Yes Yes | No
Update ==> Yes No | Yes
AUTO JOURNALLING
JOurna1 ==> No No | 1-99
JNLRead ==> None None | Updateonly | Readonly | All
JNLSYNCRoad ==> No No | Yes
JNLUpdate ==> No No | Yes
JNLAdd ==> None None | Before | After | All
JNLSYNCRoad ==> Yes Yes | No
RECOVERY PARAMETERS
RECOVery ==> None None | Backoutonly | All
Fwdrecovlog ==> No No | 1-99

```

Example 8-27 shows how to define the MQ transmit queue.

Example 8-27 RDO definition for the transmit queue

```

CEDA DEFine File( VSEWAS )
File : VSEWAS
Group : MQMITSO
DEscription ==> THIS IS THE TRANSMIT QUEUE TO WEBSPHERE
VSAM PARAMETERS
DSName ==> VSE.WAS.TRANSMIT.QUEUE
Password ==> PASSWORD NOT SPECIFIED
Lsrpoolid ==> 01 1-15 | None
Catname ==>
DSNSharing ==> Noreqs Noreqs | Allreqs | Modifyreqs
STRings ==> 001 1-255
Nsrgroup ==>
SHr4access ==> Key Key | Rba
REMOTE ATTRIBUTES

```

```

REMOTESystem ==>
REMOTENAME ==>
RECORDSize ==> 1-32767
Keylength ==> 056 1-255
INITIAL STATUS
STatus ==> Enabled Enabled | Disabled | Unenabled
Opentime ==> Firstref Firstref | Startup
BUFFERS
DATAbuffers ==> 00002 2-32767
Indexbuffers ==> 00001 1-32767
DATATABLE PARAMETERS
Table ==> No No | Cics | User
Maxnumrecs ==> 16-16777215
DATA FORMAT
RECORDFormat ==> V V | F
OPERATIONS
Add ==> Yes No | Yes
Browse ==> Yes No | Yes
DElete ==> Yes No | Yes
REAd ==> Yes Yes | No
Update ==> Yes No | Yes
AUTO JOURNALLING
JOurnal ==> No No | 1-99
JNLRead ==> None None | Updateonly | Readonly | All
JNLSYNRead ==> No No | Yes
JNLUpdate ==> No No | Yes
JNLAdd ==> None None | Before | After | All
JNLSYNWrite ==> Yes Yes | No
RECOVERY PARAMETERS
RECOVery ==> None None | Backoutonly | All
Fwdrecovlog ==> No No | 1-99

```

Please note that the group is MQITSO. If this group does not exist it will be created at the first reference. It is completely up to you to choose every other name or even the name of the group where you already hold all your MQ-related definitions.

The next step is to install the new definitions, so CICS can access them. The RDO file can be shared by more than one CICS. So make sure you are in the same CICS where MQ is supposed to run.

Installing the new definitions is done by entering the command:

```
CEDA INSTALL GROUP (MQITSO)
```

Or you can enter the name of the group you choose. If you have added the definition to an already existing group, open that group and install the new items

by typing an I like INSTALL next to the new definition. Now CICS has access to the new resources. Use CEMT for verification.

In case you created a new group, it is necessary to add this group to the list of the CICS that runs the MQ, so it can load the definitions during startup (Example 8-28).

Example 8-28 CEDA ADD GROUP

```
ADD GR
OVERTYPE TO MODIFY
CEDA ADD
  Group      ==> MQMITSO
  List       ==> vse1stf5
  Before     ==>
  After      ==> mqm

SYSID=CIC5 APPLID=A0006CI2
ADD SUCCESSFUL                TIME: 21.07.45 DATE: 03.315
PF 1 HELP          3 END      6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

You have the option of which list and at which position in the list you want to add your new group MQMITSO.

Now end the CEDA session as follows:

- ▶ CEMT I FILE (WASVSE), as shown in Example 8-29.

Example 8-29 Display WASVSE properties

```
I FILE(WASVSE)
STATUS: RESULTS - OVERTYPE TO MODIFY
Fil(WASVSE ) Vsa Clo Ena Rea Upd Add Bro Del
Dsn( WAS.VSE.RECEIVER.QUEUE ) Cat(ESCAT10)
```

- ▶ CEMT I FILE (VSEWAS), as shown in Example 8-30.

Example 8-30 Display VSETXQ properties

```
I FILE(VSEWAS)
STATUS: RESULTS - OVERTYPE TO MODIFY
Fil(VSEWAS ) Vsa Clo Ena Rea Upd Add Bro Del
Dsn( VSE.WAS.TRANSMIT.QUEUE )
```

Both Example 8-29 and Example 8-30 show the MQ file properties. They are available but still closed. If you want, you can open them at this point.

8.5.3 Defining MQ resources to VSE/ESA

As mentioned before, MQ runs on VSE as a set of transactions under the control of CICS for VSE or CICS TS. All necessary definitions are done calling the MQ master transaction MQMT. Before you can use the MQ environment it must be initialized by running the transaction MQSE with option I.

After executing MQMT, you are in the main menu of the MQ administration facility, as shown in Example 8-31.

Example 8-31 MQMT entry screen

```
11/21/2003      IBM MQSeries for VSE/ESA Version 2.1.2      A0006CI2
17:20:20      *** Master Terminal Main Menu ***      CIC5
MQWMTP                                               A001

                SYSTEM IS ACTIVE

                1. Configuration
                2. Operations
                3. Monitoring
                4. Browse Queue Records

                Option: 1
5686-A06 (C) Copyright IBM Corp. 1998, 2002. All Rights Reserved.
Clear/PF3=Exit                                     Enter=Select 11/11/2003      IBM
```

From the entry panel select option 1 to invoke the Configuration dialog (Example 8-32).

Example 8-32 Configuration main menu

```
MQSeries for VSE/ESA Version 2.1.2      A0006CI2
18:55:32      *** Configuration Main Menu ***      CIC5
MQWMCFG                                              A000

                SYSTEM IS ACTIVE

                Maintenance Options :
                1. Global System Definition
                2. Queue Definitions
                3. Channel Definitions
                4. Code Page Definitions

                Display Options      :
                5. Global System Definition
                6. Queue Definitions
```

- 7. Channel Definitions
- 8. Code Page Definitions

Option: 1

Please enter one of the options listed.

5686-A06 (C) Copyright IBM Corp. 1998, 2002. All Rights Reserved.
 Enter=Process PF2=Return PF3=Exit

Select Global System Definition to take some notes of the system and connection relevant definitions (Example 8-33).

Example 8-33 MQ Global System Definition

```

11/11/2003      IBM MQSeries for VSE/ESA Version 2.1.2      A0006C12
19:01:40              Global System Definition          CIC5
MQWMSYS              Queue Manager Information          A000
Queue Manager . . . . . : QMVSE27
Description Line 1. . . . . : QUEUE MANAGER FOR ESA270
Description Line 2. . . . . :
                        Queue System Values
Maximum Number of Tasks . . : 00001000      System Wait Interval : 00000030
Maximum Concurrent Queues . : 00001000      Max. Recovery Tasks  : 0000
Allow TDQ Write on Errors  : Y   CSMT      Allow Internal Dump  : Y
                        Queue Maximum Values
Maximum Q Depth . . . . . : 01000000      Maximum Global Locks.: 00001000
Maximum Message Size. . . . : 00004096      Maximum Local Locks  : 00001000
Maximum Single Q Access . . : 00000100
                        Global QUEUE /File Names
Local Code Page . . . : 01047
Configuration File. . : MQFCNFG
LOG Queue Name. . . . : SYSTEM.LOG
Dead Letter Name. . . : SYSTEM.DEAD.LETTER.QUEUE
Monitor Queue Name. . : SYSTEM.MONITOR

Requested record displayed.
PF2=Return PF3=Quit PF4/Enter=Read PF6=Upd PF9=Comms PF10=Log

```

8.5.4 Defining MQ local queues on VSE

If you are still in the Gobal System Definition, return to the Configuration Main Menu by pressing PF2. PF2 is always used to return to the upper level menus. PF3 will kick you out and you will have to start with MQMT again.

Enter 2 at the selection panel to enter the Queue Configuration Part of the administration tool (Example 8-34).

Example 8-34 Configuration main menu

11/21/2003	IBM MQSeries for VSE/ESA Version 2.1.2	A0006C12
20:00:25	*** Configuration Main Menu ***	CIC5
MQWMCFG		A001

SYSTEM IS ACTIVE

Maintenance Options :

1. Global System Definition
2. Queue Definitions
3. Channel Definitions
4. Code Page Definitions

Display Options :

5. Global System Definition
6. Queue Definitions
7. Channel Definitions
8. Code Page Definitions

Option: 2

Please enter one of the options listed.

5686-A06 (C) Copyright IBM Corp. 1998, 2002. All Rights Reserved.
Enter=Process PF2=Return PF3=Exit

Option 2 takes you to the queue administration panel.

In a scenario where two Queue Managers exchange messages, both sides need to have the following queues:

- ▶ Sender
- ▶ Transmit
- ▶ Receiver

The sender path from MQ-VSE to MQ on Linux

We start with the definition needed to send messages to the remote system (Example 8-35).

Example 8-35 Queue management selection panel

11/22/2003	IBM MQSeries for VSE/ESA Version 2.1.2	A0006C12
00:04:27	Queue Main Options	CIC5
MQWMQUE		A001

SYSTEM IS ACTIVE

Default Q Manager. : QMVSE27

Object Type. . . . : R L = Local Queue
R = Remote Queue
AQ = Alias Queue
AM = Alias Queue Manager
AR = Alias Reply Queue

Object Name. . . . : VSE270.was

Function has been terminated.

PF2=Return PF3=Quit PF4/Enter=Read PF5=Add PF6=Update
PF9=List PF12=Delete

Enter R for the object type the name for the new queue at the object name definition field and press PF5. Now you are in the section where you have to give all of the details regarding the new queue.

Remote queue definition for the sender path

Example 8-36 illustrates the remote queue definition for the sender path.

Example 8-36 Defining the local queue VSE270.WAS

11/21/2003	IBM MQSeries for VSE/ESA Version 2.1.2	A0006CI2
20:37:10	Queue Definition Record	CIC5
MQWMQUE	QM - QMVSE27	A001

Remote Queue Definition

Object Name. : VSE270.WAS
Description line 1 : QUEUE TO SEND MESSAGES TO THE
Description line 2 : WEBSphere APPLICATION SERVER

Put Enabled : Y Y=Yes, N=No
Get Enabled : Y Y=Yes, N=No

Remote Queue Name. : VSE270.WAS
Remote Queue Manager Name. : MQ11
Transmission Queue Name. . : VSE270.TX.WAS

Record updated OK.
 PF2=Return PF3=Quit PF4/Enter=Read PF5=Add PF6=Update
 PF9=List PF12=Delete

On this panel you have to provide the information that has an impact on your connection.

Object Name This is the name your local applications use to address the remote system. This name is not case sensitive. We decided to use a name equal to the queue name at the remote system that is receiving what we sent.

Put/Get Enabled Leave the fields with the default Y.

Remote Queue Name This must be exactly the name of the queue as it is defined at the remote system. This name is not case sensitive.

Remote Queue Manager Name Here we provide the name of the remote Queue Manager. Because MQ on VSE will store and use the name in uppercase, the Queue Manager at the remote system must have a upper-case name.

Transmission Queue Name This is the name of a local queue. The queue will be defined in the next step. It is a interim queue. All messages are first stored in that queue until delivery is acknowledged by the remote system.

Transmit queue definition for the sender path

Each local queue pointing to a remote queue has to have a transmit queue. The transmit queue works like a buffer for the outgoing messages. Example 8-37 illustrates the transmit queue definition for the sender path.

Example 8-37 Defining the transmit queue - Part 1

11/21/2003	IBM MQSeries for VSE/ESA Version 2.1.2	A0006C12
21:54:28	Queue Definition Record	CIC5
MQWMQUE	QM - QMVSE27	A001

Local Queue Definition

```
Object Name. . . . . : VSE270.TX.WAS
Description line 1 . . . . : QUEUE TO HOLD MESSAGES TILL
```

```

Description line 2 . . . . . : DELIVERED TO REMOTE SYSTEM

Put Enabled . . . . . : Y   Y=Yes, N=No
Get Enabled . . . . . : Y   Y=Yes, N=No

Default Inbound status . . : A   A=Active,I=Inactive
      Outbound status. . . : A   A=Active,I=Inactive

Dual Update Queue. . . . . :

Automatic Reorganize (Y/N) : N   Start Time. : 0000   Interval. . : 0000
VSAM Catalog . . . . . :

Record updated OK.
PF2=Return  PF3=Quit  PF4/Enter=Read  PF5=Add  PF6=Update
                                           PF9=List PF10=Queue  PF12=Delete

```

On this panel you have to provide the following information:

Object Name This is a logical name for the transmit queue. It is used in the remote queue and the sender channel definition.

Put/Get Leave the fields with the default Y.

Default In/Outbound status Leave this option with the default A.

Change to the next panel by pressing PF10 (Example 8-38).

Example 8-38 Defining the transmit queue - Part 2

```

11/21/2003      IBM MQSeries for VSE/ESA Version 2.1.2      A0006C12
23:02:17      Queue Extended Definition                  CIC5
MQWMQUE      QM - QMVSE27                                A001
Object Name. . . . . : VSE270.TX.WAS
                                     Physical Queue Information
Usage Mode . . . . . : T   N=Normal, T=Transmission
Share Mode . . . . . : Y   Y=Yes, N=No
Physical File Name . . . . . : VSEWAS   VSE.WAS.TRANSMIT.QUEUE
                                     Maximum Values
Maximum Q Depth. . . . . : 01000000   Global Lock Entries . . : 00001000
Maximum Message Length . . : 00004096   Local Lock Entries. . . : 00001000
Maximum Concurrent Accesses: 00000100
                                     Trigger Information
Trigger Enable . . . . . : Y   Y=yes, N=No
Trigger Type . . . . . : F   F=First, E=Every
Maximum Trigger Starts . . : 0001
Allow Restart of Trigger : Y   Y=Yes, N=No
Trans ID :
Program ID : MQPSEND
                                     Term ID:
                                     Channel Name: VSE270.WAS

```

User data :
:

PF2=Return PF3=Quit PF4/Enter=Read PF5=Add PF6=Update
PF9=List PF10=Queue

On this panel you have to provide the following information:

Object Name VSE270.TX.WAS is the logical name for the transmit queue. This is taken from the previous panel.

Usage Mode This must be T=Transmission.

Share Mode Leave it with the default Y.

Physical File Name VSEWAS is the filename as defined in the DLBL and RDO definitions.

Default In/Outbound status Leave this option with the default A.

Trigger Enable Y means if a messages appears in the queue some action should happen.

Trigger Type F means that the action should happen at the first occurrence.

Program ID MQPSEND: The CICS Program executed (triggered) if a message is transferred to the transmit queue.

Channel Name VSE270.WAS is the name of the channel that holds the information for the target system.

Sender channel definition

Example 8-39 shows what is needed to define the network connection to the remote system and link the connection to the transmit queue for message transfer.

Example 8-39 Define the sender channel properties

```
11/24/2003      IBM MQSeries for VSE/ESA Version 2.1.2      A0006C12
19:15:23              Channel Record          UPDATE      CICS
MQWMCHN      Channel name. . : VSE270.WAS          A000
Protocol: T (L/T)      Type : S (Sender/Receiver/Client)
```

Sender

```
Partner/Rem: 9.12.9.12
Remote TCP/IP Port . . . . : 01414
Get retry number . . . . . : 00000000    LU62 Allocation Retry Num : 00000000
Get retry delay (secs) . . : 00000000    LU62 Delay fast (secs). . : 00000000
```



```

Convert Msgs(Y/N). . . . . : Y          LU62 Delay slow (secs). . : 00000000
Transmission Queue Name. . : VSE270.TX.WAS
TP Name. . . . . :
Receiver
Dead Letter Store(Y/N) . . : N
Sender/Receiver
Max Messages per Batch . . : 000001     Message Sequence Wrap. . : 999999
Sender/Receiver/Client
Max Transmission Size . . : 032766     Max Message Size . . . . : 0004096
Split Msg(Y/N) . . . . . : N
Enable(Y/N). . . . . : Y

Enter fields to be updated.
F2=Return PF3=Quit PF4=Read PF5=Add PF6=Upd PF9=List PF10=SSL PF12=Del

```

On this panel you have to provide the following information:

Channel name	VSE270.WAS: The name for the sender channel must appear in the transmit queue definition. This is the linking element to the transmit queue.
Protocol	T: For TCP/IP because Linux only supports this type of protocol.
Type	S: This is a sender type channel.
Partner/Rem	Either the remote host name or its IP address.
Remote TCP/IP Port	1414: The port the other system is listening on.
Convert Msgs(Y/N)	Y: Because we connect to ASCII type system.
Transmission Queue Name	VSE270.TX.WAS: The name we defined in the former steps.
Max Messages per Batch	Leave it with the default, both MQ managers will negotiate about this value.
Message Sequence Wrap	999999 is the maximum VSE can handle. See also "Defining the receiver channel" on page 173.
Channel Name	VSE270.WAS: The name of the channel that holds the information for the target system.
Max Transmission Size	Leave it with the default, both MQ managers will negotiate about this value.

Max Message Size

Leave it with the default, both MQ managers will negotiate about this value.

Enable (Y?N)

Y: Yes, this channel should become active when MQ gets initialized.

The receiver path from MQ-VSE to MQ on Linux

We continue with the definition needed to receive messages from the remote system.

Receiver queue definition

Example 8-40 illustrates the definition of the local receiving queue.

Example 8-40 Defining the local receiving queue (part 1)

11/21/2003	IBM MQSeries for VSE/ESA Version 2.1.2	A0006CI2
21:46:43	Queue Definition Record	CIC5
MQWMQUE	QM - QMVSE27	A001

Local Queue Definition

Object Name. : WAS.VSE270
 Description line 1 : queue to receive messages
 Description line 2 : from websphere

Put Enabled : Y Y=Yes, N=No
 Get Enabled : Y Y=Yes, N=No

Default Inbound status . . : A A=Active,I=Inactive
 Outbound status. . . : A A=Active,I=Inactive

Dual Update Queue. :

Automatic Reorganize (Y/N) : N Start Time. : 0000 Interval. . : 0000
 VSAM Catalog :

Record being updated - Press UPDATE key again.

PF2=Return PF3=Quit PF4/Enter=Read PF5=Add PF6=Update
 PF9=List PF10=Queue PF12=Delete

On this panel you have to provide the following information:

Object Name

WAS.VSE270: This is the name of the local on which the system will receive messages from other systems.

Put/Get Enabled

Leave the fields with the default Y.

Default In/Outbound status

Leave it with the default A.

Automatic Reorganize (Y?N) N

Continue as shown in Example 8-41.

Example 8-41 Defining the local receiving queue (part 2)

```
11/21/2003      IBM MQSeries for VSE/ESA Version 2.1.2      A0006C12
21:50:11              Queue Extended Definition      CIC5
MQWMQUE          QM - QMVSE27                        A001
Object Name. . . . . : WAS.VSE270
                    Physical Queue Information
Usage Mode . . . . . : N      N=Normal, T=Transmission
Share Mode . . . . . : Y      Y=Yes, N=No
Physical File Name . . . . : WASVSE      WAS.VSE.RECEIVER.QUEUE
                    Maximum Values
Maximum Q Depth. . . . . : 01000000      Global Lock Entries . . : 00001000
Maximum Message Length . . : 00004096      Local Lock Entries. . . : 00001000
Maximum Concurrent Accesses: 00000100
                    Trigger Information
Trigger Enable . . . . . : N      Y=yes, N=No
Trigger Type . . . . . : F      F=First, E=Every
Maximum Trigger Starts . . : 0001
Allow Restart of Trigger : N      Y=Yes, N=No
Trans ID :                               Term ID:
Program ID :                             Channel Name: WAS.VSE270
User data :
:

PF2=Return PF3=Quit PF4/Enter=Read PF5=Add PF6=Update
PF9=List PF10=Queue
```

On this panel you have to provide the following information:

- | | |
|-----------------------------------|---|
| Object Name | WAS.VSE270: The logical name for the local queue. This is taken from the pervious panel. |
| Usage Mode | Must be N=Normal. |
| Share Mode | Leave it with the default Y. |
| Physical File Name | WASVSE: This is the filename as defined in the DLBL and RDO definitions. |
| Default In/Outbound status | Leave this option with the default A. |
| Trigger Enable | N: We set it to no because we do not have a program to pick up the messages. Messages will stay in the queue as long as they are read by some application or deleted. |

Trigger Type F has no meaning because trigger enable is no.

Program ID Here we would name the CICS Program to be executed (triggered) if we had one.

Channel Name Name of the receiver channel we will define next.

Sender channel definition

Example 8-42 illustrates how to define the receiver channel properties.

Example 8-42 Define the receiver channel properties

```

11/24/2003      IBM MQSeries for VSE/ESA Version 2.1.2      A0006C12
21:07:47              Channel Record      DISPLAY      CICS5
MQWMCHN  Channel name. . . : WAS.VSE270      A000
Protocol: T (L/T)      Type : R (Sender/Receiver/Client)

Sender
Partner/Rem:
Remote TCP/IP Port . . . . : 01414
Get retry number . . . . . : 00000000      LU62 Allocation Retry Num : 00000000
Get retry delay (secs) . . : 00000000      LU62 Delay fast (secs) . . : 00000000
Convert Msgs(Y/N) . . . . . : N              LU62 Delay slow (secs) . . : 00000000
Transmission Queue Name. . :
TP Name. . . :
Receiver
Dead Letter Store(Y/N) . . : N
Sender/Receiver
Max Messages per Batch . . : 000001      Message Sequence Wrap. . . : 999999
Sender/Receiver/Client
Max Transmission Size . . . : 032766      Max Message Size . . . . . : 0004096
Split Msg(Y/N) . . . . . : N
Enable(Y/N) . . . . . : Y

Channel record displayed.
F2=Return PF3=Quit PF4=Read PF5=Add PF6=Upd PF9=List PF10=SSL PF12=De1

```

On this panel you have to provide the following information:

Channel name WAS.VSE270: The name for the receiver channel must appear in the local queue definition. This is the linking element to the receiver queue.

Protocol T for TCP/IP because Linux only supports this type of protocol.

Type R: This is a receiver type channel.

Partner/Rem	Leave empty.
Remote TCP/IP Port	1414: The port the local system is listening on.
Convert Msgs(Y/N)	N: The sender will do the conversion from ASCII to EBCDIC.
Transmission Queue Name	Leave empty.
Max Messages per Batch	Leave it with the default; both MQ managers will negotiate about this value.
Message Sequence Wrap	999999 is the maximum VSE can handle. See also “Defining the sender channel” on page 172.
Max Transmission Size	Leave it with the default; both MQ managers will negotiate about this value.
Max Message Size	Leave it with the default; both MQ managers will negotiate about this value.
Enable (Y/N)	Y: Yes, this channel should become active when MQ gets initialized.

8.5.5 MQ troubleshooting

In case that the connection between Linux and VSE will not start, there are a few things you should check first.

On Linux

Log in as user root and enter the command:

```
dspmq
```

Make sure that your MQ is up and running. Next check if the listener is running and connected to the right port. The port must be the same as in your counterpart definitions on MQ for VSE or MQ for z/OS. That can be done with the following command:

```
ps -ef | grep runmq1sr
```

You receive an output as shown in Example 8-43. Watch out for the name of the MQ manager you have trouble with. If it does not appear in the list, it means that it is not started. Try to start the listener with the `runmq1sr` command. Before you do that, check the port numbers on the display. You have to use a different port for every Queue Manager.

Example 8-43 Show all active MQ listeners

```
linux11:/home/jhae # ps -fe | grep runmq1sr
mqm      1651 1650 0 Nov21 ?      00:00:03 runmq1sr -t tcp -p 5558 -m WAS_linux11_server1
mqm      1652 1651 0 Nov21 ?      00:00:00 runmq1sr -t tcp -p 5558 -m WAS_linux11_server1
mqm      1653 1652 0 Nov21 ?      00:00:00 runmq1sr -t tcp -p 5558 -m WAS_linux11_server1
mqm      1654 1652 0 Nov21 ?      00:00:00 runmq1sr -t tcp -p 5558 -m WAS_linux11_server1
mqm      1655 1652 0 Nov21 ?      00:00:00 runmq1sr -t tcp -p 5558 -m WAS_linux11_server1
mqm      21222 1 0 Nov25 ?      00:00:00 runmq1sr -m SC52LINUX11 -t tcp
mqm      21223 21222 0 Nov25 ?      00:00:00 runmq1sr -m SC52LINUX11 -t tcp
mqm      21224 21223 0 Nov25 ?      00:00:00 runmq1sr -m SC52LINUX11 -t tcp
mqm      21225 21223 0 Nov25 ?      00:00:00 runmq1sr -m SC52LINUX11 -t tcp
mqm      21226 21223 0 Nov25 ?      00:00:00 runmq1sr -m SC52LINUX11 -t tcp
mqm      6043 25162 0 Nov27 pts/2    00:00:00 runmq1sr -m MQ11 -t tcp -p 1450
mqm      6044 6043 0 Nov27 pts/2    00:00:00 runmq1sr -m MQ11 -t tcp -p 1450
mqm      6045 6044 0 Nov27 pts/2    00:00:00 runmq1sr -m MQ11 -t tcp -p 1450
mqm      6046 6044 0 Nov27 pts/2    00:00:00 runmq1sr -m MQ11 -t tcp -p 1450
mqm      6047 6044 0 Nov27 pts/2    00:00:00 runmq1sr -m MQ11 -t tcp -p 1450
root     18161 24994 0 09:36 pts/5    00:00:00 grep runmq1sr
linux11:/home/jhae #
```

If you see a listener with no port information, this listener is using the default port 1414. In a system where more than one MQ listener is active, this has an impact on the port definitions of the counterpart as well. To start the listener with a different port, you have to provide the port number with option `-p`, such as:

```
runmq1sr -m MQ11 -t tcp -p 2000
```

During your tests, it may happen that you try to connect from two different Queue Managers to the local queue and could successfully transmit one or more messages only from the first Queue Manager. If that happens, it means that the local message counter of your local queue differs from the message counter of the second remote Queue Manager on the other side. In this situation no further messages are accepted until the counters on both sides have been reset or adjusted to each other.

8.6 Configuring the MQ connector in WebSphere

Configuring the MQ connector in WebSphere consists of the following steps:

- ▶ Define a WebSphere MQ queue connection factory.
- ▶ Define WebSphere MQ queue destinations.
- ▶ Define message listeners.

The following sections describe each of these steps.

Note: You do not have to create J2C Authentication entries for MQ.

8.6.1 Defining a WebSphere MQ queue connection factory

To define a queue connection factory in the WebSphere Administrative Console:

1. Click **Resources** → **WebSphere MQ JMS Provider**.
2. Click **WebSphere MQ Queue Connection Factories**.
3. Create a new queue connection factory and enter the following parameters:
 - Name: TraderQCF.
 - JNDI Name: jms/TraderQCF.
 - Queue Manager: Specify the name of your local MQ Queue Manager.All other fields should be left as default. Click **OK** to proceed.

8.6.2 Defining WebSphere MQ queue destinations

Next, you have to create several queue destination resources:

1. Click **WebSphere MQ Queue Destinations**.
2. Create the following five queue destinations, as shown in Table 8-2.

Table 8-2 Queue destinations used in Trader application

Name	JNDI name	Base queue name	Target client
TraderCICSReqQ	jms/TraderCICSReqQ	TRADER.CICS.BRIDGEQ	MQ
TraderCICSRepQ	jms/TraderCICSRepQ	TRADER.CICS.REPLYQ	JMS
TraderIMSReqQ	jms/TraderIMSReqQ	TRADER.IMS.BRIDGEQ	MQ
TraderIMSRepQ	jms/TraderIMSRepQ	TRADER.IMS.REPLYQ	JMS
TraderProcessQ	jms/TraderProcessQ	TRADER.PROCESSQ	JMS

8.6.3 Defining message listeners

Message listeners are used by the Message Driven Bean (MDB). Even if you do not use MDB option in the Trader application, you have to define Message Listeners, otherwise the server will not run correctly.

1. Click **Application Servers** → *server_name* → **Message Listener Service**.
2. Click **Listener Ports**.
3. Click **New** to define two Message Listeners.

Message Listener #1:

- Name: TraderMQCICSListener
- Initial State: Change to Started if you configure to use MDB option

- Connection Factory JNDI name: jms/TraderQCF
- Destination JNDI name: jms/TraderCICSRepQ

Message Listener #2:

- Name: TraderMQIMSListener
- Initial State: Change to Started if you configure to use MDB option
- Connection Factory JNDI name: jms/TraderQCF
- Destination JNDI name: jms/TraderIMSRepQ

Then click **OK**.

4. Save the current configuration.

8.6.4 Deploying TraderMQ application to WebSphere

In this chapter we look at the part of Trader that focuses on MQ connections.

1. Download and unzip the Trader install package to a directory that you have read/write authority to, for example, /temp.
2. Copy the MQ EAR file to WAS's installable applications:

```
cd /temp
cp TraderMQEar.ear /opt/WebSphere/AppServer/installableApps/TraderMQEar.ear
```

3. Open the WebSphere Administrative Console and select **Applications** → **Install New Application**.
4. On the panel Preparing for the application installation, select **Server path**, enter /opt/WebSphere/AppServer/installableApps/TraderMQEar.ear in the field, and click **Next** (Figure 8-6).

The screenshot shows a dialog box titled "Preparing for the application installation". It has two main sections: "Path" and "Context Root".

- Path:** This section is titled "Browse the local machine or a remote server:". It contains two radio buttons: "Local path:" (unselected) and "Server path:" (selected). Below "Local path:" is an empty text field and a "Browse..." button. Below "Server path:" is a text field containing the path "/opt/WebSphere/AppServer/installableApps/TraderMQEar.ear".
- Context Root:** This section is titled "Used only for standalone Web modules (*.war)". It contains an empty text field.

There are two informational icons (i) on the right side of the dialog. The top one says "Choose the local path or the server path if the e...". The bottom one says "You must specify a...".

At the bottom of the dialog are two buttons: "Next" and "Cancel".

Figure 8-6 Preparing for the application installation

5. On the subsequent panels, click **Next** to accept all defaults. On the last one, click **Finish**.
6. Click **Save** to save the new configuration.

7. Go to **Applications** → **Enterprise Applications** and start the application. Put a checkmark at the TraderMQEar application and click **Start** (see Figure 8-7).












<input type="checkbox"/> Name ▾		Status ▾ 
<input type="checkbox"/>	DefaultApplication	
<input type="checkbox"/>	MDBSamples	
<input type="checkbox"/>	PlantsByWebSphere	
<input type="checkbox"/>	SamplesGallery	
<input type="checkbox"/>	TechnologySamples	
<input type="checkbox"/>	TraderCICSEAR	
<input type="checkbox"/>	TraderDBEAR	
<input type="checkbox"/>	TraderIMSEAR	
<input checked="" type="checkbox"/>	TraderMQEar	
<input type="checkbox"/>	VSEServlets	

Figure 8-7 Starting the TraderMQEar application

8. To work with the application, enter the following URL in a Web browser to access the MQ part of the Trader application, and you will see a screen similar to Figure 8-8:

`http://yourServer/TraderMQWeb/`

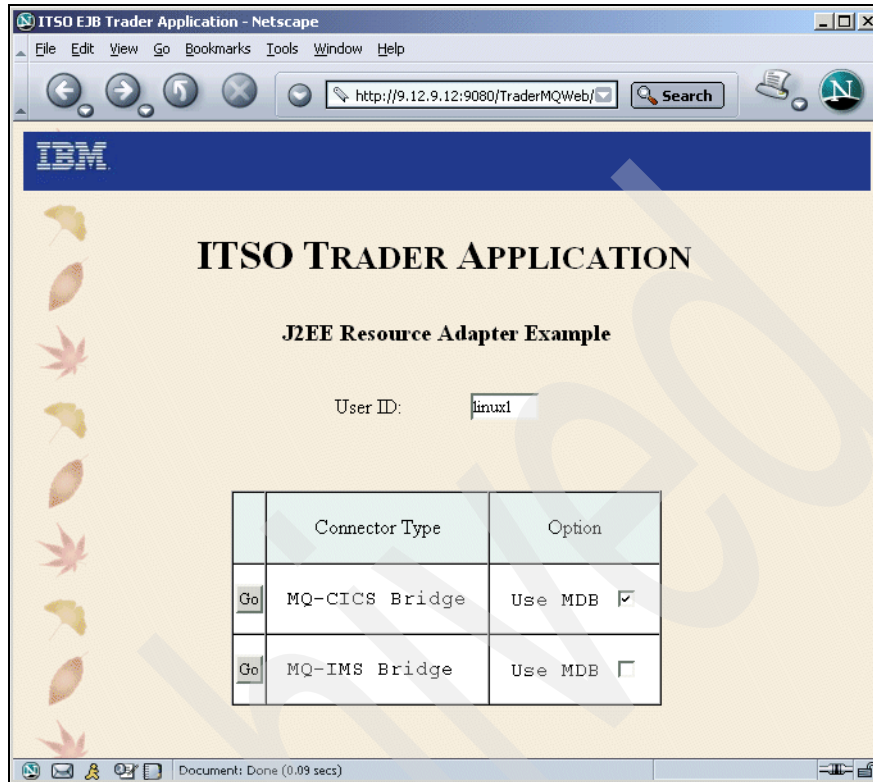


Figure 8-8 The ITSO Trader application using the MQ connectors

9. Type a user ID, for example linux1, and click **Go** at either MQ-CICS Bridge or at MQ-IMS bridge depending on the connection you want to test. If you want Trader to use a message driven bean (MDB) to receive the reply, then put a checkmark in the Use MDB box.

8.6.5 Defining resources for VSE in WebSphere

Defining WebSphere resources for VSE consists of four steps:

1. Define an MQ queue connection factory.
2. Define an MQ queue.
3. Create resource references in application .ear file.
4. Redeploy .ear file in the WebSphere admin console.

The following sections describe each of these steps.

Step 1: Defining an MQ connection factory

Open the WebSphere administration console. In the WebSphere admin console do as follows:

1. Click **Resources** → **WebSphere MQ JMS Provider**.



2. Click **WebSphere MQ Queue Connection Factories**.



3. Create a new MQ queue factory with the following properties (see figure below).

General Properties	
Scope	* cells:linux11:nodes:linux11
Name	* TraderVSE
JNDI Name	* jms/queue/QueueConnectionFactory
Description	MQ Connection Factory for VSE

We defined the following additional properties:

Queue Manager MQ11
XA Enabled NO

For the definition of the local MQ Queue Manager MQ11 on Linux refer to “Defining MQ resources to Linux for zSeries” on page 168.

4. Click **OK** and save your definitions.

Step 2: Defining an MQ queue

In the WebSphere admin console do as follows:

1. Click **Resources** → **WebSphere MQ JMS Provider**.
2. Click **WebSphere MQ Queue Destinations**.

Additional Properties	
WebSphere MQ Queue Connection Factories	
WebSphere MQ Topic Connection Factories	
WebSphere MQ Queue Destinations	
WebSphere MQ Topic Destinations	

3. Create a new MQ Queue with the following properties (see figure below).

General Properties	
Scope	* cells:linux11:nodes:linux11
Name	* VSE MQ Queue Destination
JNDI Name	* jms/queue/Queue
Description	MQ Queue to VSE

We defined the following additional properties:

Base Queue Name WAS.VSE270
Target Client MQ

Scroll down to define the following MQ Queue Connection properties (see figure below).

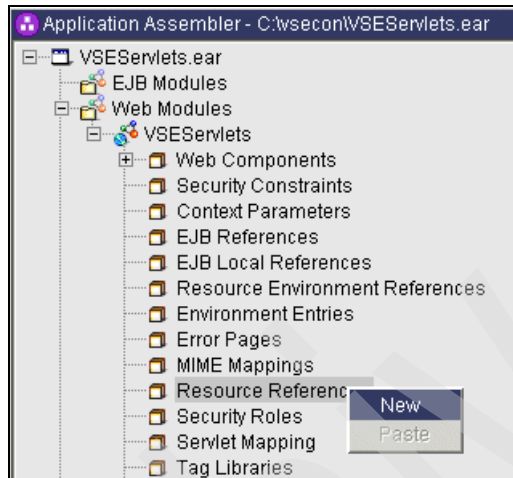
WebSphere MQ Queue Connection Properties	
Queue Manager Host	9.12.6.191
Queue Manager Port	1414
Server Connection Channel Name	WAS.VSE270

4. Click **OK** and save your changes.

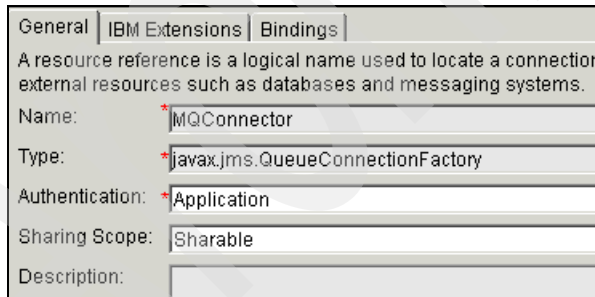
Step 3: Creating resource references in the EAR file

Start the Application Assembly tool and open your EAR file. Now create resource references for the MQ queue connection factory and the MQ queue.

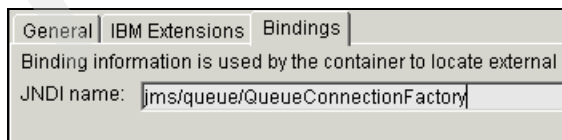
1. Expand the tree view to the Resource References node, right-click the node, and select **New**.



2. Create a new resource reference with the following properties (see figure below).



3. Specify JNDI name.



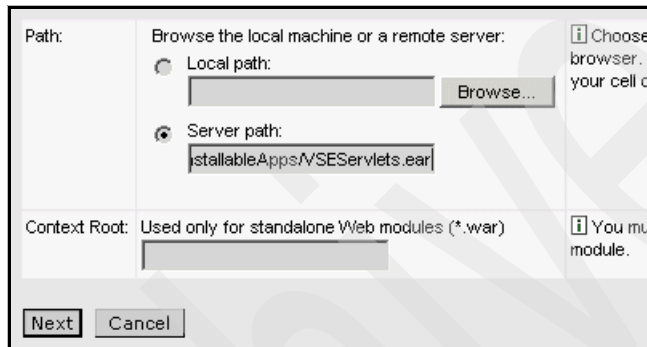
4. Click **OK** and save the EAR file.

5. Transfer the new EAR file to the WebSphere platform. We did an FTP into the /installableApps directory.

Step 4: Redeploying EAR file

Open the WebSphere admin console and do as follows:

1. Click **Applications** → **Enterprise Applications**.
2. Stop your application.
3. Select your application and click the **Update** button.
4. Specify the ear file pathname.



Path: Browse the local machine or a remote server:

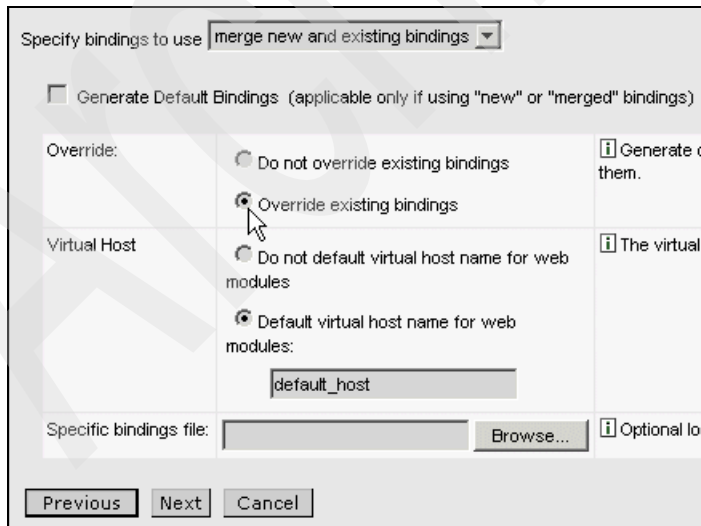
Local path: Browse...

Server path:

Context Root: Used only for standalone Web modules (*.war)

Next Cancel

5. In the next panel select **Override existing bindings**.



Specify bindings to use: merge new and existing bindings

Generate Default Bindings (applicable only if using "new" or "merged" bindings)

Override: Do not override existing bindings Override existing bindings

Virtual Host: Do not default virtual host name for web modules Default virtual host name for web modules:

Specific bindings file: Browse...

Previous Next Cancel

6. Provide further options unless already shown.

Step 1 : Provide options to perform the installation

Specify the various options available to prepare and install your application.

AppDeployment Options	Enable
Pre-compile JSP	<input type="checkbox"/>
Directory to Install Application	ppServer/installedApps/linux11
Distribute Application	<input checked="" type="checkbox"/>
Use Binary Configuration	<input type="checkbox"/>
Deploy EJBs	<input type="checkbox"/>
Application Name	VSEServlets
Create MBeans for Resources	<input checked="" type="checkbox"/>
Enable Class Reloading	<input checked="" type="checkbox"/>
Reload Interval in Seconds	3
Deploy WebServices	<input type="checkbox"/>

Next Cancel

7. Map resource references for MQ queue and connection factory.

javax.jms.Queue

Specify existing Resource JNDI name: linux11:jms/queue/Queue Apply

<input type="checkbox"/>	Module	EJB	URI	Reference Binding	JNDI Name
<input type="checkbox"/>	VSEServlets		VSEServlets.war/WEB-INF/web.xml	MQQueue	jms/queue/Queue

javax.jms.QueueConnectionFactory

Specify existing Resource JNDI name: linux11:jms/queue/QueueConnectionFactory Apply

<input type="checkbox"/>	Module	EJB	URI	Reference Binding	JNDI Name
<input type="checkbox"/>	VSEServlets		VSEServlets.war/WEB-INF/web.xml	MQConnector	jms/queue/QueueConn

8. Proceed with the panels by clicking **Next**.

9. After finishing the deployment steps, save your changes.

10. Restart your application.

Step 5: Using the MQ connection from a servlet

The code snippet shown in Example 8-44 can be used in a servlet to look up the MQ connection and MQ queue and send a test string to VSE. Here, the previously specified JNDI names of the queue connection factory and the MQ queue are used to reference the respective WebSphere definitions.

Example 8-44 Use MQ queue connection in a servlet

```
String qcfName = "jms/queue/QueueConnectionFactory";
String qName = "jms/queue/Queue";

try {
    Context ctx = new InitialContext();
    QueueConnectionFactory qcf = (QueueConnectionFactory) ctx.lookup(qcfName);
    QueueConnection connection = qcf.createQueueConnection();
    connection.start();

    boolean transacted = false;
    QueueSession session = connection.createQueueSession( transacted,
                                                         Session.AUTO_ACKNOWLEDGE);

    Queue ioQueue = (Queue)ctx.lookup(qName);
    QueueSender queueSender = session.createSender(ioQueue);

    TextMessage outMessage = session.createTextMessage();
    outMessage.setText("Hello VSE");
    queueSender.send(outMessage);
    QueueReceiver queueReceiver = session.createReceiver(ioQueue);
    Message inMessage = queueReceiver.receive(1000); // wait 1 sec.

    queueReceiver.close();
    queueSender.close();
    session.close();
    connection.close();
}
catch (Exception e)
{
    ...
}
```

IMS J2EE connectors

This chapter provides a description of the IMS J2EE connectors and describes the WebSphere Application Server V5.0.2 for Linux on zSeries and IMS Connector for Java V2.1.0.1 configuration steps to install and execute the IMS Trader sample application.

We discuss the following topics:

- ▶ IMS connectors overview
- ▶ Installing the IMS Connector for Java
- ▶ Installing the IMS resource adapter in WebSphere Application Server
- ▶ Configuring IMS J2C connection factories
- ▶ IMS Connect configuration
- ▶ Testing TraderIMS application
- ▶ Problem determination
- ▶ Thread identity support

9.1 IMS connectors overview

Figure 9-1 gives an overview of backend IMS Connect and how it works with frontend IMS Connector for Java.

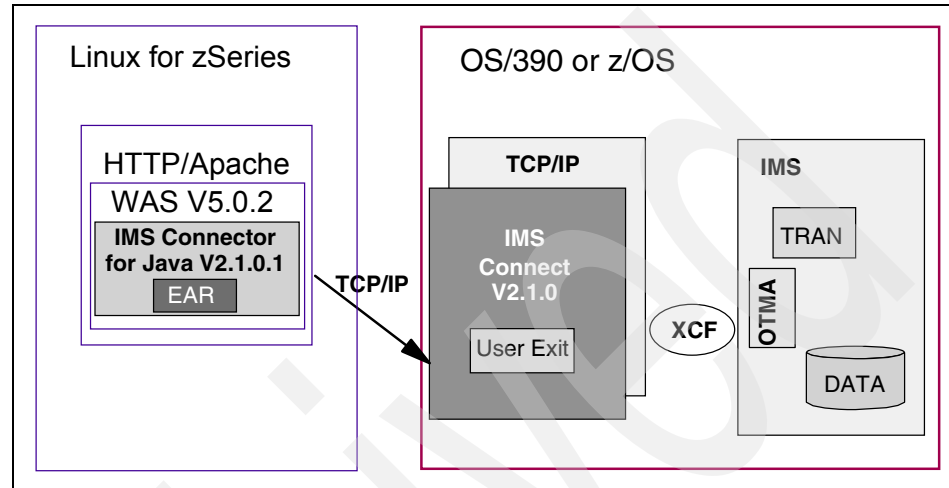


Figure 9-1 IMS Connection from Linux for zSeries to z/OS

9.1.1 IMS Connect

IMS Connect is a TCP/IP server that runs on OS/390 and enables TCP/IP clients to exchange messages with IMS Open Transaction Manager Access (OTMA). As shown in Figure 9-1, this server provides communication links between TCP/IP clients (such as our IMS Trader application using the IMS resource adapter running in WebSphere Application Server for Linux for zSeries) and IMS (datastores).

9.1.2 IMS Connector for Java

The IMS resource adapter is also called IMS Connector for Java. The IMS resource adapter is used during development to create Enterprise Service Java applications, as shown in Figure 9-2, and is used by those Java applications during runtime to access IMS transactions running on host IMS systems, as shown in Figure 9-1. WebSphere Studio Application Developer Integration Edition (WSAD.IE) is a service-based development environment, and the IMS resource adapter is one of the service providers included in it. It is also provided with IMS Connect. We used WSAD IE V5.1 to create our IMS Trader application and will be deploying the ear file that WSAD.IE created to WebSphere Application Server V5.0.1.

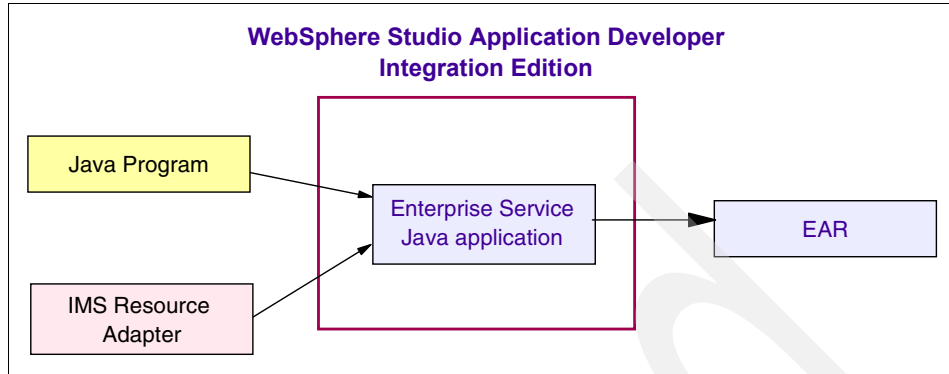


Figure 9-2 WebSphere Studio Application Developer's use of IMS resource adapter

9.2 Installing IMS Connector for Java

The code IMS Connector for Java needs to be installed on our Linux for zSeries system. We accessed the IMS Connector for Java executable from our NFS Server.

We then ran the executable with the following command:

```
./imsicosetup_linux390_2101
```

We were then presented with a Welcome Panel, as shown in Figure 9-3. We clicked **Next**.

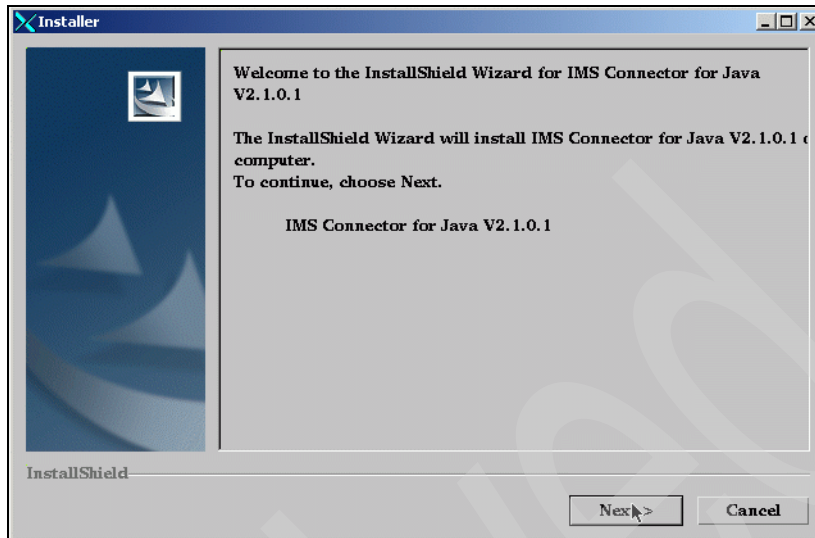


Figure 9-3 Install Wizard for IMS Connector for Java

We were then presented with a panel where we could install to the default directory, /opt, or a different directory. We accepted the default and clicked **Next**.

We were then presented with an informational panel stating the total size for the IMS Connector. We clicked **Next**.

The installer showed that it was updating the RPM database, then presented us with a panel stating that the Install Wizard had successfully installed IMS Connector for Java V2.1.0.1. Click **Finish**.

Successful completion of the installation created a new directory called /opt/IMSICO, which contained several files; one of which is the IMS resource adapter (imsico.rar). At run time, the IMS resource adapter is used with IBM WebSphere Application Server. We now describe the process to install and configure the IMS resource adapter for WebSphere Application Server so we can run our IMS Trader application.

9.3 Installing the IMS resource adapter in WebSphere Application Server

Now that we have our code installed on our Linux for zSeries system, we need to install the IMS Resource Adapter (imsico.rar) in WebSphere Application Server.

On the WebSphere Administrative Console (Figure 9-4), we expanded Resources, clicked **Resource Adapters**, selected the node on which we wanted to install the resource adapter, and then clicked **Install RAR** to start the installation process.

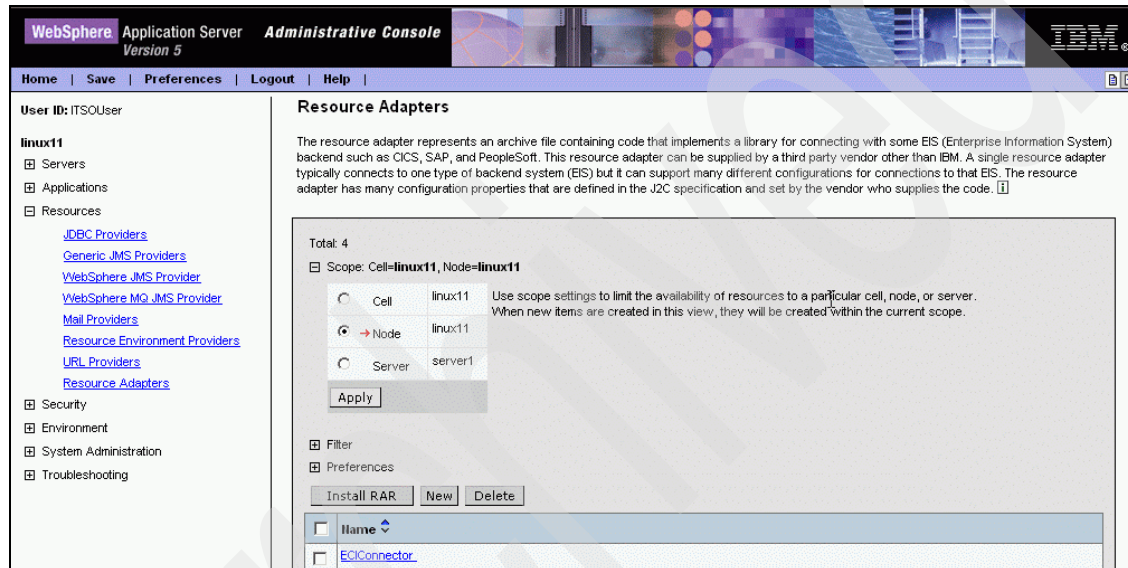


Figure 9-4 WebSphere Application Server V5.0.2 for Linux on zSeries Administration Console

On the next panel (Figure 9-5 on page 212), we chose to use the server path for IMS resource adapter and made sure that the desired node for this resource was our Linux for zSeries system (that is, linux11). We then clicked **Next**.

Install RAR File

RAR files can be installed using two methods. You can choose to upload a RAR file from local file system or you can specify an existing RAR file on a server.

Path	Browse the local machine or a remote server:		Choose the local path if the RAR resides on the same machine as the browser. Choose the server path if the RAR resides on any of the nodes in your cell context.
	<input type="radio"/> Local path: <input type="text"/> <input type="button" value="Browse..."/>	<input checked="" type="radio"/> Server path: <input type="text" value="/opt/IMS/imsico.rar"/>	
Scope	Node: <input type="text" value="linux11"/> <input type="button" value="v"/>		The RAR file will be installed and extracted on the selected node. Installation will create a resource adapter in the configuration at this scope.

Figure 9-5 Install the IMS RAR file

Tip: RAR files can be installed from either of two locations. The file can be loaded from a path on the local workstation to the browser, or from a path on the server. There may be times when the RAR file shipped with WSAD.IE is the preferred source since it may contain maintenance not available on the host.

A resource adapter configuration panel will be displayed (Figure 9-6). We entered IMS Connect as the name for this resource adapter and clicked **OK**.

Configuration		
General Properties		
Scope	* cells:linux11:nodes:linux11	<input type="checkbox"/> The scope of the configured resource provider. This value indicates the configuration location for the configuration file.
Name	IMS Connect	<input type="checkbox"/> The name of the resource adapter. If this property is not specified, the value of the display-name in the deployment descriptor will be used.
Description	IMS Connect Resource Adapter	<input type="checkbox"/> A text description for the resource provider.
Archive Path		<input type="checkbox"/> Path to the installed RAR file containing the module for this resource adapter. If this property is not specified, the archive will be extracted to the absolute path represented by the symbolic name "CONNECTOR_INSTALL_ROOT". "CONNECTOR_INSTALL_ROOT" is a variable defined in variables.xml file.
Classpath		<input type="checkbox"/> A list of paths or JAR file names which together form the location for the resource provider classes. Classpath entries are separated by using the ENTER key and must not contain path separator characters (such as '/' or '.'). Classpaths may contain variable (symbolic) names which can be substituted using a variable map. Check your drivers installation notes for specific JAR file names which are required.
Native Path		<input type="checkbox"/> An optional path to any native libraries (.dll's, .so's). Native path entries are separated by using the ENTER key and must not contain path separator characters (such as '/' or '.'). Native paths may contain variable (symbolic) names which can be substituted using a variable map.
<input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>		

Figure 9-6 Resource adapter configuration

The adapter is now displayed as being installed on this node, as shown on Figure 9-7 on page 213.

Total: 4	
<input checked="" type="checkbox"/> Scope: Cell= linux11 , Node= linux11	
<input checked="" type="checkbox"/> Filter	
<input checked="" type="checkbox"/> Preferences	
<input type="button" value="Install RAR"/> <input type="button" value="New"/> <input type="button" value="Delete"/>	
<input type="checkbox"/> Name	
<input type="checkbox"/>	ECJConnector
<input type="checkbox"/>	IMS Connect
<input type="checkbox"/>	YSEConnector
	WebSphere Relational Resource Adapter

Figure 9-7 List of installed resource adapters

We could have saved our configuration changes at this time, but since WebSphere Application Server will obtain the security information from the J2C connection factory's custom properties, we decided to configure the connection factory for the IMS adapter resource next.

9.4 Configuring IMS J2C connection factories

We clicked the newly installed IMS resource adapter, **IMS Connect**, to display its Configuration panel. We then entered the archive path and the classpath of the RAR file (Figure 9-8) and clicked **Apply**.

General Properties		
Scope	* cells:linux11:nodes:linux11	The scope of the configured resource provider. This value indicates the configuration location for the configuration file.
Name	* IMS Connect	The name of the resource provider.
Description	IMS Connect Resource Adapter	A text description for the resource provider.
Archive Path	* /opt/IMS/imsico.rar	Path to the installed RAR file containing the module for this resource adapter.
Classpath	/opt/IMS/imsico.rar	A list of paths or JAR file names which together form the location for the resource provider classes. Classpath entries are separated by using the ENTER key and must not contain path separator characters (such as "/" or "\"). Classpaths may contain variable (symbolic) names which can be substituted using a variable map. Check your drivers installation notes for specific JAR file names which are required.
Native Path		An optional path to any native libraries (.dll's, .so's). Native path entries are separated by using the ENTER key and must not contain path separator characters (such as "/" or "\"). Native paths may contain variable (symbolic) names which can be substituted using a variable map.

Apply OK Reset Cancel

Additional Properties

[J2C Connection Factories](#) J2C Connection Factories represent a set of connection configuration values.

[Custom Properties](#) Properties that may be required for Resource Providers and Resource Factories. For example, most database vendors require additional custom properties for data sources that will access the database.

[View Deployment Descriptor](#) View the Deployment Descriptor

Figure 9-8 IMS Connect Configuration panel

We next clicked **J2C Connection Factories** at the bottom of the IMS Connect - Configuration panel to display the J2C Connection Factories panel (not shown). On this panel we clicked **New** to open the New Configuration panel.

On the New J2C Connection Factory panel, we entered IMS Connection Factory for the name of the factory and for its JNDI name (Figure 9-9).

Important: There are two types of IMS Connect J2C connection factories. One directly accesses IMS Connect using XCF (LOCAL), which can be used only if you are on the same system as IMS; and the other accesses IMS Connect using TCP/IP (REMOTE), which can be used if you are on the same or on a different system as the IMS. We are using the remote type.

Configuration		
General Properties		
Scope	* cells:linux11:nodes:linux11	<input type="checkbox"/> The scope of the configured resource. This value indicates the configuration location for the configuration file.
Name	* IMS Connection Factory	<input type="checkbox"/> The required display name for the resource.
JNDI name	itsso/ims/j2ee/trader/TraderIMSComme	<input type="checkbox"/> The JNDI name for the resource, including any naming subcontexts. The name is used to link the platform binding information. The binding associates the resources defined the deployment descriptor of the module to the actual (physical) resources bound into JNDI by the platform.
Description	IMS Connection Factory for z/OS	<input type="checkbox"/> An optional description for the resource.
Category		<input type="checkbox"/> An optional category string which can be used to classify or group the resource.
Authentication Preference	BASIC_PASSWORD	<input type="checkbox"/> Specifies which of the authentication mechanisms which are defined for the corresponding resource adapter applies to this connection factory. For example, if two auth mechanism entries have been defined for a resource adapter, KerbVS and Basic Password, this will specify one of those two types. If the auth mechanism preference specified is not an available auth mechanism on the corresponding resource adapter, it is ignored. Default=BASIC PASSWORD
Component-managed Authentication Alias		<input type="checkbox"/> References authentication data for component-managed signon to the resource.
Container-managed Authentication Alias		<input type="checkbox"/> References authentication data for container-managed signon to the resource.
Mapping-Configuration Alias		<input type="checkbox"/> Select a suitable JAAS login configuration from the security-JAAS configuration panel to map the user identity and credentials to a resource principal and credentials that is required to open a connection to the back-end server.

Apply | OK | Reset | Cancel

Figure 9-9 IMS Remote J2C general properties

Important: Component-managed Authentication Alias is used to provide a user ID/password when the application's res-auth deployment descriptor specifies Application and there is no explicit user ID/password provided on the connection request.

Container-managed Authentication Alias is used to provide a user ID/password when the application's res-auth deployment descriptor specifies Container.

We clicked **Apply** to display Additional Properties at the bottom of the panel (Figure 9-9 on page 215).

We clicked **Custom Properties** to display the J2C Custom Properties panel (Figure 9-10). We entered the HostName (wtsc48.itsso.ibm.com), the PortNumber (6001), and the DataStoreName (IM4B), provided by the IMS System Programmer.

Total: 17

Filter

Preferences

Name	Value	Description	Required
IMSConnectName	-	Name of the target IMS Connect - for Local Option only.	false
HostName	wtsc52.itso.ibm.com	TCP/IP host name of the target IMS Connect.	false
PortNumber	6001	Target TCP/IP port number of IMS Connect.	false
DataStoreName	IMS4D	Name of the target IMS datastore.	false
SSL.Enabled	FALSE	Indicates if SSL is enabled for this connection factory.	false
SSLEncryptionType	Weak	The type of cipher suite to be used for encryption.	false
SSLKeyStoreName	-	Name (full path) of SSL keystore for client certificates/private keys.	false
SSLKeyStorePassword	-	Password of SSL keystore for client certificates/private keys.	false
SSLTrustStoreName	-	Name (full path) of SSL keystore for trusted certificates.	false
SSLTrustStorePassword	-	Password of SSL keystore for trusted certificates.	false
UserName	-	Default name of the user to be authorized.	false
Password	-	Default password of the user.	false
GroupName	-	Default name of the IMS group of the user.	false
TraceLevel	1	Level of information to be traced.	false
TransactionResourceRegistration	dynamic	Type of transaction resource registration (enlistment). Valid values are either "static" (immediate) or "dynamic" (deferred).	false
MFSXMLRepositoryID	default	Unique identifier of MFS XML Repository.	false
MFSXMLRepositoryURI	-	Location of MFS XML Repository.	false

Figure 9-10 IMS Remote J2C custom properties

Note the following properties:

- ▶ **HostName** - The TCP/IP host name of the system on which the IMS Connect task is running.
- ▶ **PortNumber** - The port on which the IMS Connect task is listening (Example 9-3 on page 219).
- ▶ **DataStoreName** - The name of the target IMS datastore. It must match the ID parameter of the DataStore statement specified in the IMS Connect configuration member (Example 9-3 on page 219).
- ▶ **IMSConnectName** - Not used for remote J2C connection factories.
- ▶ **UserName** - A default user ID to be used for this connection if no other is provided.
- ▶ **Password** - A password to be used with the above default user ID.
- ▶ **GroupName** - A RACF group to be used with the above user ID/password.
- ▶ **TraceLevel**:
 - 0 - No tracing or logging occurs.
 - 1 - Only errors and exceptions are logged.

- 2 - Errors and exceptions plus the entry and exit of important methods are logged.
- 3 - Errors and exceptions, the entry and exit of important methods, and the contents of buffers sent and received are logged.

Tip: If you need to add additional J2C connection factories you can click **J2C Connection Factories** at the top of the panel to redisplay the list of J2C connection factories configured for this adapter and repeat this process; otherwise save the changes.

Since we had no other J2C connection factories to enter, we saved our configuration changes. Click **Save** on the messages panel at the top of the screen. There will be a message to save to the master configuration. Click **Save**.

9.5 Deploying TraderIMS application in WebSphere Application Server

The JNDI name specified in the application needs to be resolved to a JNDI name from one of our J2C connection factories.

During the installation of the Trader application, there are several panels that are displayed. We took the defaults and clicked **Next** on all but one of the panels, step 3. In step 3, as shown in Figure 9-11 on page 217, we needed to map the resource references to the resources. The panel presented us with a list of JNDI names found in the Trader application. We selected the pull-down that displayed a list of the available J2C connection factories. We selected the factory that we defined previously for our remote connection and the desired Modules and clicked **Apply**.

→ Step 3: Map resource references to resources

Each resource reference defined in your application must be mapped to a resource.

javax.resource.cci.ConnectionFactory

Specify existing Resource JNDI name:

nda11sc48:IMSLocal [Apply]

<input checked="" type="checkbox"/>	Module	EJB	URI	Reference Binding	JNDI Name
<input checked="" type="checkbox"/>	TraderIMS.EJB	TraderService	TraderIMS.EJB.jar;META-INF/ejb-jar.xml	itso/ims/trader/TraderIMSServiceTraderIMSPort	TraderService

Previous Next Cancel

Step 4: Map virtual hosts for web modules

Figure 9-11 Map resource reference to resources

We then continued with the application installation, accepting the defaults, and successfully deployed the application in the WebSphere Application Server.

Tip: The J2C connection factory can be modified later without re-installing the application.

We have now completed the required steps in WebSphere Application Server to deploy the application.

Now that our application has been deployed in the WebSphere Application Server, we need to make sure that the z/OS system has the necessary definitions to allow us to connect to it.

9.6 IMS Connect configuration

Our goal is to describe the parameters and configuration definitions to allow our WebSphere Application Server for Linux for zSeries to connect to IMS on the z/OS machine, and not to describe all of the parameters and configuration options of IMS Connect in this book. For more information on setting up IMS Connect, see *IMS Connect Guide and Reference V2.1, SG18-7260*. Since IMS Connect was already configured on our z/OS system for local communications, the following IMS Connect setup has already been defined:

1. SCHEDxx member of SYS1.PARMLIB

SCHEDxx already had a PPT entry for HWSHWS00, as shown in Example 9-1. Note that if you are using IMS Connect for TCP/IP communications only, you can make the PPT entry SWAP.

Example 9-1 PPT entry for IMS Connect

```
PPT PGMNAME(HWSHWS00) /* PROGRAM NAME = HWSHWS00 */
CANCEL /* PROGRAM CAN BE CANCELED */
KEY(7) /* PROTECT KEY ASSIGNED IS 7 */
NOSWAP /* PROGRAM IS NOT SWAPPABLE */
NOPRIV /* PROGRAM IS NOT PRIVILEGED */
DSI /* REQUIRES DATA SET INTEGRITY */
PASS /* CANNOT BYPASS PASSWORD PROTECTION */
SYST /* PROGRAM IS A SYSTEM TASK */
AFF(NONE) /* NO CPU AFFINITY */
NOPREF /* NO PREFERRED STORAGE FRAMES */
```

If you are just adding this member, you will need to make the changes effective by either re-IPLing MVS or issuing the SETSCH= command.

2. IMS Connect PROCLIB member

A proclib member already existed, which is shown in Example 9-2. The IMS Connect configuration is specified by the HWSCFG parameter, shown in bold.

Example 9-2 IMS Connect Proclib member

```
//HWS PROC RGN=4096K,SOUT=A,
// BPECFG=BPECFGHT,
// HWSCFG=HWSCFG00
//*
//*****
/* BRING UP AN IMS CONNECT *
//*****
//STEP1 EXEC PGM=HWSHWS00,REGION=&RGN,TIME=1440,
// PARM=IBPECFG=&BPECFG,HWSCFG=&HWSCFG
//STEPLIB DD DSN=SHWSRESL,DISP=SHR
// DD DSN=SDFSRESL,DISP=SHR
// DD DSN=CEE.SCEERUN,UNIT=SYSDA,DISP=SHR
// DD DSN=SYS1.CSSLIB,UNIT=SYSDA,DISP=SHR
// DD DSN=GSK.SGSKLOAD,UNIT=SYSDA,DISP=SHR
//PROCLIB DD DSN=USER.PROCLIB,DISP=SHR
//SYSPRINT DD SYSOUT=&SOUT
//SYSUDUMP DD SYSOUT=&SOUT
//HWSRCORD DD DSN=HWSRCDR,DISP=SHR
```

3. The IMS Connect configuration parameter

The IMS Connect configuration parameters are shown in Example 9-3. From the member that is pointed to by the HWSCFG parameter in the IMS Connect procedure, we needed to take note of two keyword parameters: DATASTORE ID and PORTID. These were the values that we specified for our IMS resource adapter on the J2C customer properties panel in Figure 9-10 on page 216.

Example 9-3 IMS Connect configuration parameters

```
HWS (ID=IM4BCONN,RRS=Y,RACF=Y
TCPIP (HOSTNAME=TCPIP,PORTID=(6001,LOCAL),MAXSOC=2000)
DATASTORE (ID=IM4B,GROUP=HAOTMA,MEMBER=HWS814B,TMEMBER=SCSIMS4B)
```

The DATASTORE ID provides the custom property DataStoreName for our IMS J2C connection factory.

The TCPIP PORTID provides the custom property portNumber for our remote J2C connection factory.

Now that we have verified our definitions, it is time to test the application.

9.7 Testing TraderIMS application

We can test our TraderIMS application by using the following HTML:

<http://myserver/TraderIMSWeb/>



Figure 9-12 The ITSO Trader application using the IMS Connector

Type a user ID, for example linux1, and click **Go**.

9.8 Problem determination

To perform problem determination (Figure 9-13) we recommend enabling the following WebSphere traces and setting the trace level in the J2C connection factory customer property to 3:

- ▶ com.ibm.ejs.j2c.*=all=enabled
- ▶ com.ibm.connector2.*=all=enabled

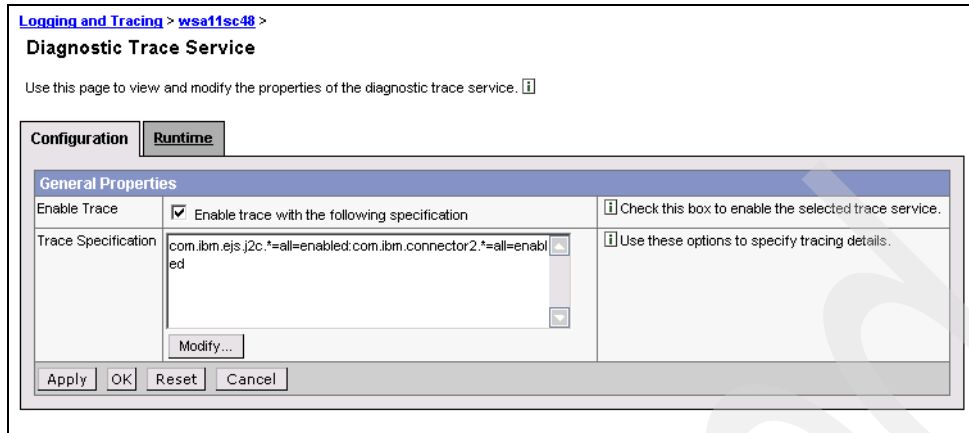


Figure 9-13 Diagnostic traces

9.8.1 Common errors

The following are some common errors:

- ▶ ICO0079E:com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@3b1bf125.getOutputStream (InteractionSpec) error. IMS returned DFS™ message: DFS064 08:31:04 DESTINATION CAN NOT BE FOUND OR CREATED.
 - Explanation: The first eight characters of the input could not be recognized as a valid transaction, logical terminal name, or command.
 - Usual cause: The transaction name specified in the input request is not recognized by the target IMS system.
- ▶ ICO0001E:com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@d6fd946.processOutputOTMAMsg(byte [], InteractionSpec, Record) error. IMS Connect returned the error: RETCODE=[8], REASONCODE=[SECFNPU]. Security failure; no password and no user ID.
 - Explanation: The application descriptor specifies the res-auth Application but the application did not provide a user ID or password.
 - Solution: Consider changing the res-auth to Container.
- ▶ ICO0003E:com.ibm.connector2.ims.ico.IMSTCPIPManagedConnection@5072da31.connect() error. Failed to connect to host [p390.raleigh.ibm.com], port [4000]. [java.net.ConnectException: Connection refused: connect]
 - Explanation: The IMS Connect task on the specified host is not accepting the connection request.
 - Solution: Verify that the IMS Connect task is active on the target host system and is listening on the specified port.

- ▶ ICO0064E:com.ibm.connector2.ims.ico.IMSLocalOptionManagedConnection@12d18e56.processSubject(javax.security.auth.Subject aSubject) error. Invalid security credential.
 - Explanation: IMS connect was unable to validate the user ID and password passed on the request with the external security manager.
 - Solution: An invalid user ID and/or password was provided.

9.9 Thread identity support

IMS J2C connection factories support the assignment of a thread identifier as the owner of a connection for authentication purposes. This assignment is enabled when the following conditions are met:

- ▶ Contained Managed resource authority (res-auth=Container) is specified in the application deployment descriptors.
- ▶ The J2C Connection factory uses local (no TCP/IP) access, and no Container-managed Authentication Alias is specified.

VSE Java-based connector to access VSAM data

The VSE Java-based connector can be used to access all kinds of VSE data such as VSAM, DL/I, POWER™, Librarian, and ICCF. Java programs such as applets, servlets, or stand-alone programs can also use this connector to access console or job submission functions. In this chapter, we describe how to access VSE/VSAM data from a WebSphere environment.

We discuss the following topics:

- ▶ VSE Java-based connector overview
- ▶ Installing the VSE Java-based connector
- ▶ Using the VSE connector client as a resource adapter
- ▶ Using the VSE connector client as a JDBC provider
- ▶ Setting up sample data for Trader
- ▶ Installing an application in WebSphere
- ▶ Configuring for SSL secure connections
- ▶ Problem determination

10.1 VSE Java-based connectors overview

The VSE Java-based connectors consist of a client part and a server part, as shown in Figure 10-1. The VSE Connector Client (VCC) includes a Java class library of JavaBeans. These beans represent VSE system components and file systems, such as VSAM, POWER and the operator console. There is extensive online documentation and ready-to-run samples. The server part is the VSE Connector Server (VCS), a batch application that receives requests from the VSE JavaBeans and uses native methods to access VSE data and functions.

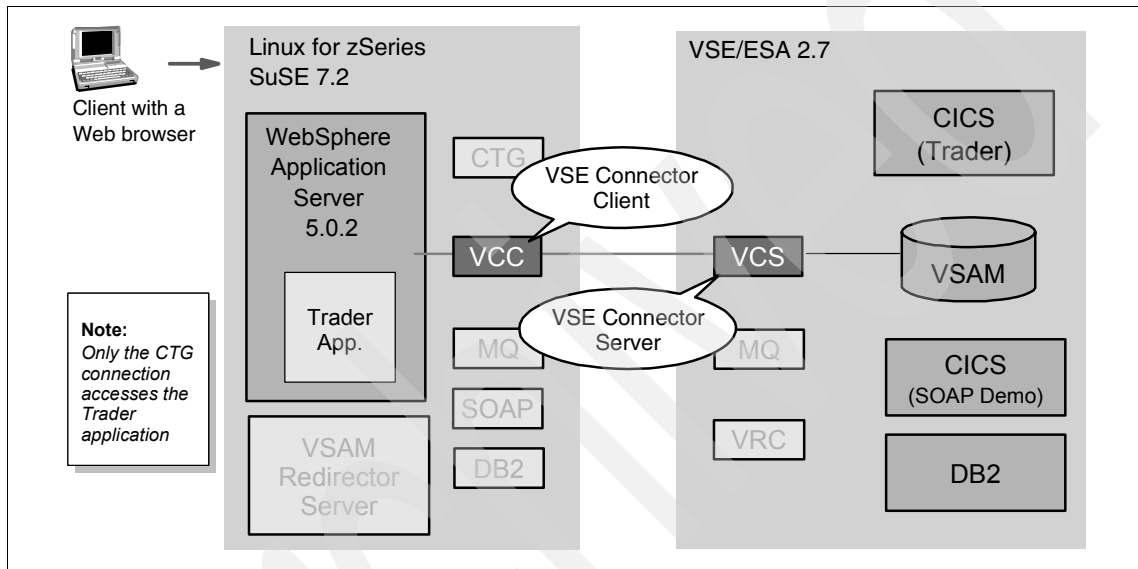


Figure 10-1 VSE Java-based connector overview

There is a number of other connectors, for example the VSE Script connector, which allows access to VSE data from non-Java programs. It uses a proxy server that executes scripts. This connector can be used to access VSE from spreadsheet applications, like MS Excel or Lotus® 1-2-3® via VisualBasic. There is also a variety of tools around these connectors, for example, to print host data formatted on LAN-attached printers or to set up SSL.

A detailed description can be found in the book *VSE/ESA e-business Connectors User's Guide*, SC33-6719.

All connector components, online books, and documentation can be downloaded from the following Web site:

<http://www-1.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/>

10.1.1 Client-server components

The VSE Java-based connector consists of a client and server components.

- ▶ The VSE Connector Client (VCC) includes:
 - The VSE JavaBeans class library (VSEConnector.jar). This includes a VSAM JDBC driver.
 - The related Java documentation.
 - Online documentation in HTML format, including details about using the CICS SOAP support.
 - Samples for all kinds of data access.
- ▶ The VSE Connector Server (VCS) is a LE/VSE C-application. It runs on VSE, and uses native methods to access VSE data and functions.

Both components talk to each other via TCP/IP using a proprietary application protocol.

Important: Make sure you have the most recent level of both components.

The VSE JavaBeans class library can be deployed to WebSphere as a resource adapter. Also, JDBC requests are possible against VSAM data. So the primary use of this connector in our project was to access VSAM data via a servlet.

Figure 10-2 shows a typical scenario with a servlet accessing VSE/VSAM data.

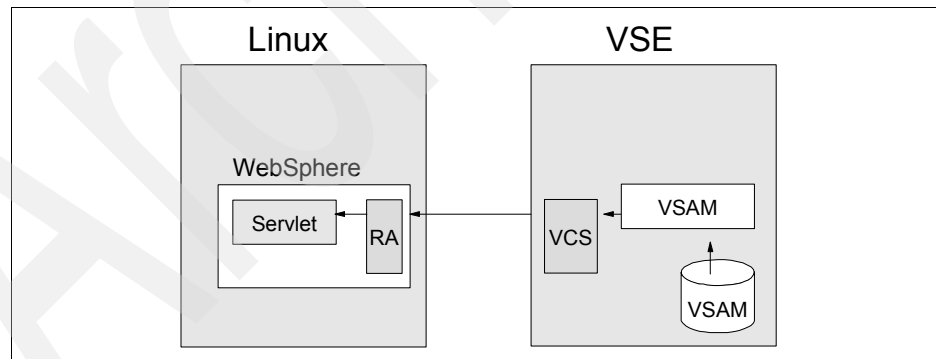


Figure 10-2 Access VSAM data from a servlet

The servlet uses the VSE-provided resource adapter (RA) to connect to the VSE Connector Server (VCS) running on VSE. The connector server accesses the VSAM data and sends it back to the resource adapter, which in turn delivers it to the servlet.

10.2 Installing the VSE Java-based connector

The installation of the VSE Java-based connector requires both client and server implementation.

10.2.1 Client

The VSE Connector Client is shipped with VSE and can be downloaded from either of the following:

- ▶ PRD1.BASE, member IESINCON.W

Download the member in binary, rename the file extension to .zip, and open it with any zip tool, like winzip or pkzip. There are four files contained in the zip file:

- install.class - Contains the product code and install wizard
- install.bat - Install batch file for Windows
- install.cmd - Install batch file for OS/2 or Windows NT®
- install.sh - Install batch file for Unix/Linux

- ▶ The Internet

<http://www-1.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/>

Note: The installation process is based on Java and needs JDK 1.3 or later on your workstation. We recommend that you use JDK 1.4, because some VSE tools, which are based on the connector class library, require JDK 1.4.

Running the appropriate installation batch file on your workstation will launch an install wizard, which prompts you for some install options.

Tip: When installing on Windows, the installation process adds some environment variables, which are referenced by connector-based tools. These new variables become active when rebooting your workstation. But you can get them active without reboot, by clicking **Control Panel** → **System** → **Environment Variables** and clicking **OK** on this dialog box.

To create a WebSphere resource adapter, we need the following jar files, which are located in the connector client's install directory:

- ▶ VSEConnector.jar, the VSE JavaBeans class library
- ▶ cci.jar, the J2C Common Connector Interface definitions
- ▶ ibmjsse.jar, the SSL-related classes

10.2.2 Server

The VSE Connector Server is configured via a set of job skeletons, which are provided in ICCF library 59.

SKVCSCFG	The main config member.
SKVCSLIB	Define list of VSE libraries.
SKVCSPLG	Define server plugins.
SKVCSSSL	Define SSL parameters.
SKVCSSTJ	A template for the server startup job.
SKVCSUSR	Configure users and IP addresses.
SKVCSCAT	To catalog the config members.

At this point in time we did not change anything here. The server is started with the job STARTVCS, which is placed in the reader queue when installing VSE. The server runs in dynamic class R by default.

10.3 Using VSE Connector Client as resource adapter

This section describes how to use the VSE Connector Client as a WebSphere resource adapter.

A resource adapter defines the access to a specific Enterprise Information System (EIS), like VSE, z/OS, or other vendor-specific back-end systems. This allows WebSphere applications to just use resource adapters to connect to remote platforms without the need to know about IP addresses, user IDs, passwords, and so on. All system- and connection-specific information is encapsulated by the resource adapter.

A resource adapter is given by a .rar file, which in fact is a .jar file that contains special classes implementing an interface according to the J2EE connector specification. The VSEConnector.jar file implements the Common Connector Interface (CCI). In addition to that, the .rar file must contain a deployment descriptor file, which describes the properties of the related back-end system. This is an XML file called ra.xml in directory META-INF. This file defines parameters needed to access the back-end system.

The following section shows the steps for deploying the VSE Connector Client class library as a WebSphere resource adapter.

10.3.1 Defining a resource adapter

Copy the VSEConnector.jar file to VSEConnector.rar in order to deploy it. Do not rename the original jar file, because in this case other connector-based

applications would no longer be able to access it. You have to accomplish the following steps:

1. Log on to the WebSphere admin console.
2. Select **Resources** and click **Resource Adapters**.

Click **Install RAR** to install the VSEConnector.rar file as a new resource adapter. The file will be extracted into a WebSphere directory.

3. Enter the path name of the VSEConnector.rar file and click **Next**.

The screenshot shows a dialog box for installing a Resource Adapter (RAR). It has two main sections: 'Path' and 'Scope'. In the 'Path' section, there are two radio buttons: 'Local path' (unselected) and 'Server path' (selected). Below 'Local path' is a text field and a 'Browse...' button. Below 'Server path' is a text field containing the path '/opt/vse/VSEConnector.rar'. In the 'Scope' section, there is a 'Node:' label followed by a dropdown menu showing 'linux11'. At the bottom of the dialog, there are two buttons: 'Next' and 'Cancel'. A mouse cursor is pointing at the 'Next' button.

Enter the name and description of the new resource adapter. Also enter the place-name of the two additional VSE Connector jar-files in the field Classpath. Leave the other fields blank.

The screenshot shows the 'General Properties' dialog box for the 'VSEConnector' resource adapter. It has a table-like structure with the following fields:

General Properties		
Scope	* cells:linux11:nodes:linux11	<i>i</i> The scope of the configured...
Name	VSEConnector	<i>i</i> The name of the resource ad used.
Description	VSE Resource Adapter	<i>i</i> A text description for the res...
Archive Path		<i>i</i> Path to the installed RAR file o extracted to the absolute path re variable defined in variables.xml
Classpath	/opt/vse/cci.jar /opt/vse/lbmjsse.jar	<i>i</i> A list of paths or JAR file nam by using the ENTER key and mus names which can be substituted required.

Click **OK**.

4. Click the new resource adapter to specify further properties.

<input type="checkbox"/>	Name ▾
<input type="checkbox"/>	VSEConnector
	WebSphere Relational Resource Adapter

- Click **J2C Connection Factories** to create a new connection factory. While a resource adapter defines the properties of a specific application on a back-end system as a whole, like MQ on z/OS or the VSE Connector Server on VSE, a connection factory defines the properties of one specific connection to one single back-end system.

Additional Properties	
J2C Connection Factories	J2C Connection Factories
Custom Properties	Properties that may be required for data
View Deployment Descriptor	View the Deployment Des

- Click **New** to create a new connection factory. Then enter the properties of the connection factory.

General Properties	
Scope	* cells:linux11:nodes:linux11
Name	* VSE Connection Factory
JNDI name	eis/VSEConnector

You may leave the other fields blank.

Note: You can freely choose the JNDI name here. However, a naming convention in a production environment would call it, for example, eis/vse270 to indicate a specific VSE back-end system. Further connection factories, which go to other VSE systems, would similarly include their names into their JNDI names.

Then press **OK**.

- Scroll down and click **Custom Properties** where you should see the VSE-specific parameters as defined in the ra.xml file in VSEConnector.rar.

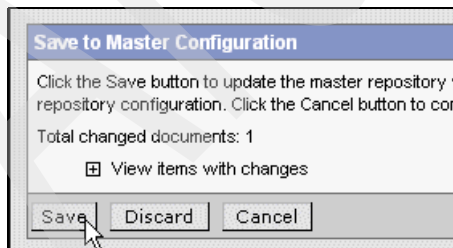
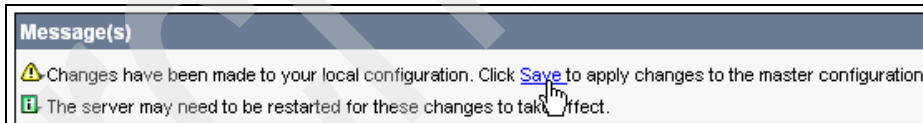
Additional Properties	
Connection Pool	An optional set of connection p
Custom Properties	Properties that may be required vendors require additional cust

Here we specified the settings according to our VSE system.

Name ▾	Value ▾
ServerName	9.152.82.82
PortNumber	2893
UserName	JSCH
Password	MYPASSW
SSLVersion	OFF
CipherSuites	SSL_RSA_WITH_DES_CBC_SHA,SSL_RSA_WITH_3DES_EDE_CBC_SHA,SSL
KeypingFile	Keyping.pfx
KeypingPassword	ssitest

At this moment we do not use SSL. So we only need the IP address or symbolic name of the VSE system, a valid VSE, user ID and its password. The port 2893 is the default port of the VSE Connector Server running on VSE.

8. Save your changes.



The next section shows you how to reference this resource adapter in a servlet.

10.3.2 Related servlet code

The following code snippet is taken from the `doGet()` method of a servlet and shows how to connect to VSE using the defined resource adapter.

Example 10-1 Access VSE data and functions from a servlet

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
    ...
    VSEConnectionSpec spec;
    VSESystem system;
    Context ctx;

    try
    {
        ctx = new InitialContext();
        spec = new VSEConnectionSpec(ctx, "eis/VSEConnector");
        system = new VSESystem(spec);
        system.connect(); (1)
    }
    catch (NamingException e)
    {
        ...
    }
    ...
}
```

Calling the `connect()` method is not necessary in an application. The connection is established implicitly when needed. But the `connect()` method can be used to check the connection.

The next chapter shows how to define a JDBC provider using the VSE Connector Client class library.

10.4 Using the VSE connector client as a JDBC provider

In the VSE connector implementation, the JDBC driver is contained in the same jar file (`VSEConnector.jar`) as the VSE resource adapter. In general, this need not to be the case. Figure 10-3 shows a JDBC data source that connects to the VSE Connector Server (VCS). The VSE resource adapter is not used here.

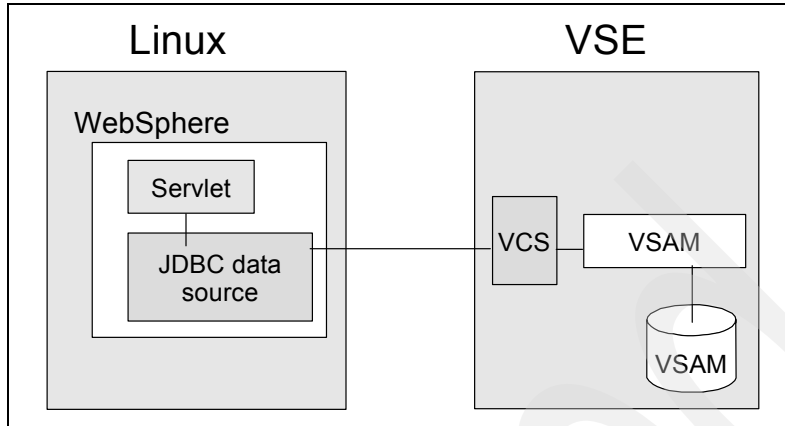


Figure 10-3 Access VSAM data via JDBC

Figure 10-3 shows a servlet that uses the VSAM JDBC provider, contained in the VSE resource adapter, to access VSAM data on VSE in a relational way.

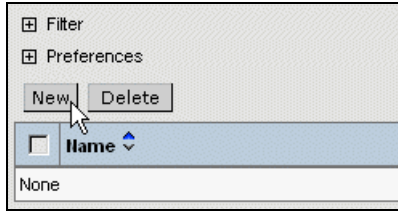
10.4.1 Defining a JDBC provider

To define a VSAM JDBC provider to WebSphere, you use the same class files as for the above described resource adapter. However, the JNDI name is different. Proceed as follows.

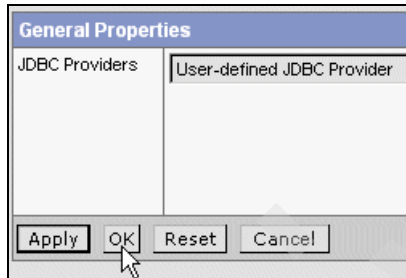
1. Log on to the admin console by entering any user ID. Click **Resources** → **JDBC Providers**.



2. Click **New** to add a new JDBC provider.



3. Create a user-defined JDBC provider.



4. Enter the properties for the JDBC provider. In the field Classpath enter the three VSE Connector jar files. Remove any other entries.

General Properties	
Scope	* cells:linux11:nodes:linux11
Name	* VSAMJDBC
Description	VSE VSAM JDBC Provider
Classpath	/opt/vse/VSEConnector.jar /opt/vse/ibmjssc.jar /opt/vse/ccj.jar
Native Library Path	
Implementation Classname	* com.ibm.vse.jdbc.VsamJdbcConnect
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>	

In the field Implementation Classname enter:

`com.ibm.vse.jdbc.VsamJdbcConnectionPoolDataSource`

It must be exactly this string, because it references the Java class, which implements the JDBC data source. Then click **OK**.

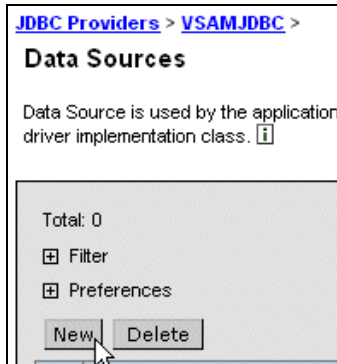
- Specify further properties. Click the JDBC provider name.

<input type="checkbox"/> Name ▾	Description ▾
<input type="checkbox"/> DB2 JDBC Provider (XA)	DB2 JDBC Provider
<input type="checkbox"/> VSAMJDBC	VSE VSAM JDBC Provider

- Click **Data Sources**.

Additional Properties	
Data Sources	Data Source is used by the e which provides the specific
Data Sources (Version 4)	This is the WebSphere 4.x d must use this data source.

- Click **New** to create a new data source for this JDBC provider.



8. Specify the data source properties.

General Properties	
Scope	* cells:linux11:nodes:linux11
Name	* VSAM JDBC Data Source
JNDI Name	jdbc/vsamjdbc
Container managed persistence	<input type="checkbox"/> Use this Data Source in container managed persistence (CMP)
Description	Data Source for VSAM JDBC Driver

Click **OK**.

Note: You can freely choose the JNDI name here. However, a naming convention in a production environment would call it, for example, jdbc/vsam/vse270 to indicate a specific VSE back-end system. Further JDBC data sources, which go to other VSE systems, would similarly include their names into their JNDI names.

9. Specify further properties of the data source. Click the data source name.

<input type="checkbox"/>	Name ▾	JNDI Name ▾	Description ▾
<input type="checkbox"/>	VSAM JDBC Data Source	jdbc/vsamjdbc	Data Source for VSAM JDBC Driver

10. Click **J2C Data Authentication Entries**.

Related Items	
J2C Authentication Data Entries	Specifies a list of userid and password

11. Add a new entry specifying your VSE user ID and password.

General Properties	
Alias	* userName
User ID	* JSCH
Password	* *****
Description	VSE user ID
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>	

Go back to the data source properties and select this newly created authentication entry from check box Component-managed Authentication Alias. This VSE user ID is used by the JDBC driver to connect to VSE. If it is not specified, the VSE Connector Server would reject the connection later, because JDBC would try to connect without sending a user ID.

General Properties	
Scope	* cells:linux11:nodes:linux11
Name	* VSAM JDBC Data Source
JNDI Name	jdbc/vsamjdbc
Container managed persistence	<input type="checkbox"/> Use this Data Source in container managed persistence (CMP)
Description	Datasource for VSAM JDBC Driver
Category	
Statement Cache Size	10 statements
Datasource Helper Classname	com.ibm.websphere.rsadapter.General
Component-managed Authentication Alias	linux11/username
Container-managed Authentication Alias	

Click **Apply**.

12. Scroll down and click **Custom Properties** to define database-specific properties needed to access VSE.

Additional Properties	
Connection Pool	An optional set of connection pool
Custom Properties	Properties that may be required for data sources that

13. Add the VSE server name. Leave the type field as String.

General Properties	
Scope	* cells:linux11:nodes:linux11
Required	false
Name	* serverName
Value	9.152.82.82
Description	VSE Backend
Type	java.lang.String
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>	

14. Add the VSE Connector Server port number.

Important: It is important that you select type **Integer** from the drop-down combo box. Otherwise the connection will not work, because the JDBC driver expects this parameter as an integer value.

General Properties	
Scope	* cells:linux11:nodes:linux11
Required	false
Name	* port Number
Value	2893
Description	Port to VSE Connector Server
Type	java.lang.Integer
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>	

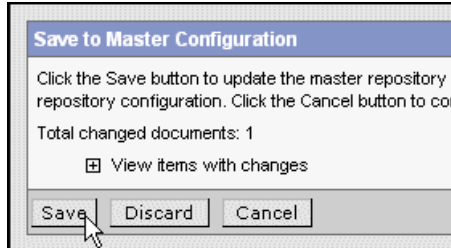
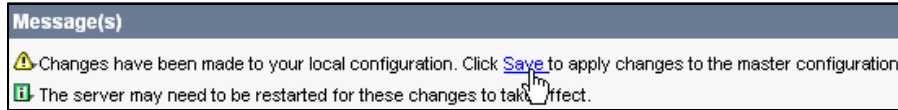
15. Indicate to not use SSL by specifying SSL=OFF, which is the default. For details about how to define an SSL connection, see 10.7, “Configuring for SSL secure connections” on page 262.

General Properties	
Scope	* cells:linux11:nodes:linux11
Required	false
Name	* SSLVersion
Value	OFF
Description	SSL version (OFF, SSL, TLS)
Type	java.lang.String
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>	

16. Now you should have a list similar to the following figure.

<input type="checkbox"/>	Name	Value	Description
<input type="checkbox"/>	SSLVersion	OFF	SSL version (OFF, SSL, TLS)
<input type="checkbox"/>	portNumber	2893	Port of VSE Connector Server
<input type="checkbox"/>	serverName	9.152.82.82	VSE Backend

17. Save your changes and restart the WebSphere service.



For details on defining a VSAM JDBC data source for SSL connections, refer to 10.7.12, “Configuring an SSL JDBC data source” on page 280.

The following section shows how to use this JDBC driver from a servlet to access VSAM data.

10.4.2 Related servlet code

The following code snippets can be used in a servlet to retrieve VSAM data via JDBC. First, you have to look up the JDBC data source via its JNDI name.

Example 10-2 Look up JDBC data source

```
try
{
    // Lookup a JDBC Connection
    Context ctx = new InitialContext();
    DataSource ds = (DataSource)ctx.lookup("jdbc/vsamjdbc");
    java.sql.Connection jdbcCon = ds.getConnection();
}
catch (Throwable t)
{
    ...
}
```

The next step is to retrieve the data records. The SQL query uses the concatenated file IDs of the VSAM catalog and cluster, separated by a backslash, and followed by the VSAM map name. The data fields are retrieved by specifying their VSAM field names, which are defined in the VSAM map.

```
try
{
    // Create the statement
    java.sql.Statement stmt = jdbcCon.createStatement();

    // Submit the query
    java.sql.ResultSet rs = stmt.executeQuery(
        "SELECT * FROM VSESP.USER.CATALOG\\TRADER.COMPANY\\COMPANY_MAP");

    // Walk through the results
    while (rs.next())
    {
        String company = rs.getString("COMPANY");
        int share_price = rs.getInt("SHARE_PRICE");
        int unit_val_7 = rs.getInt("UNIT_VALUE_7DAYS");
        ...
    }
    rs.close();
    stmt.close();
}
...
```

10.5 Setting up sample data for Trader

In order to access non-relational VSAM data in a relational way, for example via JDBC, the internal record structure must be made known externally. Typically, the data fields of a given VSAM record are only known by the application program (for example COBOL or PL/I). Even VSAM has no information about the internal structure of a record, except of the length and position of key fields.

Now there are a couple of ways to make this record structure, here called VSAM map, available to any kind of VSE connector-based Java program:

- ▶ Extract the record layout from a given COBOL copy book.
- ▶ Import/export the record structure from an XML file.
- ▶ Get a given VSAM map from a VSE system.
- ▶ Define a map by hand.

VSE provides three functions that support VSAM data mapping:

- ▶ The VSE Navigator
- ▶ The VSAM map tool
- ▶ The IDCAMS command RECMAP

To create the maps for Trader, we used the VSE Navigator, because we did not have any COBOL copy books or other files from which we could create the maps on VSE automatically.

10.5.1 Installing the VSE Navigator

The VSE Navigator is a Java-based system management tool that provides a graphical user interface for VSE. The Java client is implemented on the basis of the VSE Connector Client and thus connects to the VSE Connector Server on VSE via TCP/IP.

The tool is not shipped as part of VSE. It can be downloaded from:

<http://www-1.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/>

To install the VSE Navigator:

1. Download the Navigator zip file in binary and extract the containing files into an empty directory. There are four files contained in the zip file:
 - install.class - Contains the product code and install wizard.
 - install.bat - Install batch file for Windows.
 - install.cmd - Install batch file for OS/2 or Windows NT.
 - install.sh - Install batch file for Unix/Linux.
2. Run the installation batch file appropriate for your platform and follow the install wizard dialogs.

10.5.2 Installing the VSAM maptool

The VSAM maptool allows you to maintain VSAM maps by providing the following basic functions:

- ▶ Extract a map from a given COBOL copy book.
- ▶ Import (receive) a given map from a given VSE system.
- ▶ Export a map to a VSE system (send it to VSE).
- ▶ Import a map from a XML file.
- ▶ Export a map to a XML file.

So it is especially interesting when you have big data structures with lots of fields, which would make it too time consuming for generating maps manually.

The VSAM maptool can be downloaded from:

<http://www-1.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/>

It is not shipped as part of VSE. To install the maptool:

1. Download the zip file and extract its contents in a new empty directory.

2. Start the tool by running the run.bat / run.sh file.

10.5.3 Creating the sample VSAM files

The ITSO trader CICS application works on two VSAM files that we defined on VSE as follows. The first file represents the trader.company table.

Example 10-4 Definition of the trader.company file

```
* $$ JOB JNM=TRCOMP,CLASS=0,DISP=D,NTFY=YES
// JOB JSCH DEFINE FILE
// EXEC IDCAMS,SIZE=AUTO
DEFINE CLUSTER ( -
    NAME (TRADER.COMPANY ) -
    CYLINDERS(2 2 ) -
    SHAREOPTIONS (1) -
    RECORDSIZE (80 80 ) -
    VOLUMES (DOSRES SYSWK1 ) -
    NOREUSE -
    INDEXED -
    FREESPACE (15 7) -
    KEYS (20 0 ) -
    NOCOMPRESSED -
    TO (99366 ) -
    DATA (NAME (TRADER.COMPANY.@D@ ) -
    CONTROLINTERVALSIZE (4096 ) -
    INDEX (NAME (TRADER.COMPANY.@I@ )) -
    CATALOG (VSESP.USER.CATALOG )
    IF LASTCC NE 0 THEN CANCEL JOB
/*
// OPTION STDLABEL=ADD
// DLBL TRCOMPA,'TRADER.COMPANY',,VSAM, X
    CAT=VSESPUC
/*
// EXEC IESVCLUP,SIZE=AUTO
A TRADER.COMPANY TRCOMPA VSESPUC
/*
/&
* $$ EOJ
```

The second file represents the trader.customer table. It will initially contain no data.

Example 10-5 Definition of the trader.customer file

```
* $$ JOB JNM=TRCUST,CLASS=0,DISP=D,NTFY=YES
// JOB JSCH DEFINE FILE
// EXEC IDCAMS,SIZE=AUTO
```

```

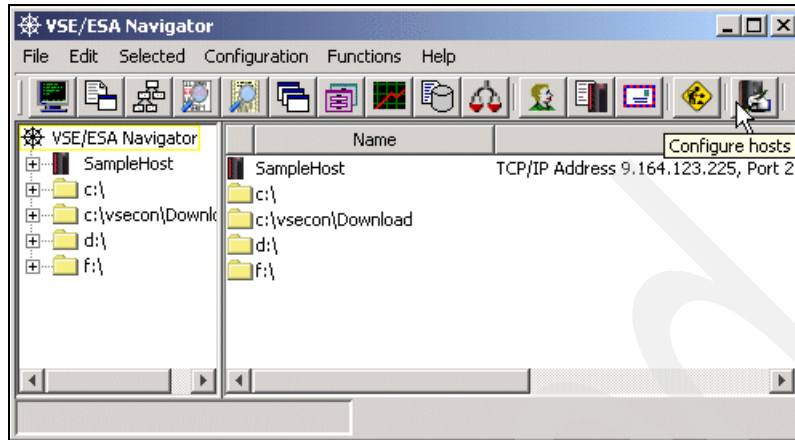
DEFINE CLUSTER ( -
    NAME (TRADER.CUSTOMER                                ) -
    CYLINDERS(2      2                                ) -
    SHAREOPTIONS (1) -
    RECORDSIZE (90    90    ) -
    VOLUMES (DOSRES SYSWK1 ) -
    NOREUSE -
    INDEXED -
    FREESPACE (15 7) -
    KEYS (80 0    ) -
    NOCOMPRESSED -
    TO (99366  ) -
    DATA (NAME (TRADER.CUSTOMER.@D@                    ) -
    CONTROLINTERVALSIZE (4096  )) -
    INDEX (NAME (TRADER.CUSTOMER.@I@                    )) -
    CATALOG (VSESP.USER.CATALOG                          )
IF LASTCC NE 0 THEN CANCEL JOB
/*
// OPTION STDLABEL=ADD
// DLBL TRCUSTO,'TRADER.CUSTOMER',,VSAM,                X
    CAT=VSESPUC
/*
// EXEC IESVCLUP,SIZE=AUTO A TRADER.CUSTOMER
TRCUSTO VSESPUC
/*
/&
* $$ E0J

```

10.5.4 Creating the VSAM maps

Start the VSE Navigator and define your VSE host.

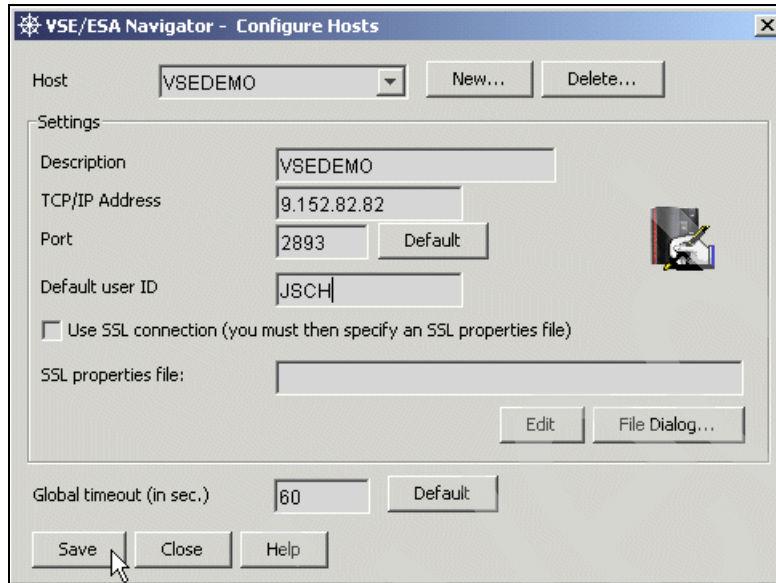
1. Click the **Configure Hosts** tool bar button.



2. Click **New** and enter the name of your host. The name must not contain spaces, slashes, or backslashes.

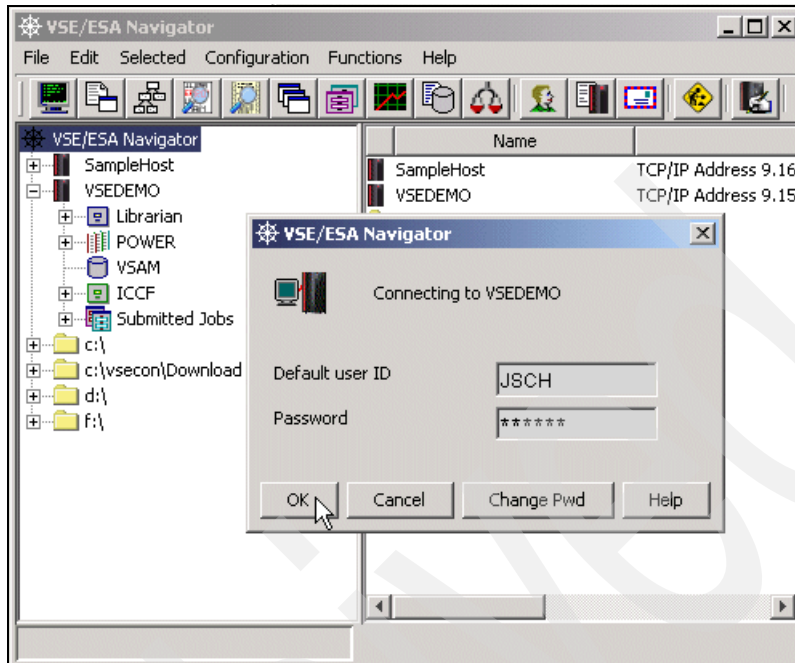


3. Enter your VSE IP address and your default VSE user ID. We recommend that you use an administrator type user ID, like SYSA. Then save and **Close**.

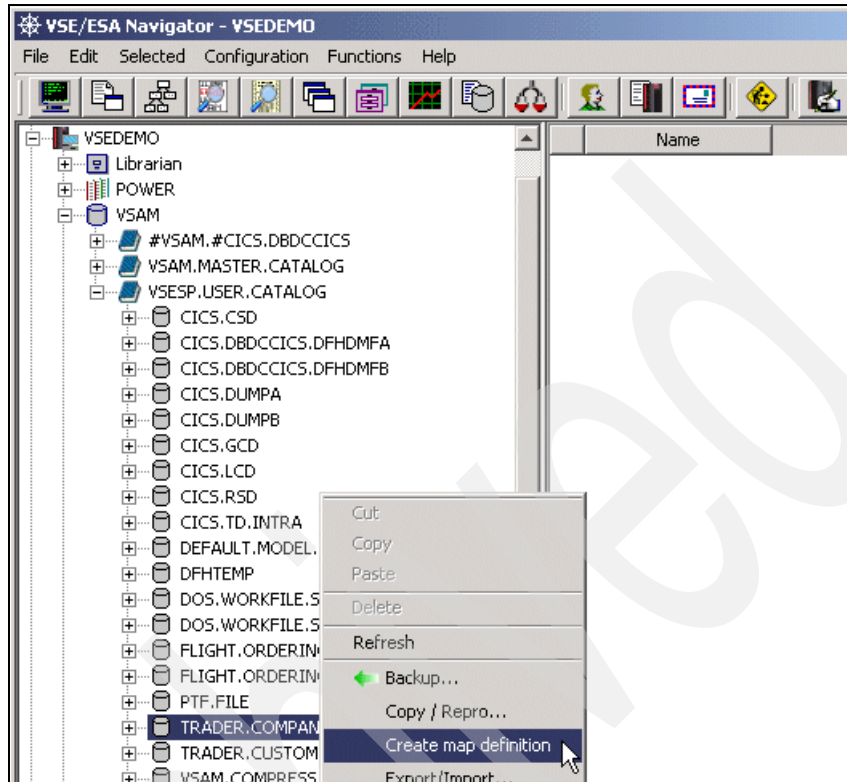


We will not use SSL at this time, so just leave the other controls untouched.

4. Now connect to the VSE system by expanding the VSE host tree node and then opening the VSAM tree node. You are prompted to enter your password.



- Now proceed to the two VSAM files by expanding the VSAM user catalog tree node. Then right-click the TRADER.COMPANY file and select menu choice **Create map definition**.



6. Enter the name of the map, for example COMPANY_MAP.
7. Then define the data fields as required by the DB2 table layout for trader.company. Here is the DDL to define the table to DB2.

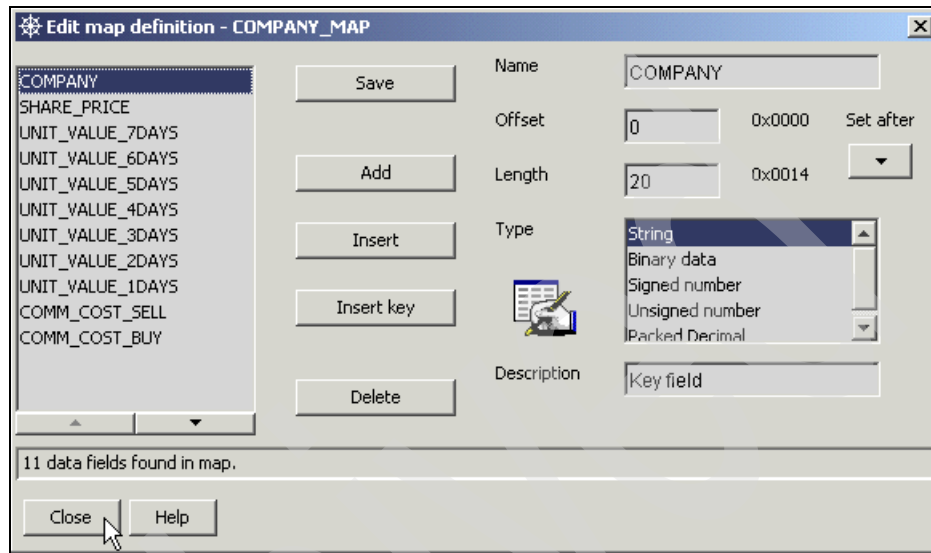
Example 10-6 Data layout for trader.company

```

create table trader.company (
  company char(20) not null primary key,
  share_price real,
  unit_value_7days real,
  unit_value_6days real,
  unit_value_5days real,
  unit_value_4days real,
  unit_value_3days real,
  unit_value_2days real,
  unit_value_1days real,
  comm_cost_sell int,
  comm_cost_buy int ) ;

```

The VSAM map definition looks like that shown in the following picture. For each data field you have to specify the name, offset, length, and data type. All real values are defined here as unsigned numbers.



10.5.5 Populating the sample VSAM files

For the Trader scenario, we have to populate only the trader.company file. The trader.customer file is initially left empty.

There are a number of possibilities to put data into a VSAM file:

- ▶ With the old way using DITTO
- ▶ Using an application, like a COBOL program
- ▶ Using a REPRO job that uses SYSIPT to read the data from the job
- ▶ Using a simple Java program

Typically you would use the first two ways of filling a VSAM file with data when all data fields are strings. If they are not, you may have some trouble changing the VSAM records later in order to have the correct binary data in it.

In our case, only the first field (COMPANY) is a string field; all others are integers. So we just wrote a simple Java program using the VSE JavaBeans to add the data with the correct data types in the first run. For an API documentation refer to the *VSE Connector Client*.

Example 10-7 Populate VSAM files with a simple Java program

```
String catalogID = "VSESP.USER.CATALOG";
```

```

String clusterID = "TRADER.COMPANY";
String mapname   = "COMPANY_MAP";

/* Setup data ... */
String[][] data = {
    {"Casey_Import_Export ", "0079", "0059", "0063", "0065", "0070", "0072", ...},
    {"Glass_and_Luget_Pl c ", "0019", "0017", "0022", "0020", "0016", "0020", ...},
    {"Headworth_Electrical", "0124", "0141", "0138", "0137", "0133", "0133", ...},
    {"IBM                    ", "0163", "0157", "0156", "0159", "0161", "0160", ...}
};

/* Add data ... */
VSEVsamRecord record = new VSEVsamRecord(system, catalogID,
                                           clusterID, mapname);

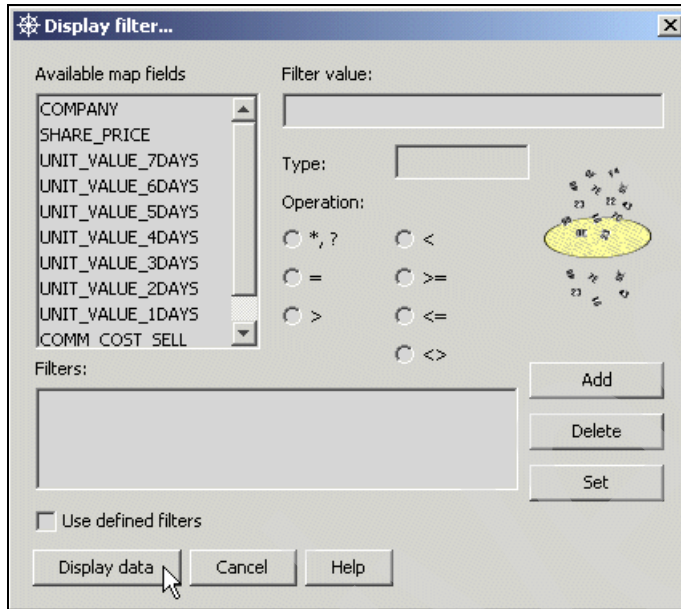
for (int i=0;i<data.length;i++)
{
    record.setKeyField(0, new String(data[i][0]));
    for (int k=1;k<11;k++)
        record.setField(k, new Integer((data[i][k])));
    record.add();
}

```

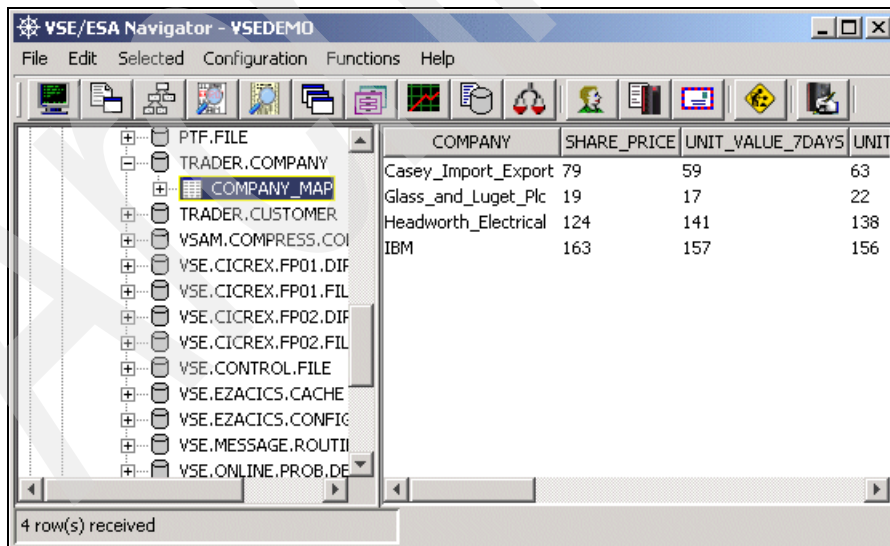
The advantage of such a Java program is that all data is transferred field by field using the correct data type, as specified by the underlying VSAM map. No manual post processing of the data is necessary afterwards.

The full source of the program is contained in “Sample Java program to populate VSAM files on VSE” on page 322.

After populating the VSAM file, you can view, and optionally change, the data via right-clicking the map and selecting **Display VSAM data**. A filter dialog box allows you to select only a subset of all records in the file.



Just click **Display data** to see all records. The display should then look like the following figure.

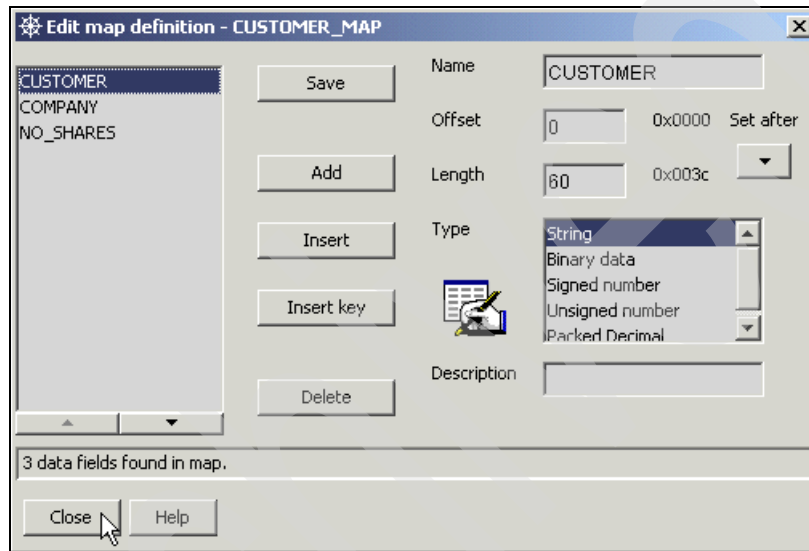


Now define the VSAM map for the trader.customer file. The DDL looks like the following figure.

Example 10-8 Data layout for trader.customer

```
create table trader.customer (  
    customer char(60) not null,  
    company char(20) not null,  
    no_shares int,  
    primary key (customer, company));
```

The map definition will look like the following figure.



As said before, this file will be initially empty, so there is no step to populate it.

10.6 Installing an application in WebSphere

There are several ways to create an application for WebSphere:

- ▶ Using the Application Assembly Tool (AAT)
- ▶ Using the WebSphere Application Developer (WSAD)

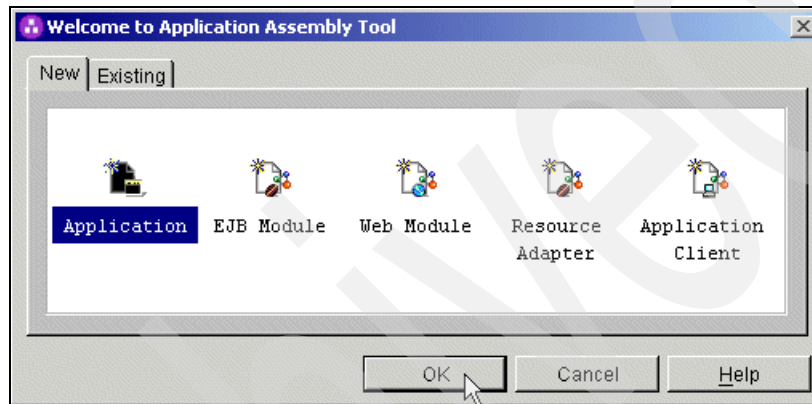
The following sections show the main steps of creating an Enterprise Application using the Application Assembly Tool, which comes with WebSphere. We did that on Windows, so the resulting .ear file had then to be copied to Linux on zSeries. At this point we assume that all files that belong to the application are available, especially all class files are available.

The screen captures are taken from an example that comes with the VSE Connector Client and are intended to show the principle of creating and deploying WebSphere applications.

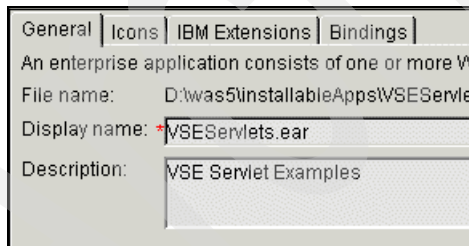
10.6.1 Setting up an EAR file in the Application Assembly Tool

Start the Application Assembly tool and follow these steps in order to create an .ear file.

1. Create a new application.



2. Enter a display name and a description.



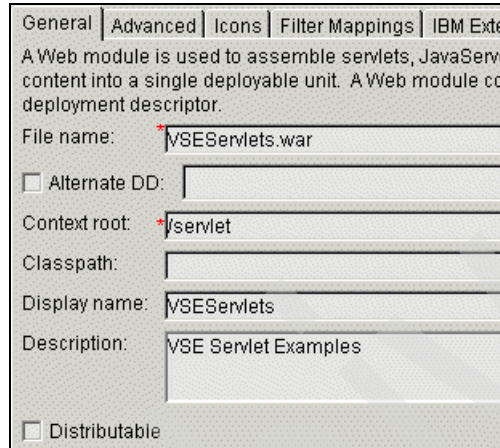
3. Create a Web module for the application.



4. Enter the properties for the Web module. The context root specifies one part of the URL, which you have to use to call this servlet from a Web browser. In this example, the URL would start like:

`http://computername/servlet/.....`

The context root is then followed by the servlet's mapping name, which is specified below in step 12 on page 258.



General | Advanced | Icons | Filter Mappings | IBM Ext

A Web module is used to assemble servlets, JavaServ
content into a single deployable unit. A Web module co
deployment descriptor.

File name: *VSEServlets.war

Alternate DD:

Context root: */servlet

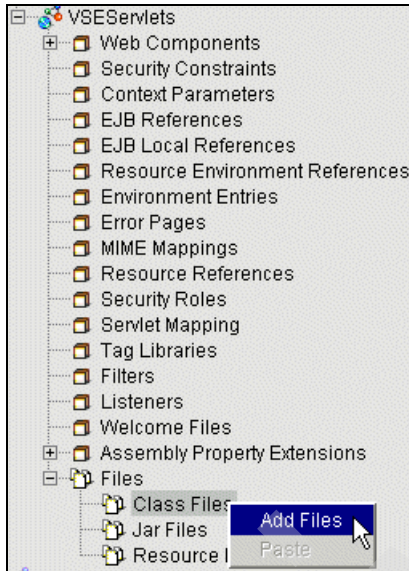
Classpath:

Display name: VSEServlets

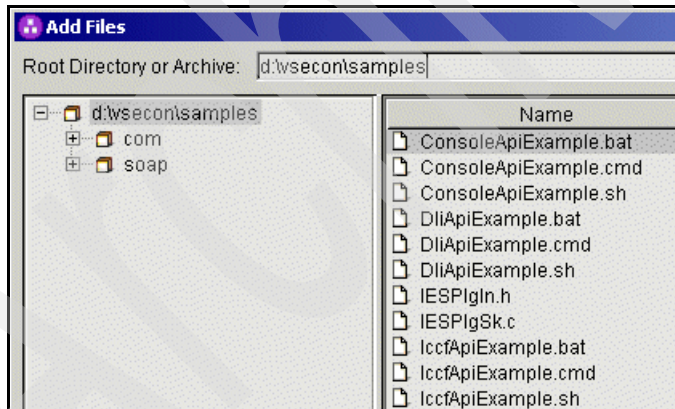
Description: VSE Servlet Examples

Distributable

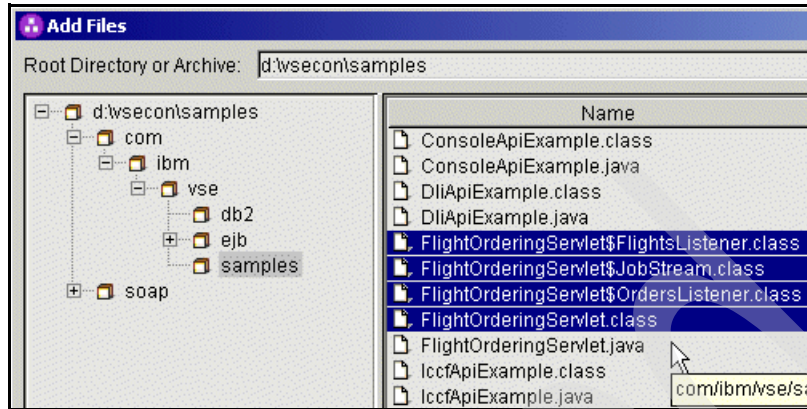
5. Add files that belong to the application. Depending on the particular application this can be class files, but also static HTML files, GIF or JPG images, and so on.



6. In the dialog box enter the root directory of the application and press Enter to display the subdirectories.



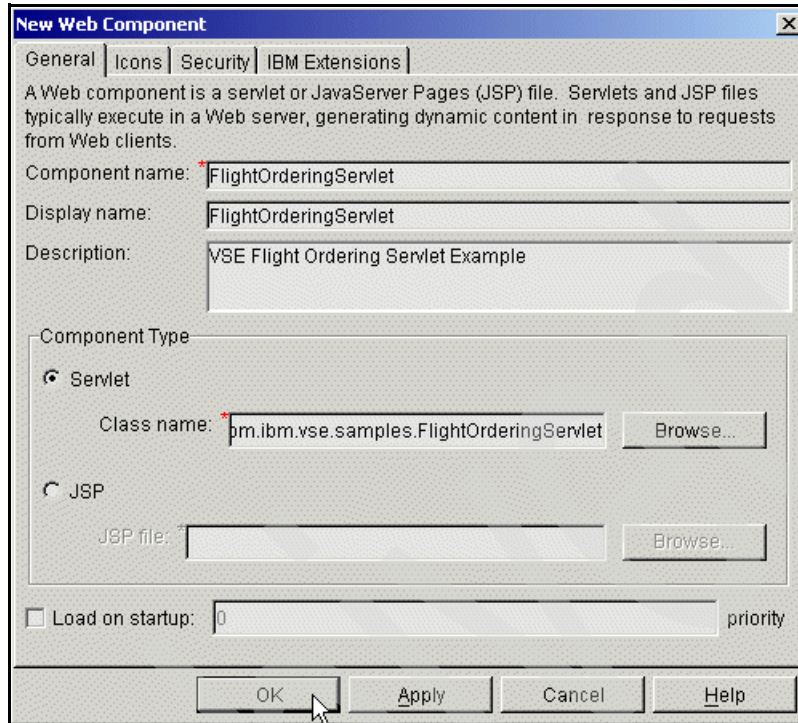
7. Enter the root directory and press Enter to display the files in all subdirectories. Then expand the tree view in order to view the related files.



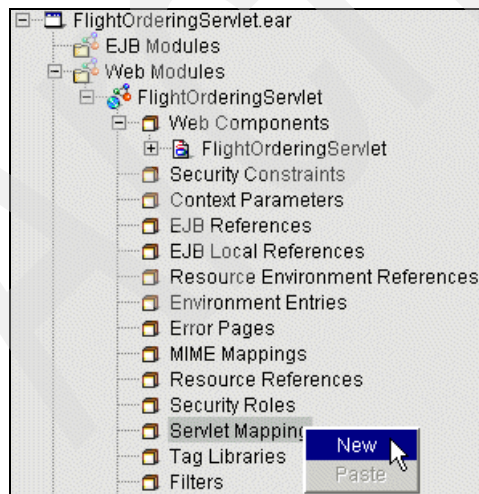
8. Select the appropriate files and add them to the application.
9. Create Web components.



10. Enter properties for Web components. The servlet's class name specifies the servlet's main class.

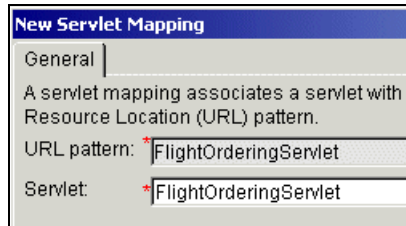


11. Create servlet mappings.

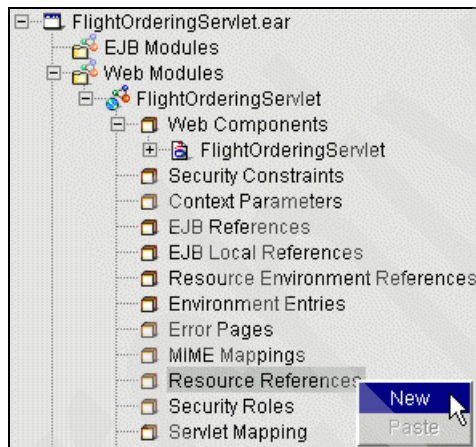


12. Enter the servlet mapping properties. The URL pattern specifies how the servlet can be called from a Web browser. In this example, the complete URL to invoke the servlet would be:

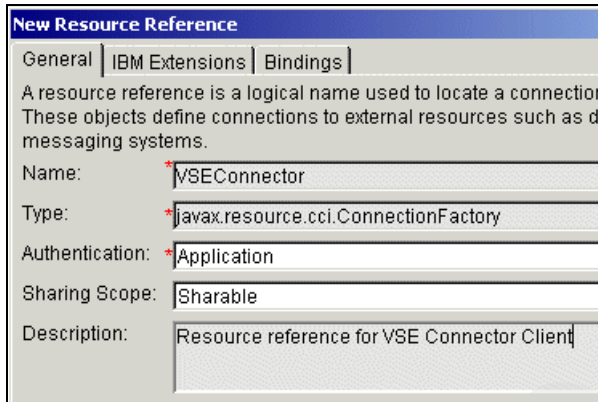
`http://computername/servlet/FlightOrderingServlet`



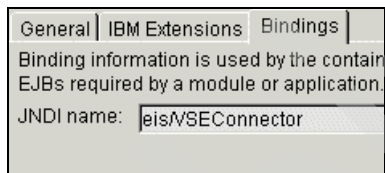
13. Create resource references.



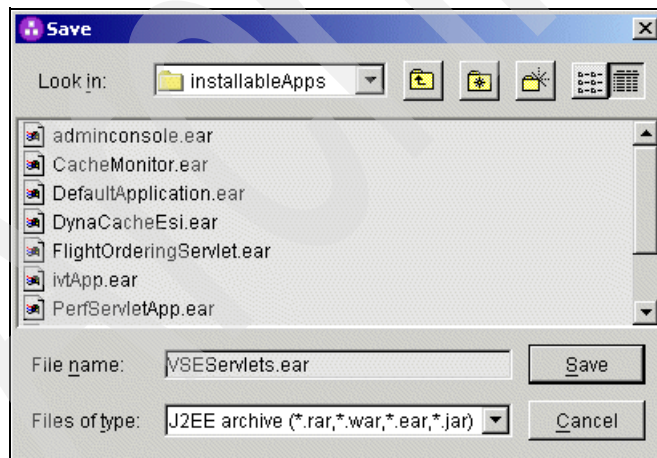
14. Enter the resource reference properties.



15. Enter your previously chosen JNDI name on the Bindings tab.



16. Save your application.



17. Close the Application Assembly Tool.

Now you have to copy the .ear file into directory /installableApps below the WebSphere installation directory. In our environment this was:

```
/opt/WebSphere/AppServer/installableApps
```

10.6.2 Deploying the EAR file in WebSphere Administrative Console

This section gives you an overview of the main steps to deploy a given .ear file to WebSphere. The screen captures are taken from an example provided with the VSE Connector Client. For details about deploying the Trader application, refer to:

- ▶ Deploy the MQ part of Trader (8.3, “WebSphere MQ setup on z/OS backend” on page 162).
- ▶ Deploy the DB2 part of Trader (7.3, “Customizing WebSphere Application Server for DB2 Connect” on page 144).
- ▶ Deploy the CICS part of Trader (5.5, “Configuring WebSphere for CICS connections” on page 105).
- ▶ Deploy the IMS part of Trader (9.5, “Deploying TraderIMS application in WebSphere Application Server” on page 217).

To deploy an .ear file in the WebSphere Administrative Console, follow these steps.

1. Log on to the WebSphere admin console.
2. Install a new application.



3. Enter the path name of the application's .ear file.

Path: Browse the local machine or a remote server:

Local path: Browse...

Server path:

Context Root: Used only for standalone Web modules (*.war)

Next Cancel

4. Create bindings.
5. Enter deployment options. We recommend that you install the new application into directory /installedApps/<computername>, which, in our case was:

/opt/WebSphere/AppServer/installedApps/linux11

Also, we recommend that you enable class reloading.

Step 1: Provide options to perform the installation

Specify the various options available to prepare and install your application.

AppDeployment Options	Enable
Pre-compile JSP	<input type="checkbox"/>
Directory to Install Application	<input type="text" value="ppServer/installedApps/linux11"/>
Distribute Application	<input checked="" type="checkbox"/>
Use Binary Configuration	<input type="checkbox"/>
Deploy EJBs	<input type="checkbox"/>
Application Name	<input type="text" value="VSEServlets"/>
Create MBeans for Resources	<input checked="" type="checkbox"/>
Enable Class Reloading	<input checked="" type="checkbox"/>
Reload Interval in Seconds	<input type="text" value="3"/>
Deploy WebServices	<input type="checkbox"/>

Next Cancel

6. Map resource references to resources. In this step it is important to review all resource references and check if they really match with the JNDI names that

you want to use in your servlet Java code. Here we show some of them as being used in our scenario.

javax.jms.QueueConnectionFactory					
Specify existing Resource JNDI name: <input type="text" value="select..."/> <input type="button" value="Apply"/>					
<input type="checkbox"/>	Module	EJB	URI	Reference Binding	JNDI Name
<input type="checkbox"/>	VSEServlets		VSEServlets.war,WEB-INF/web.xml	TraderQCF	jms/TraderQCF
<input type="checkbox"/>	VSEServlets		VSEServlets.war,WEB-INF/web.xml	MQConnector	jms/TraderVSE

javax.resource.cci.ConnectionFactory					
Specify existing Resource JNDI name: <input type="text" value="select..."/> <input type="button" value="Apply"/>					
<input type="checkbox"/>	Module	EJB	URI	Reference Binding	JNDI Name
<input type="checkbox"/>	VSEServlets		VSEServlets.war,WEB-INF/web.xml	VSEConnectorSSL	eis/VSEConnector/ssl
<input type="checkbox"/>	VSEServlets		VSEServlets.war,WEB-INF/web.xml	VSEConnector	eis/VSEConnector

javax.sql.DataSource					
Specify existing Resource JNDI name: <input type="text" value="select..."/> <input type="button" value="Apply"/>					
<input type="checkbox"/>	Module	EJB	URI	Reference Binding	JNDI Name
<input type="checkbox"/>	VSEServlets		VSEServlets.war,WEB-INF/web.xml	VSAMJDBC	jdbc/vsamjdbc

7. Proceed until the final panel, keeping defaults.
8. Click **Finish**.
9. Save your definitions.
10. Go back to the list of Enterprise Applications and start the new application.

10.7 Configuring for SSL secure connections

To set up SSL for secure connections between the WebSphere platform and a VSE host, we used the IBM-provided tool Keyman/VSE to create RSA keys and digital certificates. However, there are also other ways to do that. Refer to the *VSE/ESA e-business Connectors User's Guide, SC33-6719*, for details.

We did the following steps to set up SSL:

1. Download and install the Keyman/VSE tool.

2. Use Keyman/VSE to generate an RSA key and digital certificates to be used during the SSL handshaking.
3. Store the RSA key, the root certificate and server certificate on VSE.
4. Store the root and server certificate in a keyring file on the WebSphere platform.
5. Change your servlet code in order to support SSL.
6. Configure the VSE Connector Server for SSL.

The following sections describe each of these steps. We used SSL server authentication, so there is no client certificate needed. We used self-signed certificates, which are not signed by an official Certificate Authority (CA) like Thawte or Verisign. So our certificates can only be used in an internal test environment and would be not trusted outside.

For detailed information about SSL refer to the *VSE/ESA e-business Connectors User's Guide*, SC33-6719, or the *VSE Connector Client*.

10.7.1 Installing Keyman/VSE

Keyman/VSE is not shipped as part of VSE. It can be downloaded from:

<http://www-1.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/>

Notes: The tool is based on the VSE Connector Client, so at this point we assume that you have downloaded and installed the VSE Connector Client first from the same Web page. See 10.2, "Installing the VSE Java-based connector" on page 226, for details.

Keyman/VSE requires a JDK 1.4 in order to access crypto-related classes provided by the JDK.

To install Keyman/VSE:

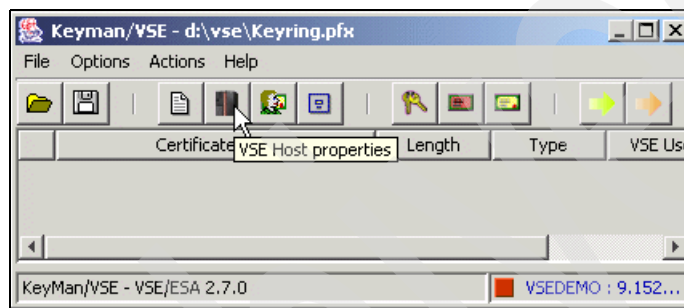
1. Download the Keyman/VSE zip-file in binary and extract the containing files into an empty directory. There are four files contained in the zip file:
 - install.class - Contains the product code and install wizard.
 - install.bat - Install batch file for Windows.
 - install.cmd - Install batch file for OS/2® or Windows NT.
 - install.sh - Install batch file for Unix/Linux.
2. Run the installation batch file appropriate for your platform and follow the install wizard dialogs.

10.7.2 Generating keys and certificates

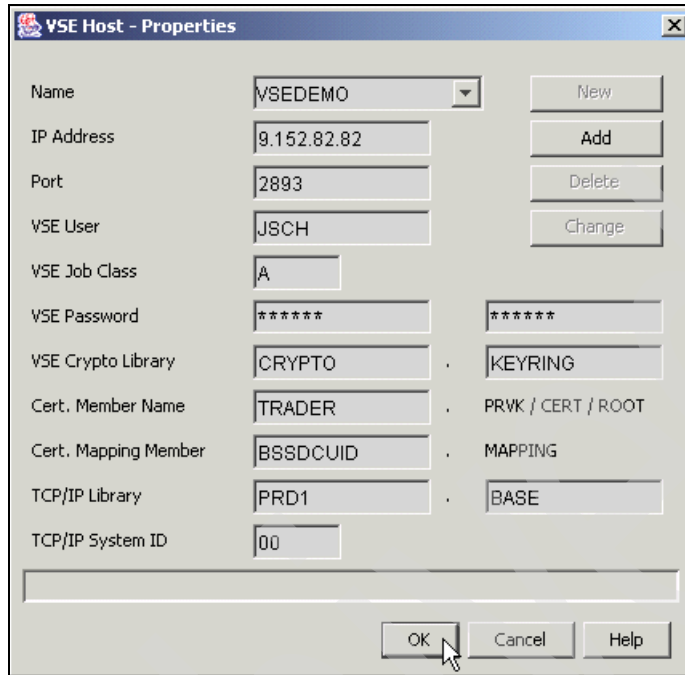
After starting the Keyman/VSE tool, you see the main window, which provides tool bar buttons for generating RSA keys and certificates. Follow these steps to generate the SSL keys and certificates.

To access the online help from the tool's help menu, you have to define a Web browser to the tool. Help is then displayed using this browser. Use the menu option **Help** → **Select help browser** to define the help browser. The How to help section explains all kinds of tasks in detail.

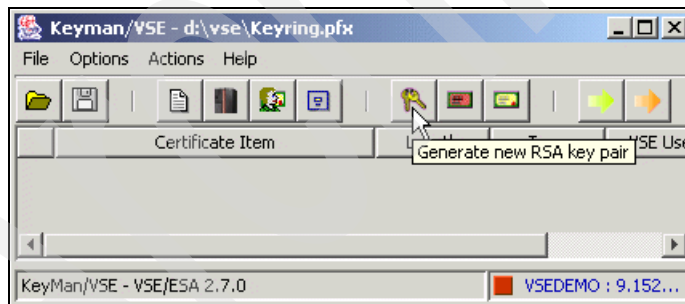
1. Configure your VSE host. Click the **VSE host properties** tool bar button to set up your VSE host.



2. Now define your VSE settings as shown here.



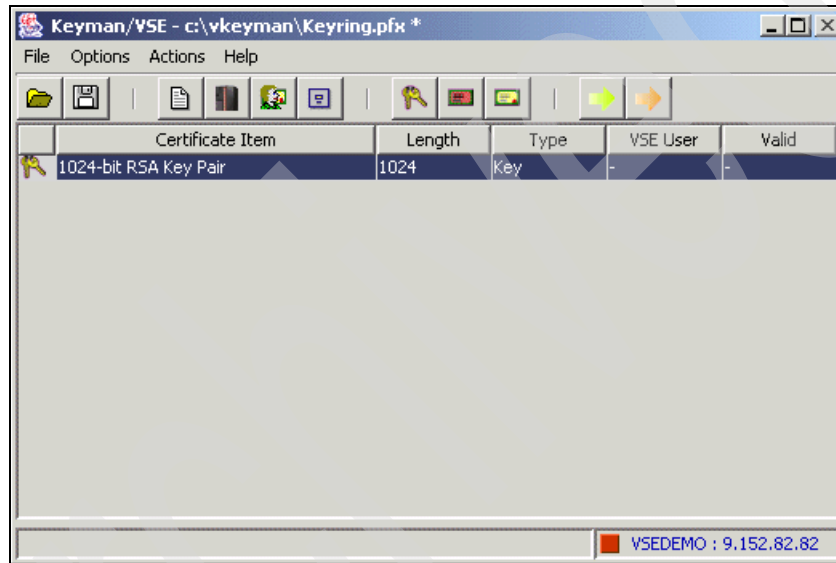
3. Generate an RSA key pair. Click the **Generate RSA key pair** tool bar button.



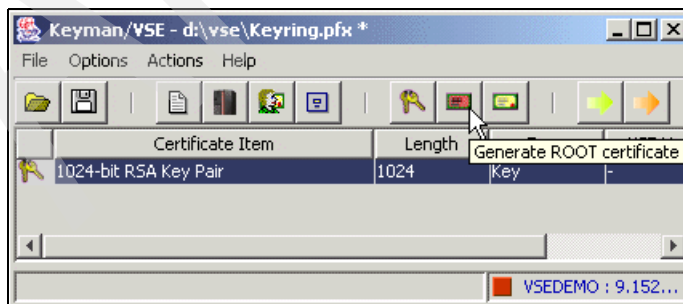
4. Select key length 1024 and click the generate key button.



5. The key is now displayed in the GUI.



6. Generate a self-signed root certificate. Click the **Generate Root certificate** tool bar button.



7. Enter personal information to identify yourself as a Certificate Authority (CA) and click **Generate cert.**

Enter Personal Information for ROOT Cert

Common name: VSE Root certificate for Trader

Organizational Unit: ITSO

Organization: IBM

City/Location: Poughkeepsie

State/Province: NY

Country: US United States (US)

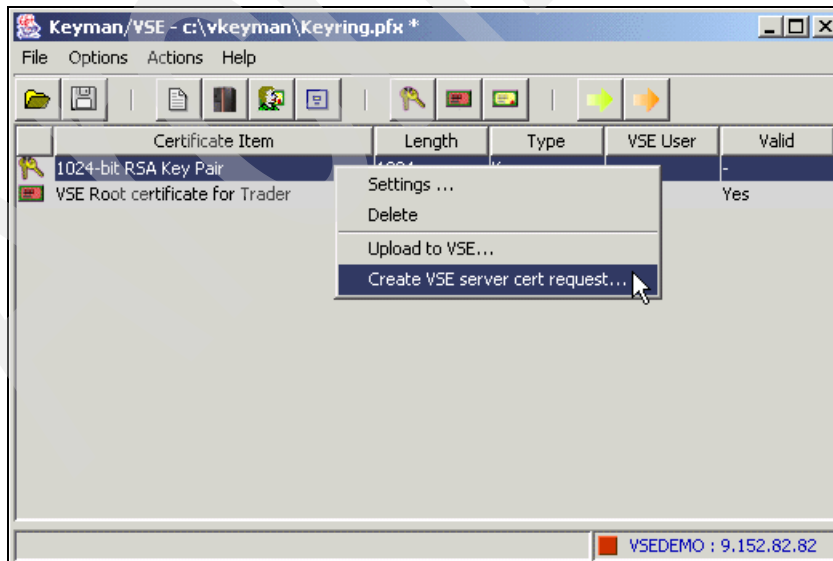
e-mail: vseesa@de.ibm.com

Expires: 2004-11-4 1 year

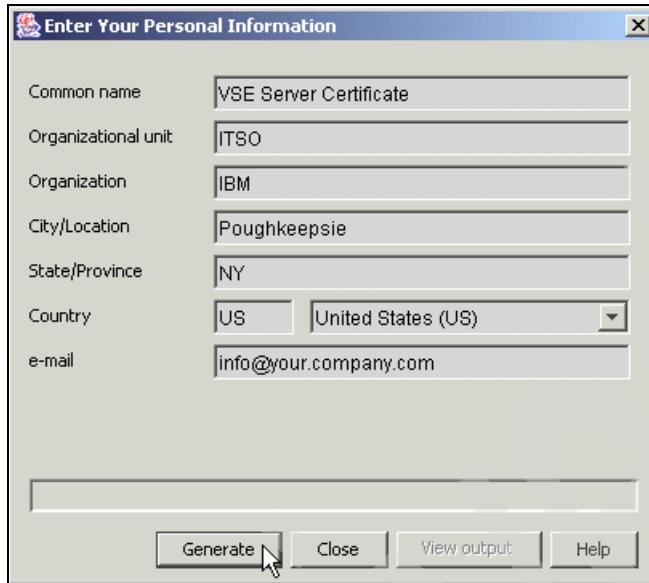
Key length: 1024

Generate cert Close Help

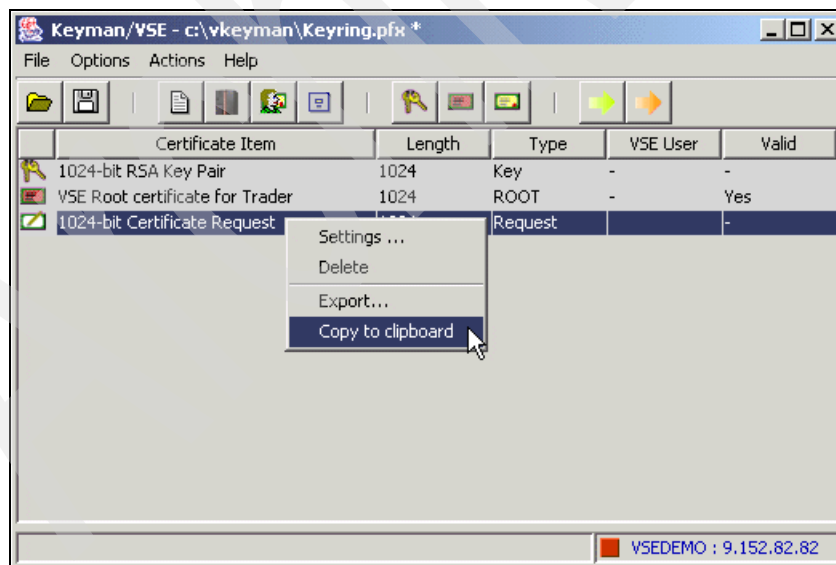
8. Generate a VSE server certificate request. Right-click the key and select **Create VSE server cert request.**



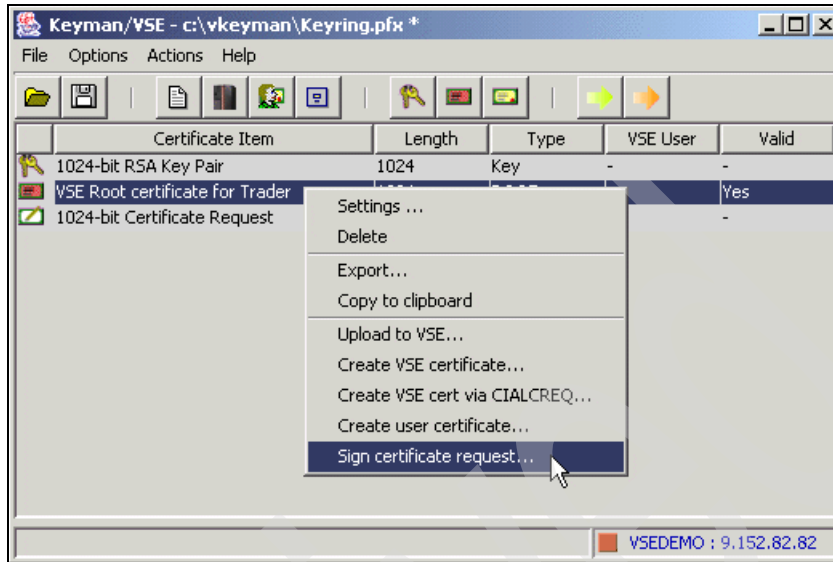
9. Enter personal information to identify the VSE system.



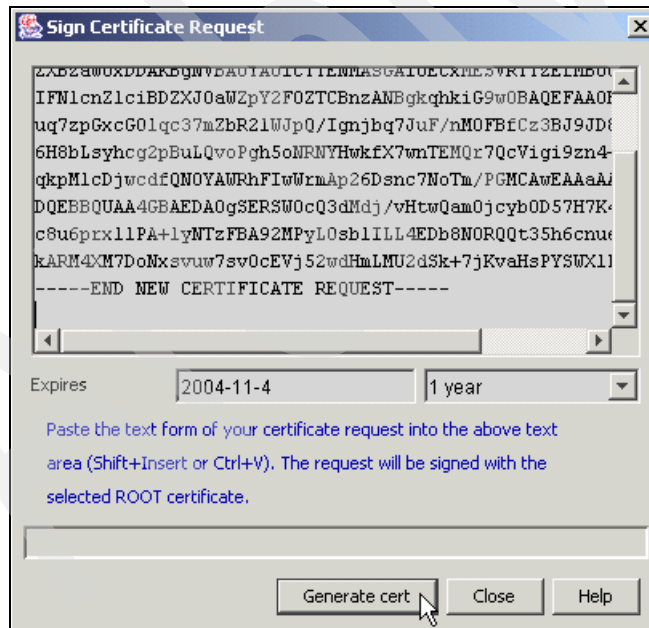
10. Copy the request into the clipboard. Right-click the request and select **Copy to clipboard**.



11. Sign the request with your self-signed root certificate. Right-click the root certificate and select **Sign certificate request**.



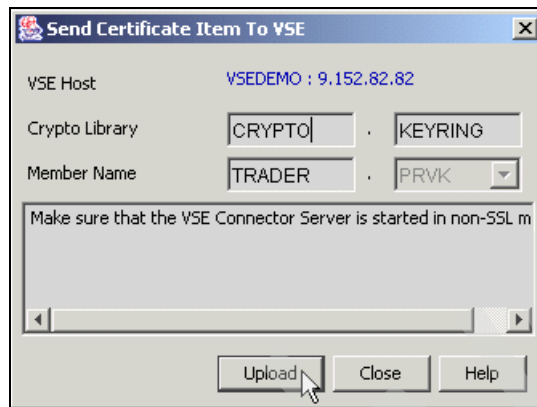
12. Paste the clipboard content into the text area and click **Generate cert.**



13. Delete the certificate request from the list by right-clicking it and selecting **Delete** (or pressing the DEL key on your keyboard), and upload the three remaining items (key, root cert, and server cert) to VSE.

10.7.3 Uploading certificate items to VSE

Right-click each remaining item and select **Upload to VSE**. This box shows how the RSA key is uploaded. You might change the VSE library or library member name at this point, but the member type is of course preset.



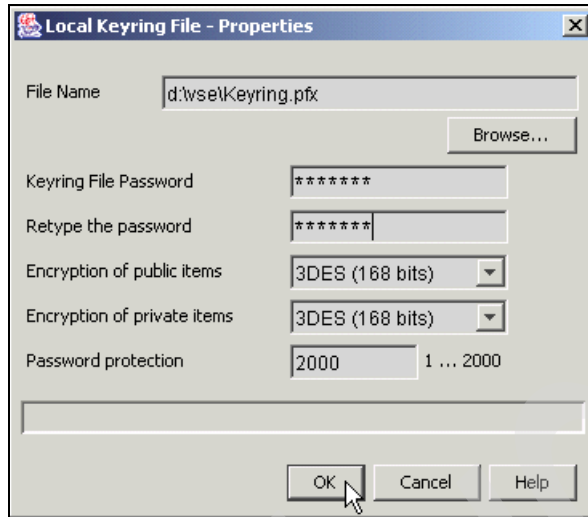
On the VSE side you should see console messages similar to the ones shown in Example 10-9.

Example 10-9 Catalog RSA key on VSE

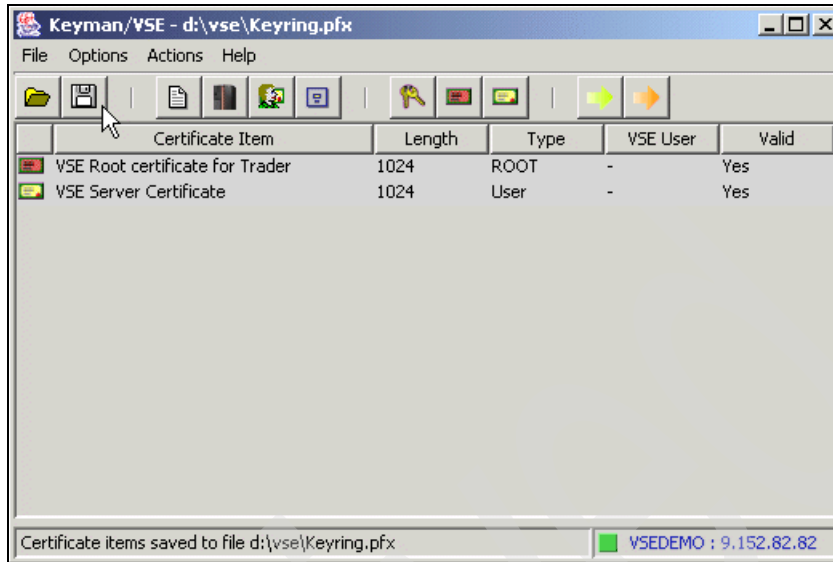
```
BG 0000 // JOB CIALSRVR
          DATE 11/04/2003, CLOCK 23/07/53
BG 0000 CIALSRVR 01.05 B 07/22/03 20.45
BG 0000 Default password phrase will be used
BG 0000 SETPORT 6045
BG 0000 Waiting for PC to send rsa private key.
BG 0000 1024-bit RSA key written into CRYPTO .KEYRING .TRADER .PRVK
BG 0000 EOJ CIALSRVR MAX.RETURN CODE=0000
          DATE 11/04/2003, CLOCK 23/07/59, DURATION 00/00/06
BG 0001 1Q34I  BG WAITING FOR WORK
```

Repeat this step for the root certificate and the server certificate.

Specify your local keyring file properties, that is, the name and location of the file, the settings for the encryption of the certificate items in the file, and the keyring file password. Click the **Local file** properties tool bar button or select **Options** → **Keyring file properties**. Remember your keyring file password, you will need it later when coding your servlet or defining your SSL connection factory or SSL JDBC data source in WebSphere.



14. Store both certificates in the local keyring file. The root certificate is required on the client side to be able to perform the SSL handshaking. The server certificate is not required on the client side, but allows the client to verify the received server certificate during the SSL handshake by comparing it with the local copy.
15. Delete the key, which is not stored in the keyring file, then click the **Save** tool bar button.



The next step is to transfer the keyring file to the WebSphere platform so it can be accessed by the servlet.

10.7.4 Transferring the keyring file to WebSphere

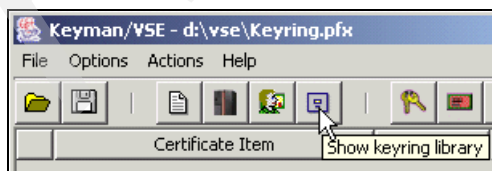
We used FTP to copy the keyring file into a new directory on Linux for zSeries from where WebSphere can access it:

/opt/vse

This directory is referenced in the SSL Connection Factory properties; see 10.7.6, "Defining an SSL connection factory in WebSphere" on page 273.

10.7.5 Checking VSE keyring library

Keyman/VSE allows you to access the VSE keyring library by clicking the **Show keyring library** tool bar button.



The VSE keyring library now contains these members: The VSE server certificate (TRADER.CERT), the private key (TRADER.PRVK), and the self-signed root certificate (TRADER.ROOT). You may view their properties by right-clicking an item and selecting **Settings**.



10.7.6 Defining an SSL connection factory in WebSphere

To support an SSL connection to VSE, we created a second J2C connection factory for the VSE resource adapter. Here, we specified the necessary SSL properties.

1. Log on to the WebSphere admin console.
2. Click **Resource Adapters**.
3. Click resource adapter **VSEConnector**.
4. Click **J2C Connection Factories**.
5. Create a new connection factory for SSL.



6. Specify the properties of the new connection factory.

Note: You can freely choose the JNDI name here. However, a naming convention in a production environment would call it, for example, eis/vse270/ssl to indicate the SSL connection to a specific VSE back-end system. Further connection factories, which go to other VSE systems, would similarly include their names into their JNDI names.

General Properties	
Scope	* cells:linux11:nodes:linux11
Name	* VSE SSL Connection Factory
JNDI name	eis/VSEConnector/ssl

Click **OK**.

7. Select the connection factory to specify further properties.

<input type="checkbox"/> Name ▾	JNDI Name ▾
<input type="checkbox"/> VSE Connection Factory	eis/VSEConnector
<input type="checkbox"/> VSE SSL Connection Factory	eis/VSEConnector/ssl

8. Scroll down and click **Custom Properties**.

Additional Properties	
Connection Pool	An optional set of connection pool settings.
Custom Properties	Properties that may be required for Resource additional custom properties for data sources

9. Edit the given parameters so that you get a list similar to the following.

Name ▾	Value ▾
ServerName	9.152.82.82
PortNumber	2893
UserName	JSCH
Password	mypasswd
SSLVersion	SSL
CipherSuites	SSL_RSA_WITH_DES_CBC_SHA,SSL_RSA_WITH_3DES_EDE_CBC_SHA,SSL
KeypingFile	/opt/vse/Keyping.ptx
KeypingPassword	ssltest

Leave the list of cipher suites as it is provided by default. We set SSLVersion to SSL, which means SSL V3.0. Specifying TLS would mean SSL V3.1 respectively TLS V1.0. We copied the keyring file to the directory /opt/vse, which is reflected in the KeypingFile parameter.

10. Save your changes.

10.7.7 Adding SSL resource reference in EAR file

Start the Application Assembly tool and open your EAR file. Now add a new resource reference for the SSL connection:

1. Specify general properties.

General	IBM Extensions	Bindings
A resource reference is a logical name used to locate a connection factory or other external resources such as databases and messaging systems.		
Name:	*VSEConnectorSSL	
Type:	*javax.resource.cci.ConnectionFactory	
Authentication:	*Application	
Sharing Scope:	Sharable	
Description:		

2. Specify the JNDI name for SSL.

General	IBM Extensions	Bindings
Binding information is used by the container to locate external resources.		
JNDI name:	eis/VSEConnector/ssl	

3. Save the EAR file.

4. Copy the saved EAR file to your WebSphere platform into directory installableApps.

5. Go back to the WebSphere admin console.

10.7.8 Redeploying the EAR file

In the WebSphere admin console:

1. Click **Applications** → **Enterprise Applications**.
2. Stop your application.
3. Select your application and click the **Update** button.
4. Specify ear file path name.

Path:	Browse the local machine or a remote server:	<i>i</i> Choose browser. C your cell co
	<input type="radio"/> Local path: <input type="text"/> <input type="button" value="Browse..."/>	
	<input checked="" type="radio"/> Server path: <input type="text" value="stallableApps/VSEServlets.ear"/>	
Context Root:	Used only for standalone Web modules (*.war)	<i>i</i> You must module.
	<input type="text"/>	
<input type="button" value="Next"/> <input type="button" value="Cancel"/>		

5. In the next panel select override existing bindings.

Specify bindings to use <input type="text" value="merge new and existing bindings"/>		
<input type="checkbox"/> Generate Default Bindings (applicable only if using "new" or "merged" bindings)		
Override:	<input type="radio"/> Do not override existing bindings <input checked="" type="radio"/> Override existing bindings	<i>i</i> Generate d them.
Virtual Host	<input type="radio"/> Do not default virtual host name for web modules <input checked="" type="radio"/> Default virtual host name for web modules: <input type="text" value="default_host"/>	<i>i</i> The virtual
Specific bindings file:	<input type="text"/> <input type="button" value="Browse..."/>	<i>i</i> Optional lo
<input type="button" value="Previous"/> <input type="button" value="Next"/> <input type="button" value="Cancel"/>		

6. Provide further options unless already shown.

Step 1: Provide options to perform the installation

Specify the various options available to prepare and install your application.

AppDeployment Options	Enable
Pre-compile JSP	<input type="checkbox"/>
Directory to Install Application	ppServer/installedApps/linux11
Distribute Application	<input checked="" type="checkbox"/>
Use Binary Configuration	<input type="checkbox"/>
Deploy EJBs	<input type="checkbox"/>
Application Name	VSEServlets
Create MBeans for Resources	<input checked="" type="checkbox"/>
Enable Class Reloading	<input checked="" type="checkbox"/>
Reload Interval in Seconds	3
Deploy WebServices	<input type="checkbox"/>

Next Cancel

7. Map resource references to resources.

javax.resource.cci.ConnectionFactory

Specify existing Resource JNDI name:

linux11:eis/VSEConnector/ssl

<input type="checkbox"/>	Module	EJB	URI	Reference Binding	JNDI Name
<input type="checkbox"/>	VSEServlets		VSEServlets.war,WEB-INF/web.xml	VSEConnectorSSL	eis/VSEConnector/ssl
<input type="checkbox"/>	VSEServlets		VSEServlets.war,WEB-INF/web.xml	VSEConnector	eis/VSEConnector

8. Restart your application.

10.7.9 Configuring VSE Connector Server for SSL

The VSE Connector Server is configured through a set of config members. There are job skeletons in ICCF library 59 that can be used to catalog the files. We recommend that you copy the job skeletons into a separate ICCF library before changing them. You need the following members to do the SSL setup:

- ▶ SKVCSCFG to enable SSL
 - Specify YES for parameter SSLENABLE.

Example 10-10 Enable SSL for VSE Connector Server

```
...
; *****
; 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 = YES
...
```

► SKVCSSSL to specify SSL properties

Specify the name of your private key and certificate members. We used TRADER as the member name when uploading the key and certificates to VSE. You do not need to change the list of cipher suites.

Example 10-11 Specify VSE Connector Server SSL properties

```
...
SSLVERSION = SSL30
KEYRING = CRYPTO.KEYRING
CERTNAME = TRADER
SESSIONTIMEOUT = 86440
AUTHENTICATION = SERVER
...
```

► SKVCSCAT to catalog the changes

Edit the job skeleton to catalog the two changed config members. Then submit the catalog job and restart the VSE Connector Server.

Note: Now the server can only handle SSL connections. If you want to also connect from other applications without using SSL, you have to start another instance of the VSE Connector Server in a separate partition using a different port number. If these are WebSphere applications, the new port number must be referenced in the related connection factories.

Example 10-12 Catalog VSE Connector Server config members

```
* $$ JOB JNM=VCSCAT,DISP=D,CLASS=0
// JOB VCSCAT CATALOG VCS CONFIGURATION MEMBERS
// EXEC LIBR,PARM='MSHP'
ACCESS S=PRD2.CONFIG
CATALOG IESVCSRV.Z REPLACE=Y
* $$ SLI ICCF=(SKVCSCFG),LIB=(11)
```



```
/+
CATALOG IESSSLCF.Z REPLACE=Y
* $$ SLI ICCF=(SKVCSSSL),LIB=(11)
/+
/*
/&
* $$ E0J
```

10.7.10 Restarting VSE Connector Server

The VSE Connector Server must be restarted in order to read the new configuration members and activate SSL. First, stop the server using following console command:

```
msg r1,data=shutdown
```

Then restart the server by releasing the STARTVCS job.

```
r rdr,startvcs
```

Check the startup messages if the server is running in SSL mode.

Example 10-13 Check VSE Connector Server startup messages

```
msg r1,data=status
AR 0015 1140I  READY
R1 0045 IESC1029I STATUS COMMAND
R1 0045     SERVER CONFIG FILE     = DD:PRD2.CONFIG(IESVCSRV.Z)
R1 0045     CONFIGURATION INFORMATION:
R1 0045     MAX NUM. OF CLIENTS     = 256
R1 0045     TCP/IP SERVER PORT      = 2893
R1 0045     SSL ENABLED             = YES
...
```

10.7.11 Changing your servlet code to support SSL

In the servlet code we just use the second J2C connection factory, which we have defined to support SSL. So the only difference in the servlet is the different JNDI name used when creating the initial context.

Example 10-14 Use an SSL connection in the servlet

```
...
Context ctx = new InitialContext();
VSEConnectionSpec spec = new VSEConnectionSpec(ctx, "eis/VSEConnector/ssl");
...
```

10.7.12 Configuring an SSL JDBC data source

The definition of an SSL JDBC data source is very similar to defining a JDBC data source without using SSL. So this section shows only such screen prints that are different from those to define a JDBC data source without SSL. Refer to Figure 10-3 on page 232. In the following steps we add a second JDBC data source to the one already created.

1. Log on to the admin console by entering any user ID. Click **Resources** → **JDBC Providers**.
2. Click **New** to add a new JDBC provider.
3. Create a user-defined JDBC provider.
4. Enter the properties for the JDBC provider. In the field Classpath enter the three VSE Connector jar files. Remove any other entries.

General Properties	
Scope	* cells:linux11:nodes:linux11
Name	* VSE SSL JDBC Provider
Description	VSE SSL JDBC Provider for VSAM
Classpath	/opt/vse/VSEConnector.jar /opt/vse/ibmjsse.jar /opt/vse/ccci.jar

In the field Implementation Classname enter:

`com.ibm.vse.jdbc.VsamJdbcConnectionPoolDataSource`

It must be exactly this string, because it references the Java class, which implements the JDBC data source. Then click **OK**.

5. Specify further properties. Click the JDBC provider name.

<input type="checkbox"/> Name
<input type="checkbox"/> DB2 Legacy CLI-based Type 2 JDBC Driver
<input type="checkbox"/> VSAMJDBC
<input type="checkbox"/> VSE SSL JDBC Provider

6. Click **Data Sources**.

7. Click **New** to create a new data source for this JDBC provider.
8. Specify the data source properties.

General Properties	
Scope	* cells:linux11:nodes:linux11
Name	* SSL JDBC Data Source for VSAM
JNDI Name	jdbc/vsamjdbc/ssl

Click **OK**.

Note: You can freely choose the JNDI name here. However, a naming convention in a production environment would call it, for example, jdbc/vsam/vse270/ssl to indicate the SSL JDBC VSAM connection to a specific VSE back-end system. Further connection factories, which go to other VSE systems, would similarly include their names in their JNDI names.

9. Specify further properties of the data source. Click the data source name.
10. Select our previously created authentication entry from the check box "Component-managed Authentication Alias".

Component-managed Authentication Alias	linux11/VSE user for ECI
--	--------------------------

This VSE user ID is used by the JDBC driver to connect to VSE. If it is not specified, the VSE Connector Server would reject the connection later, because JDBC would try to connect without sending a user ID. Click **Apply**.

11. Scroll down and click **Custom Properties** to define database-specific properties needed to access VSE.
12. Add the VSE server name as property serverName. Enter it mixed case exactly as shown here. Leave the type field as String.
13. Add the VSE Connector Server port number as property portNumber. Enter it mixed case exactly as shown here. Select type as Integer.

Note: It is important that you select type **Integer** from the drop-down combo box. Otherwise the connection will not work, because the JDBC driver expects this parameter as an integer value.

14. Now indicate to use SSL by specifying `SSLVersion=SSL`, which means to use SSL Version 3.0. Specifying TLS would mean using `SSL V3.1`, respectively `TLS V1.0`. However, you should always use `SSL V3.0` if not otherwise needed. You should end up with a list similar to the following.

<input type="checkbox"/>	Name ▾	Value ▾
<input type="checkbox"/>	serverName	9.152.82.82
<input type="checkbox"/>	portNumber	2883
<input type="checkbox"/>	SSLVersion	SSL
<input type="checkbox"/>	keyringFile	/opt/vse/Keyring.pfx
<input type="checkbox"/>	keyringPassword	ssstest
<input type="checkbox"/>	cipherSuites	SSL_RSA_WITH_DES_CBC_SHA,SSL_RSA_WITH_3DES_EDE_CBC_SHA,SSL

15. Save your changes and restart WebSphere.

10.7.13 Considerations on SSL key lengths

Depending on the RSA key length, only a subset of the cipher suites is applicable. Also, some cipher suites can only be used with a specific SSL version. The following table shows these relationships.

Example 10-15 Relationships

RSA key len	Cipher suite	Encryption	Hex code	SSL version
512 bit	SSL_RSA_WITH_NULL_MD5	None	01	SSL
512 bit	SSL_RSA_WITH_NULL_SHA	None	02	SSL
512 bit	SSL_RSA_EXPORT_WITH_DES40_CBC_SHA	40-bit DES	08	SSL or TLS
1024 bit	SSL_RSA_WITH_DES_CBC_SHA	56-bit DES	09	SSL or TLS
1024 bit	SSL_RSA_WITH_3DES_EDE_CBC_SHA	168-bit TDES	0A	SSL or TLS

The cipher suite with hex code 62 is not supported by the Java-based connector.

Tip: in a production environment you should always use 1024-bit handshaking together with the most secure cipher suite `SSL_RSA_WITH_3DES_EDE_CBC_SHA` using triple-DES (TDES).

10.7.14 Considerations on different SSL scenarios

Until now, we considered SSL server authentication, because this is the easiest way of setting up SSL. But there are other ways to set up SSL, which are more complex to define.

Detailed information about all kinds of SSL setup can be found in the *VSE/ESA e-business Connectors User's Guide*, SC33-6719, and in the *VSE Connector Client*. SSL setup for CICS Web Support (CWS) is described in the *CICS Enhancements Guide*, GC34-5763.

This section tries to give you an overview of different SSL setup possibilities, supported by the Java-based connector in VSE.

VSE provides three flavors of SSL setup:

- ▶ SSL server authentication

Here the server side sends its certificate to the client, who decides whether to trust this certificate or not. The client side does not need any certificate.

This scenario is typically used for online banking, where the client uses a Web browser to connect to a bank's Web page. The bank server sends its digital certificate, and the client decides whether to just accept it for this session or to import it into the Web browser permanently. The client authenticates itself using PIN and TAN numbers.

- ▶ SSL client authentication

Here both sides need to have a digital certificate. The server side sends its certificate to the client and requests the client's certificate for authentication.

This scenario is used in sensitive environments, with an even higher need for security. Typically, both sides have copies of the counterpart's certificate, so that authentication can be done by simply comparing the received certificate with the local copy.

- ▶ SSL client authentication with implicit logon

This is a special feature of zSeries operating systems, based on SSL client authentication. Here, client certificates are mapped to mainframe user IDs, which allows a client to sign on to a host without providing explicit logon parameters, such as user ID and password. When the server side receives a client certificate during the SSL handshaking process, it retrieves the user ID and password from its security manager and signs the client on implicitly.

10.8 Problem determination

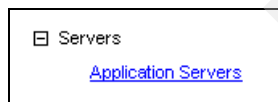
This section gives you some hints of how to debug or trace a WebSphere application, like a servlet, without a big development environment.

10.8.1 Activating stdout trace in WebSphere

You can use the `System.out.println()` method to generate trace output to standard out. However, you have to explicitly specify the file where stdout goes to. Otherwise no output will be written. At least this was the observation in our test environment.

To specify stdout:

1. Open the WebSphere admin console.
2. Click **Application Servers**.



3. Click your server.



4. Click **Logging and tracing**.



A screenshot of the 'Additional Properties' dialog box in the WebSphere admin console. It shows a table with several properties and their descriptions. The 'Logging and Tracing' property is selected, and a mouse cursor is pointing at the link.

Additional Properties	
Transaction Service	Specify settings for th
Web Container	Specify thread pool ar and tuning parameters
EJB Container	Specify cache and da
Dynamic Cache Service	Specify settings for th
Logging and Tracing	Specify Logging and T
Message Listener Service	Configuration for the M whereby MPE are d

5. Click **JVM logs**.

Logging and Tracing	
Diagnostic Trace	View and r
JVM Logs	View and r
Process Logs	View or mc
IBM Service Logs	Configure t

6. Here enter any file name for stdout and stderr.

General Properties	
System.out	
File Name:	d:\was5\logs\wse_std_out.log
File Formatting	Basic (Compatible)
Log File Rotation	<input checked="" type="checkbox"/> File Size Maximum Size 1 MB

7. Save your changes to the main configuration.

8. Restart the WebSphere server. In our test environment the changes did not become active otherwise.

10.8.2 Tracing a servlet

Sometimes you may want to change your existing servlet code and quickly apply the changes to a running WebSphere environment. This is easy, because all servlet classes are stored in the server machine's file system and can be re-loaded dynamically (make sure you specified Enable class reloading, when installing the enterprise application).

A very simple servlet development environment could look like:

- ▶ Edit a servlet's Java source code using any text editor you like.
- ▶ Use a simple batch file or script to compile the code and copy the resulting class file to the right destination.

Example 10-16 Update the servlet class file in WebSphere

```
setlocal
set lib=d:\j2ee\lib
set
classpath=.;d:\vsecon\cci.jar;d:\vsecon\ibmjse.jar;%lib%\j2ee.jar;%lib%\jaas.jar;%lib%\jta-spec1_0_1.jar;%classpath%
d:\jdk1.3.1\bin\javac com\ibm\vse\samples\MyServlet.java
```

```
copy com\ibm\vse\samples\MyServlet.class
D:\was5\installedApps\jschmidb\VSEServlets.ear\VSEServlets.war\WEB-INF\classes\
com\ibm\vse\samples
pause
endlocal
```

A similar batch process can be set up with transferring the servlet's class file to a Linux platform using FTP, or do the whole batch job on Linux.

- ▶ After compiling the servlet code and copying the class file, you can immediately run the updated servlet with a Web browser without stopping the WebSphere service or even stopping the enterprise application.

VSE Java-based connector to access DL/1 data

This chapter provides a description of VSE Java-based connector to access DL/1 data on VSE.

We discuss the following topics:

- ▶ DL/1 database access overview
- ▶ Prerequisites for the DL/1 connector
- ▶ DL/1 example

More information about configuring DL/1 for access via the VSE JavaBeans and defining the sample database can be found in the *VSE/ESA e-Business Connectors User's Guide*, SC33-6719.

11.1 DL/I database access overview

With VSE/ESA 2.7 the Java-based connector provides native access to DL/I, as shown in Figure 11-1.

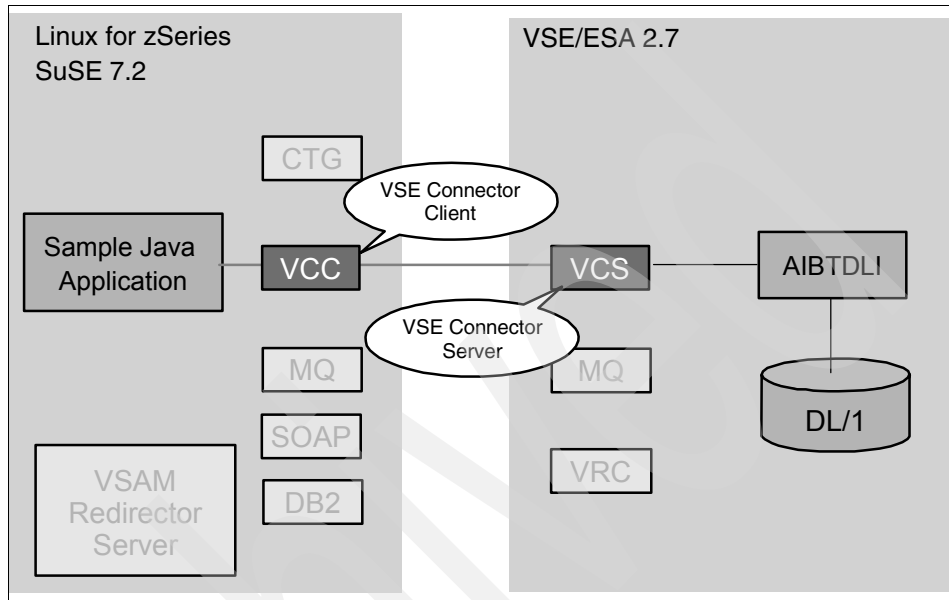


Figure 11-1 DL/I database access overview

In previous VSE releases this was only possible via DB2 Stored Procedures. This can be used in any kind of Java program, including WebSphere applications, like applets, servlets and JSPs.

The following new VSE JavaBeans are now part of the VSE Connector Client class library (VSEConnector.jar).

- ▶ VSEDli - Represents the DL/I subsystem on VSE/ESA
- ▶ VSEDliPsb - Represents a DL/I PSB with its corresponding PSB name
- ▶ VSEDliPcb - Represents a DL/I PCB that can be used to execute DL/I

On the host side, the VSE Connector Server has a plugin IESDLIPL that uses the AIBTDLI interface introduced with VSE/ESA 2.5. This is an batch interface, but it routes the DLI calls to a CICS system and executes the DL/I call in the DL/I online nucleus.

For more information about the AIBTDLI interface and its features/restrictions, see the *VSE/ESA e-business Connectors User's Guide*, SC33-6719.

11.1.1 Prerequisites for the DL/I connector

The following prerequisites must be met in order to use the DL/I connector:

- ▶ The AIBTDLI interface must be installed.
- ▶ DL/I VSE 1.11 or later.
- ▶ APAR PQ39683 for DL/I must be applied.
- ▶ Your CICS/DLI system should have all PSBs defined in the DL/I online nucleus DLZNUCxx and it should have an active MPS system.
- ▶ The DL/I task termination exit DLZBSEOT must be resident in the SVA.

11.1.2 The DL/I example

To run the IBM-provided DL/I example, which is included in the VSE Connector Client, the following definitions are necessary:

- ▶ The sample database STDIDBP must be created; see job skeleton SKDLISMP in ICCF library 59.
- ▶ The sample database STDIDBP must be defined in the CICS FCT.

The following code snippets show some major code parts of the DL/I example.

Access the DL/I subsystem

The first step accesses the DL/I subsystem (Example 11-1).

Example 11-1 Access the DL/I subsystem

```
...
VSEDli dli = new VSEDli(system);

/* Schedule the PSB */
psb = dli.getDliPsb("STBICLG");
psb.schedule();

/* get the first PCB */
pcb = psb.getVSEDliPcb(0);

byte[] ioarea;
String[] ssas;
```

List the segments

Example 11-2 shows/prints the contents of the IO area.

Example 11-2 List the 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;

    /* Print out the contents of the IOArea
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("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);
```

The example further shows how to add, update, or delete a DL/I segment.

Generated output from the DL/I example

When running the example, output similar to the Example 11-3 should be produced.

Example 11-3 Output from the DL/I example

```
D:\vsecon\samples>java com.ibm.vse.samples.DliApiExample
Please enter your VSE IP address:
9.152.82.82
Please enter your VSE user ID:
jsch
Please enter password:
mypassw
Schedule the PSB
```

```
Num PCBs: 1
IO len: 168

Get a PCB
DBDName = STDIDBL
Processing Options: AP

List all DL/I segments
Status =
IOArea:
  ITNUMB = 000300
  ITDESC = RESISTORS
  IQOH   = 000040
  IQOR   = 000080
  IUNIT  = 000002
Status =
IOArea:
  ITNUMB = 000500
  ITDESC = CAPACITORS
  IQOH   = 000500
  IQOR   = 000500
  IUNIT  = 000001
Status =
...
Add/update a segment
Status = GE
Segment not found, do an insert
IOArea:
  ITNUMB = 000700
  ITDESC = INSERTED ITEM
  IQOH   = 000001
  IQOR   = 000002
  IUNIT  = 000003
Status =

Delete a segment
Status =
Do the delete
IOArea:
  ITNUMB = 000700
  ITDESC = INSERTED ITEM
  IQOH   = 000001
  IQOR   = 000002
  IUNIT  = 000003
Status =

Terminate the PSB
```

On the VSE console the following output is produced.

Example 11-4 VSE console messages from DL/I example

```
R1 0118 DLZ143I MPS BATCH CONNECT REQUEST FOR PARTID=F2, APPLID=DBDCCICS
R1 0118 DLZ081I MPS BATCH DL/I PARTITION STARTED
F2 0105 DLZ103I R1 BPC STOPPED NORMALLY
```

VSE VSAM Redirector connector

This chapter describes the VSE VSAM Redirector connector, which allows you to redirect VSAM accesses to any database or file system on any Java-enabled platform that can be reached via TCP/IP. This allows existing VSE programs, which access VSAM data, to work with any data on remote platforms, such as DB2 data on Linux for zSeries or z/OS.

We first show how to run an IBM-provided example, which redirects VSAM data into an HTML file. Next we show the possibility of integrating the Redirector connector in the Trader scenario where VSAM data is redirected into a DB2 for z/OS database.

We describe the following topics:

- ▶ VSAM Redirector connector overview
- ▶ Client-server components
- ▶ Installing the Redirector server
- ▶ Testing the setup with the HtmlHandler
- ▶ Setup for Trader using the DB2 hanfler

12.1 VSAM Redirector connector overview

The VSAM Redirector connector makes use of a VSAM vendor exit implementation, where VSAM requests can be intercepted and, instead of accessing an underlying VSAM file, forwarded to another platform into any other file system or database.

Figure 12-1 shows our VSAM Redirector connector scenario. A Trader COBOL program accesses a VSAM file, which is redirected by the VSAM Redirector Client (VRC) on VSE to the VSAM Redirector Server, running on Linux for zSeries. The request is targeted into a remote DB2 database on z/OS (not shown) via DB2 Connect on Linux.

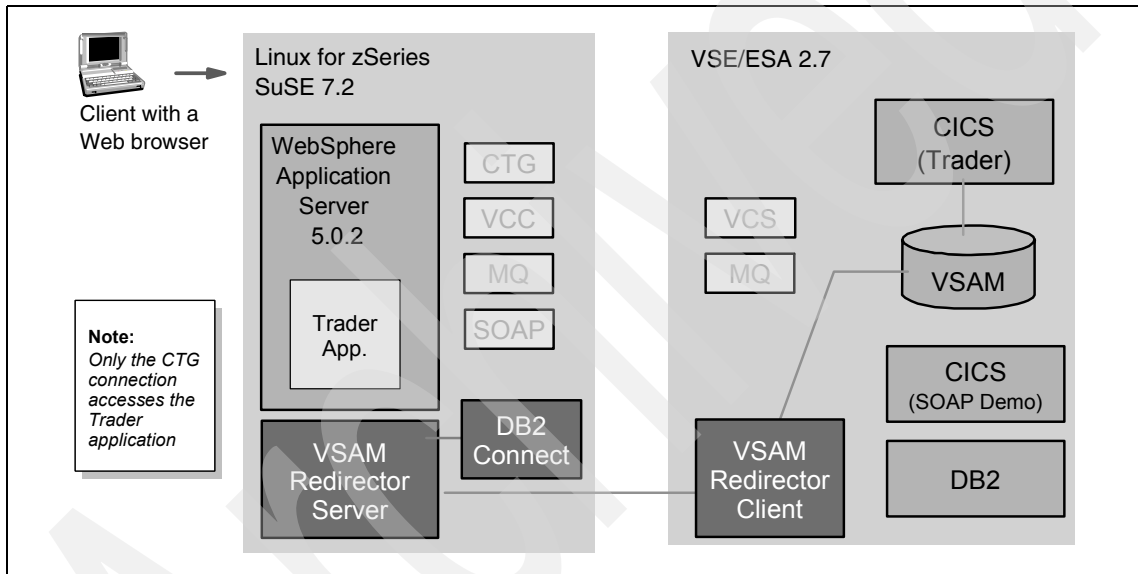


Figure 12-1 VSAM Redirector connector overview

The VSAM Redirector connector can be used in the following ways:

- ▶ Redirect access to VSAM to a remote data store, for example a relational database:
 - Read from a VSAM file but write to a DB2 database.
 - Read and write from and to a DB2 database, having just a dummy VSAM file on VSE.
- ▶ Synchronize VSAM data with a remote database

Read and write from and to VSAM but propagate all write operations also to a remote database. Other applications can access the DB2 data, which is always in sync with the VSAM data.

A detailed description of all configuration parameters can be found in the book *VSE/ESA e-business Connectors User's Guide*, SC33-6719.

All connector components, online books and documentation can be downloaded from:

<http://www-1.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/>

12.2 Client-server components

The VSAM Redirector has a server and a client component. The client is part of the VSE host side, so no installation is necessary. The server part is a stand-alone Java application, which has to be installed on a remote Java-enabled platform. This can be the system where the database runs, or a middle-tier platform between VSE and the database.

Important: Make sure that your Redirector client and server parts always have the same APAR level. If this is not the case and there were internal protocol changes between the client and server, you might experience strange effects, including hang situations.

There is a utility to access a DB2 or Oracle database in order to create tables that are used to receive redirected VSAM data and to hold control information about the VSAM data structure. This utility is part of the Redirector server installation and can be started with the batch file `create.bat`, respectively `create.sh` shell script. The utility uses JDBC to define the tables in the target database, so it needs the related jar file, which implements the database-specific JDBC driver. We recommend that you always use the original jar file that comes with your database (DB2 or Oracle).

Figure 12-2 shows a more detailed picture of the data flow using the VSAM Redirector connector.

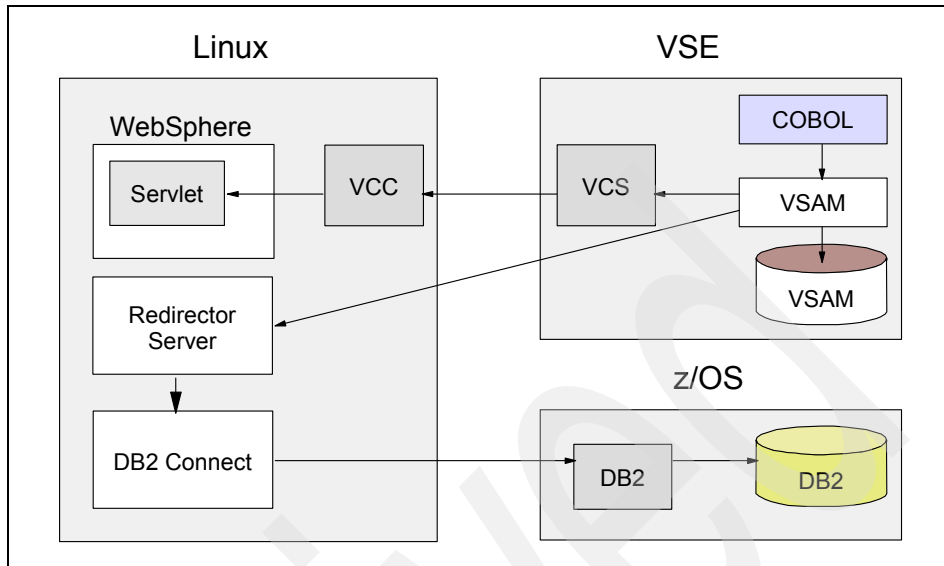


Figure 12-2 VSAM Redirector scenario

A servlet uses the VSE Connector Client (VCC) as a Websphere resource adapter to connect to the VSE Connector Server (VCS) on VSE to access a VSAM file. This VSAM file is updated by a COBOL program. Because write access to the VSAM file is redirected to the Redirector Server on Linux, all updates are made synchronously in a DB2 table on z/OS. So you will always have real-time data synchronization from VSAM to DB2.

One important point of configuring the Redirector is to decide upon the redirection mode you want to use.

- ▶ Do you want to work with data that resides on another platform? In this case, the VSAM file will be just a dummy file without any data. You always access data on the other platform.
- ▶ Do you want to synchronize an existing VSAM file with any kind of data store on another platform? In this case each initial VSAM request will result in both a VSAM request and a redirected request. So especially write requests will be performed on both sides: VSAM on VSE and any database on a remote platform.

Refer to the *VSE/ESA e-business Connectors User's Guide*, SC33-6719, for detailed information of the Redirector configuration parameters.

12.3 Installing the Redirector server

The following sections describe how to download, install, configure, and start the Redirector Server, which is independent of the redirection scenario. All information about which file to be redirected and which parameters to use is specified later when configuring the Redirector client.

12.3.1 Downloading the Redirector server

The VSAM Redirector server is shipped with VSE and can be downloaded from either of the following:

- ▶ PRD1.BASE, member IESVSMRD.W

Download the member in binary, rename the file extension to zip, and open it with any ZIP-tool, like winzip or pkzip. There are four files contained in the zip file:

- install.class - Contains the product code and install wizard
- install.bat - Install batch file for Windows
- install.cmd - Install batch file for OS/2 or Windows NT
- install.sh - Install batch file for Unix/Linux

- ▶ The Internet

<http://www-1.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/>

12.3.2 Installing the Redirector server

The Redirector server is installed by running the applicable install batch file for your workstation. You get prompted with some standard install wizard dialogs. There is a “silent install” option (/p), which can be used when there is no graphics support. We added the /p option for the non-GUI mode installation, as shown in Example 12-1.

Example 12-1 Use silent install for Redirector server on Linux for zSeries

```
#!/bin/sh
CLASSPATH=.:$CLASSPATH
java install /p
```

After installing the Redirector server we had to provide the run.sh and create.sh scripts with the correct Linux file permissions:

```
chmod 744 run.sh
```

Running this script starts the Redirector server. The server accepts the following commands:

status	Shows the status of the server
stop x all	Stops client with number x (shown in status) or stops <i>all</i> clients
quit	Stops all clients and exits server

12.3.3 Configuring the Redirector server

Figure 12-3 shows a simplified overview of the involved Redirector components. There are two points where configuration and implementation effort is necessary:

- ▶ Configuring the Redirector server and handler on the Linux for zSeries side
- ▶ Configuring the Redirector client on the VSE side

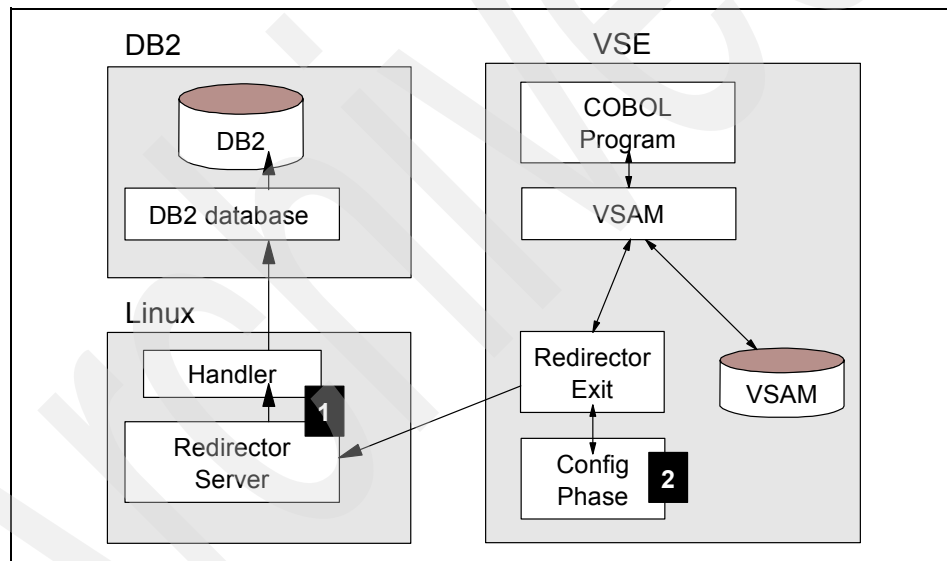


Figure 12-3 Redirector configuration and implementation

This section describes the configuration of the Redirector server. The configuration of the Redirector client is described later in 12.4, “General setup using the HtmlHandler” on page 300, and 12.5, “Special setup for Trader using the DB2 handler” on page 309.

Configuring the Redirector server consists of two steps:

1. Setting up the Redirector server properties file. We did not change anything here. Example 12-2 shows the default properties.

Example 12-2 Setting up the Redirector server's properties file

```
#VSAMRedirectorServer

#print messages (on) or do not print (off)
messages=on

#port where the RedirectorServer listens
listenport=2387

#number of maximum connections allowed
maxconnections=256

#codepage translator class
codepagetranslator=com.ibm.vse.server.DefaultTranslator

#enter the tracelevel, 0 for no trace, 1 is normal, 2 is extended tracing
tracelevel=2
```

Note: In a production environment you should set `messages = off` and `tracelevel = 0`.

2. Implementing your VSAM request handler, which defines the logic for accessing the remote file system or database.

When using the IBM-provided HTML handler, you have to modify the Java code to reflect the data layout of your redirected VSAM records. When using the IBM-provided DB2 handler, there is normally no need to change any Java code.

12.3.4 Starting the Redirector server

The Redirector server is started via the `run.sh` script. There is no GUI. The server issues messages like those shown in Example 12-4 while starting up, and then waits for connections.

The `run.sh` script needs the `db2java.zip` file, which implements the DB2-provided JDBC driver, in its classpath. You should always use the `db2java.zip` file that comes with your DB2 installation. We copied the file into the Redirector install directory and modified the `run.sh` script as shown in Example 12-3.

Example 12-3 Modify run.sh scrip

```
export CLASSPATH=.:VsamRedir.jar:db2java.zip:$CLASSPATH
java com.ibm.vse.redirector.VSAMRedirectorServer
```

Now the Redirector server can be started. You should see output similar to that shown in Example 12-4.

Example 12-4 Starting the Redirector server

```
linux11:/opt/vsereidir # ./run.sh
Licensed Materials - Property of IBM
(C) Copyright IBM Corp. 1998, 2000. All Rights Reserved.

US Government Users Restricted Rights -
Use, duplication or disclosure restricted by
GSA ADP Schedule Contract with IBM Corp.

VSAMRedirectorServer starting...
Nov 10, 2003 9:08:35 AM - Listening socket created on port 2387
Nov 10, 2003 9:08:36 AM - Waiting for connections...
Enter 'quit' to stop the server
```

Now that the Redirector server is started on Linux, the next step is to get to the VSE side and decide which redirection scenario to use. In a first step we tested the Redirector setup and network connections with the IBM-provided simple HTML handler example.

12.4 General setup using the HtmlHandler

This section shows how to set up and run the IBM-provided Redirector sample, which redirects VSAM access into an HTML file on another platform. We will use a simple IDCAMS REPRO job as a VSAM application, which writes to the target HTML file.

Figure 12-4 shows the scenario of a REPRO job copying VSAM data from a source file via the Redirector client (VRC) into an HTML file on a Windows PC.

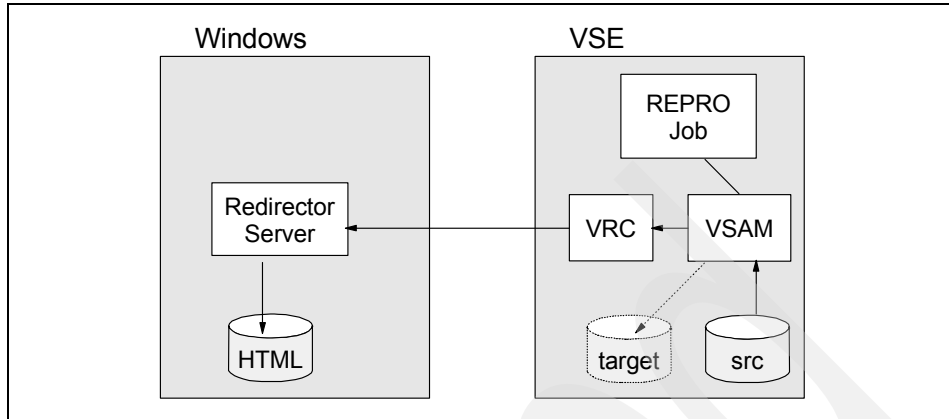


Figure 12-4 Redirector HTML handler scenario

Configuring the Redirector client consists of providing information of which VSAM files will be redirected to which remote database or file system.

Setting up the Redirector client on VSE and testing the setup with a REPRO job consists of the following steps:

- ▶ Step 1: Decide which VSAM source file to use.
- ▶ Step 2: Create a new VSAM file as the target of the REPRO operation.
- ▶ Step 3: Configure and activate the Redirector client exit via job skeleton SKRDCFG in ICCF library 59, which allows you to specify all necessary parameters. The job consists of several steps, which are explained below.
- ▶ Step 4: Modify the Java code of the IBM-provided HTML handler to reflect the specific VSAM record layout.
- ▶ Step 5: Create the IDCAMS REPRO job.
- ▶ Step 6: Run the REPRO job.
- ▶ Step 7: Check the contents of the HTML file.

The following sections describe each step in detail.

12.4.1 Step 1: Decide on the VSAM source file

For our test we took the TRADER.COMPANY file as the source file.

12.4.2 Step2: Create a new file as target file

Now we created a second VSAM file to be used as the target of the REPRO operation. As we will use redirection mode OWNER=REDIRECTOR, this file is never really accessed, but VSAM needs the control blocks.

Notes: There must at least be one dummy record in the redirected file if it will be opened for READ. You may add a record with DITTO or using the VSE Navigator. In our case, the target file was only opened for WRITE. The target VSAM file must have the same characteristics as the source file, otherwise it will not work.

Example 12-5 Define dummy target file for REPRO

```
* $$ JOB JNM=DEFPCPY,CLASS=0,DISP=D,NTFY=YES
// JOB JSCH DEFINE FILE
// EXEC IDCAMS,SIZE=AUTO
DEFINE CLUSTER ( -
    NAME (TRADER.COMPANY.COPY                ) -
    CYLINDERS(2          2          ) -
    SHAREOPTIONS (1) -
    RECORDSIZE (80      80      ) -
    VOLUMES (DOSRES SYSWK1 ) -
    NOREUSE -
    INDEXED -
    FREESPACE (15 7) -
    KEYS (20 0 ) -
    NOCOMPRESSED -
    TO (99366 )) -
    DATA (NAME (TRADER.COMPANY.COPY.@@      ) -
    CONTROLINTERVALSIZE (4096 )) -
    INDEX (NAME (TRADER.COMPANY.COPY.@I@    )) -
    CATALOG (VSESP.USER.CATALOG            )
IF LASTCC NE 0 THEN CANCEL JOB
/*
// OPTION STDLABEL=ADD
// DLBL TRCOMC,'TRADER.COMPANY.COPY',,VSAM,          X
    CAT=VSESPUC
/*
// EXEC IESVCLUP,SIZE=AUTO
A TRADER.COMPANY.COPY                                TRCOMC VSESPUC
/*
/&
* $$ EOJ
```

This file is now referenced in the next step, where the Redirector exit is configured.

12.4.3 Step 3: Configure and activate Redirector client exit

The Redirector client setup is mainly done by modifying and submitting the job SKRDCFG, which is contained in ICCF library 59. It consists of the following job steps, which are all contained in the SKRDCFG job.

1. Assemble and link the Redirector config phase.

Example 12-6 Configure Redirector client

```

* $$ 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='TRADER.COMPANY.COPY',                   X
      EXIT='IESREDIR',                                  X
      OWNER=REDIRECTOR,                                 X
      IP='9.12.6.182',                                  X
      HANDLER='com.ibm.vse.htmlhandler.HtmlHandler',   X
      OPTIONS=' '

*
END
/*
// IF $MRC GT 4 THEN
// GOTO NOLINK
// EXEC LNKEDT,PARM='MSHP'
/. NOLINK
/*
...

```

In this example, the redirection mode is set to REDIRECTOR, which means that all requests are made against the redirected data location. The VSAM file itself is just a dummy file. So the REPRO job will just copy the data from a

source file into the target HTML file. The OPTIONS string is empty, because the HTML handler does not need any options. The referenced IP address belongs to a Windows PC.

Note: For an overview of the Redirector options please refer to the *VSE/ESA e-business connectors User's Guide, SC33-6719*.

2. Load config phase IESRDCFG into the SVA.

Example 12-7 Load config phase into SVA

```
// LIBDEF *,SEARCH=PRD2.CONFIG
SET SDL
IESRDCFG,SVA
/*
```

3. Copy the IESVEX01 phase into PRD2.CONFIG as member IKQVEX01.

Example 12-8 Copy IESVEX01 phase into PRD2.CONFIG

```
// EXEC LIBR,PARM='MSHP'
CONNECT S=PRD1.BASE:PRD2.CONFIG
COPY IESVEX01.PHASE:IKQVEX01.PHASE REPLACE=YES
/*
```

4. Load IKQVEX01 into SVA. This step should be done only once.

Example 12-9 Load IKQVEX01 into SVA

```
// LIBDEF *,SEARCH=PRD2.CONFIG
SET SDL
IKQVEX01,SVA
/*
```

5. Load IESVRDANC phase into the SVA. This step should be done only once.

Example 12-10 Load IESVRDANC phase into the SVA

```
// LIBDEF *,SEARCH=PRD2.CONFIG
SET SDL
IESRDANC,SVA
/*
```

6. Register the current Redirector phase.

Example 12-11 Register Redirector phase

```
// LIBDEF *,SEARCH=PRD1.BASE
// EXEC IESRDLDA
```

/*

12.4.4 Step 4: Modify the HtmlHandler Java source

Before running the IBM-provided HTML handler example, we had to modify and recompile the Java source of the handler to reflect the VSAM record layout.

1. Modify the IBM-provided sample HTML handler.

The sample implementation `HtmlHandler.java` is located in subdirectory `com/ibm/vse/htmlhandler`. This Java program is called by the Redirector server whenever it receives redirected VSAM data that is destined for this handler. Because it reflects the structure of the received VSAM records, it has to be changed in order to work with a specific record layout.

The first method that needs to be changed is `open()`. This method is called when a Redirector client connected to the server and is sending a file open request.

Example 12-12 Changed method `open()` in `HtmlHandler`

```
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 TRADER.COMPANY cluster:</h2>");
        htmloutput.write(
            "<table border><tr><th>Company</th><th>Share Price</th>" +
            "<th>Unit Value 7 Days</th><th>Unit Value 6 Days</th></tr>");
    }
    catch(Exception ex)
    {
        System.out.println("Error creating output file!" + ex);
    }
    ...
}
```

The second method that needs to be changed is the method request(). This method is then called for each received VSAM record. The record data can be retrieved from the VSAMRequestInfo instance. Here, we do not consider all columns of the VSAM record. Let us just look at the first ones.

Example 12-13 Changed method request() in HtmlHandler

```
public void request(VSAMRequestInfo requestInfo,int stringID)
throws VSAMRequestException
{
    String company = "";
    int share_price = 0;
    int unit7 = 0;
    int unit6 = 0;
    String sout = null;

    if (requestInfo.isINSERT () )
    {
        company = requestInfo.getString(0, 20);
        share_price = requestInfo.getNumber(20, 4);
        unit7 = requestInfo.getNumber(24, 4);
        unit6 = requestInfo.getNumber(28, 4);

        // now output data to html file
        sout = "<tr><td>" + company + "</td><td>" + share_price + "</td><td>"
            + unit7 + "</td><td>" + unit6 + "</td></tr>";
        try {
            htmloutput.write(sout);
            htmloutput.newLine();
        }
        catch (Exception ex)
        {
            ...
        }
    }
    ...
}
```

2. Recompile the HtmlHandler.

We created a simple Windows batch file to recompile the handler. Put a pause statement at its end so that it does not close the window immediately when running it from the Windows Explorer. This will give you a chance to view probable compile errors.

Example 12-14 Recompile the Htmlhandler

```
set classpath=.;VsamRedir.jar;%classpath%
javac com\ibm\vse\htmlhandler\HtmlHandler.java
```

pause

12.4.5 Step 5: Create a sample VSAM application

This job represents our VSAM application, which accesses the TRADER.COMPANY file. Running the job just copies each record into an HTML file on the Windows platform.

Example 12-15 Create sample VSAM application (REPRO job)

```
* $$ JOB JNM=REPRO,CLASS=A,DISP=D,NTFY=YES
// JOB JSCH COPY FILE
// DLBL COPYIN,'TRADER.COMPANY',,VSAM, X
      CAT=VSESPUC
// DLBL COPYOUT,'TRADER.COMPANY.COPY',,VSAM, X
      CAT=VSESPUC
// EXEC IDCAMS,SIZE=AUTO
REPRO INFILE (COPYIN) -
      OUTFILE (COPYOUT) -
      NOREUSE
/*
/&
* $$ EOJ
```

12.4.6 Step 6: Run the REPRO job

On the VSE side, the job just runs like it would run without any Redirector functionality.

Example 12-16 Running the REPRO job

```
BG 0001 1Q47I  BG REPRO 01144 FROM (JSCH) , TIME=14:33:56
BG 0000 // JOB JSCH COPY FILE
      DATE 11/11/2003, CLOCK 14/33/56
BG 0000 EOJ JSCH  MAX.RETURN CODE=0000
      DATE 11/11/2003, CLOCK 14/33/59, DURATION 00/00/03
```

On the Windows side, the server issues the messages shown in Example 12-17.

Example 12-17 Redirector server side when receiving data

```
C:\vseredir>java com.ibm.vse.redirector.VSAMRedirectorServer
Licensed Materials - Property of IBM
(C) Copyright IBM Corp. 1998, 2000. All Rights Reserved.
```


US Government Users Restricted Rights -
Use, duplication or disclosure restricted by

GSA ADP Schedule Contract with IBM Corp.

```
VSAMRedirectorServer starting...
Nov 11, 2003 9:37:38 AM - Listening socket created on port 2387
Enter 'quit' to stop the server
Nov 11, 2003 9:37:38 AM - Waiting for connections...
Nov 11, 2003 9:37:43 AM - Client connection request from 9.152.82.82
Nov 11, 2003 9:37:43 AM - Client has been accepted.
Nov 11, 2003 9:37:43 AM - Connection has been accepted from 9.152.82.82
Now receiving records:
....
Ready, 4 records received.
HTML file created: 'output.html'
Nov 11, 2003 9:37:45 AM - Connection has been terminated from 9.152.82.82
Nov 11, 2003 9:37:45 AM - Client has been disconnected.
```

12.4.7 Step 7: Check the HTML output file.

The generated HTML file looks like the panel in Figure 12-5.



Company	Share Price	Unit Value 7 Days	Unit Value 6 Days
Casey_Import_Export	79	59	63
Glass_and_Luget_Plc	19	17	22
Headworth_Electrical	124	141	138
IBM	163	157	156

Figure 12-5 Output from redirected trader.company file

It shows only these columns, which we have considered in the HTML handler.

The next chapter describes the use of the more complex DB2 handler and how to redirect VSAM access into a DB2 database on z/OS via DB2 Connect on Linux for zSeries.

12.5 Special setup for Trader using the DB2 handler

Setting up a DB2 handler for the Redirector involves some additional steps, which were not necessary when using the simple HTML Handler. Basically, the Redirector needs at least two tables in the DB2 database:

1. A map table, which contains the record structure of the redirected VSAM file
2. A data table, which receives the redirected VSAM data

The following section shows how to create these DB2 tables.

12.5.1 Providing VSAM map definition

Creating the DB2 tables should be done with the IBM-provided CreateDB2Tables program, which is run using the create.sh script. The Java source code is given in the directory com/ibm/vse/db2handler/create.

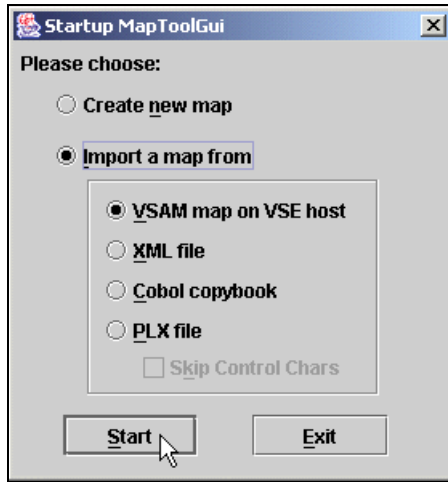
To create the map table, the tool needs an XML file containing the VSAM map definitions of the cluster that will be redirected. You can write this XML file yourself using a simple text editor. But when there is already a map definition on the VSE host, you can use the VSE/ESA MapTool to download the VSAM map into an XML file.

Notes: The database server, against which the **CreateDB2Tables** program is run, can be any DB/2 server, but it must be a UDB version, so VSE DB/2 is not supported.

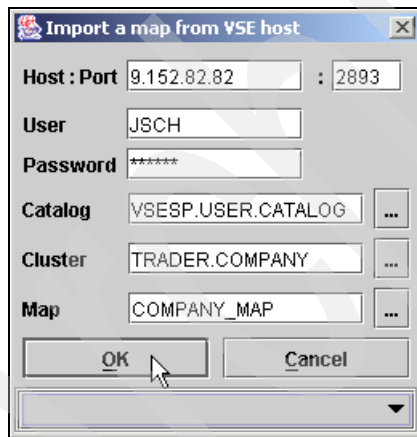
It is very important to map at least the **KEY** of the VSAM record to a database field. The key field must be of type String.

Make sure that the VSE Connector Server is started on VSE in non-SSL mode. The map tool needs a connection to the VSE Connector Server on VSE in order to download the VSAM map.

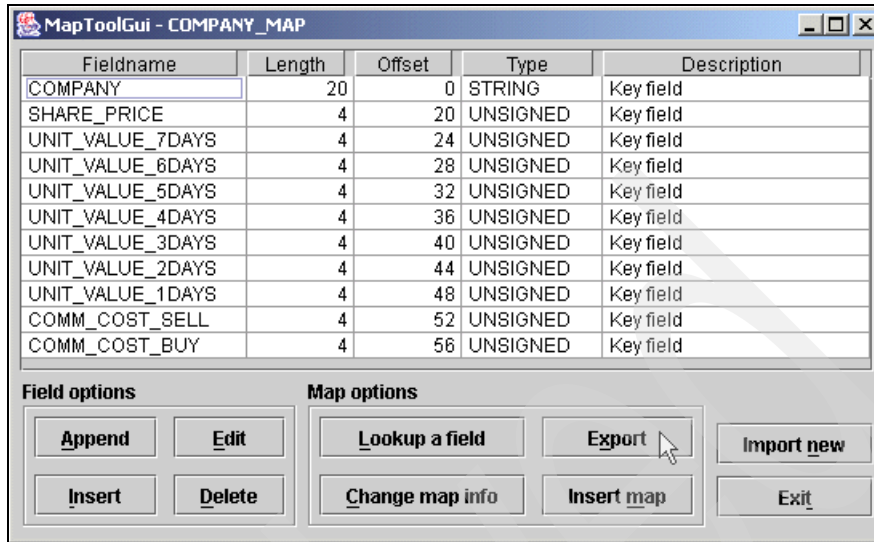
After starting the maptool, you can choose between several options. Select **Import map** from VSAM map on VSE host and click **Start**.



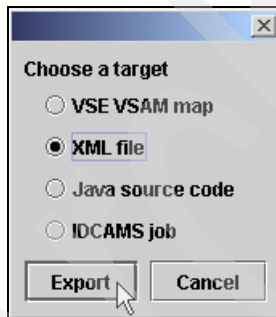
Enter the required parameters or select them from a list by clicking the buttons to the right of the text fields for Catalog, Cluster, or Map. Then click **OK**.



After connecting to the VSE system and downloading the map information, you see the data fields in the above dialog box. To export the map to XML click **Export**.



In the next box select XML file and click **Export**. A file dialog allows you to choose a file name and a directory in which to store the XML file.



Note: An extra database column UNIQRBACNT is appended to the defined fields. This column is defined to count the records in the table. It is used to compute the VSAM RBA value that is returned to VSE VSAM on INSERT, UPDATE, GET, POINT and DELETE requests. This ensures that each record will have a unique RBA value.

Now as the map XML file is created, you have to make some preparations in order to run the IBM-provided Java program, which creates the two DB2 tables.

12.5.2 Populating the sample VSAM files

For the Trader scenario, we have to populate only the trader.company file. The trader.customer file is initially left empty.

Please refer to 10.5.5, “Populating the sample VSAM files” on page 249, for details about several ways of populating the sample VSAM file. At this point we assume that the four sample records are contained in the trader.company file.

12.5.3 Preparing for creating DB2 tables

Before you can run the create.sh file you have to make sure that db2java.zip, which comes with DB2, is contained in your local classpath.

You should always use the db2java.zip file that comes with your DB2 installation. We used the db2java.zip file from the DB2 Connect installation on Linux for zSeries in /opt/IBM/db2/V8.1/java and copied it into the Redirector's install directory and modified the create.sh script to point directly to the file.

The xerces.jar file, which is needed for XML-related functions, was part of the VSAM Redirector installation.

We modified the classpath in create.sh in order to access the two jar files, as shown in Example 12-18.

Example 12-18 Modify create.sh script

```
export CLASSPATH=.:xerces.jar:db2java.zip:$CLASSPATH
java com.ibm.vse.db2handler.create.CreatedBTables
```

12.5.4 Creating DB2 tables

Running the create.sh file now prompts for the required information to access the DB2 database. The ClassNotFoundException about the Oracle JDBC driver can be ignored here, because we use DB2 and did not have any Oracle code installed.

Example 12-19 shows the process for creating the tables in DB2. We ran this script on Linux for zSeries, so that the DB URL points to the local DB2 Connect installation on Linux. The tables are then created on DB2 on z/OS.

Example 12-19 Run create.sh script

```
linux11:/opt/vseredir # ./create.sh
...
*** Error while registering the JDBC drivers: ***
java.lang.ClassNotFoundException: oracle.jdbc.driver.OracleDriver
```

```
...
XML filename : trader.company.xml
DB url       : jdbc:db2:trader
DB user      : linux1
WARNING: Due to the internals of Java, password will be visible while entering!
DB password  : dbpwd
DB table name : trader.company_redirected
map table name: trader.company_maps
map name     : company_map
DB system, 1 = DB/2 ; 2 = Oracle : 1
Target database system is DB/2.
Importing XML file ... got 11 fields from XML file.
Connect to database...
```

Now the MapInfo table will be created with name 'trader.company_maps'.

```
...
```

Do you want to proceed (Yes/Skip/Exit)? **yes**

```
...
```

Now the data table will be created with name 'trader.company_redirected'.

```
...
```

Do you want to proceed (Yes/Skip/Exit)? **yes**

```
...
```

Please specify for which access this database should be prepared.

- 1 KSDS without AIX
- 2 KSDS with AIX
- 3 ESDS without AIX
- 4 ESDS with AIX
- 5 Exit

Please choose: **1**

Please enter the name of the field which maps the whole VSAM key.
This field must have the same offset and length of the VSAM key field.

Primary key field: **COMPANY**

SQL string to be sent to database:

```
CREATE TABLE trader.company_redirected ( UNIQRBACNT INTEGER NOT NULL GENERATED
BY DEFAULT AS IDENTITY ( START WITH 0, INCREMENT BY 1 ) , COMPANY CHARACTER
(20) NOT NULL , SHARE_PRICE INTEGER , UNIT_VALUE_7DAYS INTEGER ,
UNIT_VALUE_6DAYS INTEGER , UNIT_VALUE_5DAYS INTEGER , UNIT_VALUE_4DAYS
INTEGER , UNIT_VALUE_3DAYS INTEGER , UNIT_VALUE_2DAYS INTEGER ,
UNIT_VALUE_1DAYS INTEGER , COMM_COST_SELL INTEGER , COMM_COST_BUY INTEGER ,
PRIMARY KEY (COMPANY) , UNIQUE (UNIQRBACNT) )
```

Press ENTER to continue.

Creating indexes for NON-UNIQUE fields.

Ready.

```
linux11:/opt/vseredir #
```

Note: We recommend that you enter the name of the primary key field case sensitive, exactly as it is defined in the XML file. Because we used the VSAM map tool to create the XML file, all field names were created in uppercase letters.

We had the problem that the table definition was incomplete after completing the create tables step. The DB2 message was:

```
SQL0540N The definition of table "TRADER.COMPANY_REDIRECTED" is incomplete because it lacks a primary index or a required unique index. SQLSTATE=57001
```

So we used the following statement to create an index for the UNIQRBACNT field, which is used internally by the DB2 handler:

```
CREATE TYPE 2 UNIQUE INDEX TRADER.COMPANY_REDIR_X3 ON TRADER.COMPANY_REDIRECTED (UNIQRBACNT)
```

This solved the problem.

After creating the DB/2 tables, the Linux part of the preparation is done.

12.5.5 Setting up the VSE side

On the VSE side you now have to set up the Redirector config phase for redirecting the VSAM file TRADER.COMPANY using the DB2 handler.

Example 12-20 shows the relevant part of the SKRD CFG job that has to be changed in order to redirect the file.

The HANDLER parameter now specifies the DB2 handler. The OPTIONS string now contains relevant information to access the DB2 database. The IP address belongs to our Linux for zSeries installation.

Example 12-20 Setup Redirector config phase for DB2 handler

```
*
IESRDENT CATALOG='VSESP.USER.CATALOG', X
          CLUSTER='TRADER.COMPANY.COPY', X
          EXIT='IESREDIR', X
          OWNER=REDIRECTOR, X
          IP='9.12.9.12', X
          HANDLER='com.ibm.vse.db2handler.DB2Handler', X
          OPTIONS='dburl=jdbc:db2:trader;dbuser=linux1; X
          dbpassword=dbpwd;dbtable=trader.company_redirected; X
```

```
mactable=trader.company_map;map=company_map'
```

*

Now we cataloged the config phase again by submitting the SKRDCFG job. There is no need to modify the IBM-provided DB2 Handler Java code.

12.5.6 Testing the setup

For a quick test we can now run our original IDCAMS REPRO job to copy the TRADER.COMPANY file into TRADER.COMPANY.COPY, which will result in redirecting the copied VSAM data into the DB2 table trader.company_redirected on DB2.

- ▶ Make sure that the changed SKRDCFG job has been submitted.
- ▶ Make sure that the Redirector server is started on Linux.
- ▶ Run the REPRO job on VSE.

You should get similar output in the Redirector server console as shown in Example 12-21.

Example 12-21 Redirector server, receiving data from VSE

```
linux11:/opt/vseredir # ./run.sh
...
VSAMRedirectorServer starting...
Nov 20, 2003 4:09:59 PM - Listening socket created on port 2387
Nov 20, 2003 4:09:59 PM - Waiting for connections...
Enter 'quit' to stop the server
Nov 20, 2003 4:10:21 PM - Client connection request from 9.152.82.82
Nov 20, 2003 4:10:21 PM - Client has been accepted.
Nov 20, 2003 4:10:21 PM - Connection has been accepted from 9.152.82.82
.o0 DefaultRequestHandler.initialize()
.o0 DefaultRequestHandler.open()
*** Could not register a JDBC driver: ***
java.lang.ClassNotFoundException: oracle.jdbc.driver.OracleDriver
[trader.company_redirected] Configured for database system: IBM DB/2
[trader.company_redirected] **** Initialize BEGINS >>>>
[trader.company_redirected] Get DB connection...
[trader.company_redirected] Get map info from DB table...
[trader.company_redirected] **** getMapInfo BEGINS >>>>
[trader.company_redirected] --- Database query #0 ---
[trader.company_redirected] Retrieving mapping from trader.company_maps
(map=company_map)
[trader.company_redirected] Retrieved 11 fields.
[trader.company_redirected] Using database table: trader.company_redirected
[trader.company_redirected] VSAM cluster - Key position,length: 0 , 20
[trader.company_redirected] This is a KSDS cluster (KEY+ADR access).
[trader.company_redirected] Field used for KEY access: COMPANY
```

```

[trader.company_redirected] Prepare the statements...
[trader.company_redirected] Committing after each INSERT (Note: UPD+DEL is
auto-committed).
[trader.company_redirected] Set position to first record...
[trader.company_redirected] --- Database query #1 ---
[trader.company_redirected] Could not find any record, positioning is not
done.
[trader.company_redirected] **** Initialize READY >>>>
.o0 DefaultRequestHandler.request(), STRID=1

```

```

[trader.company_redirected] RequestInfo:
INSERT
KEY:Y ADR:n | DIR:n SEQ:Y SKP:n | NUP:Y UPD:n NSP:n
FWD:Y BWD:n | KEQ:n KGE:Y GEN:n | LRD:n | keylen=(no key)
[trader.company_redirected] **** handleRequests BEGINS >>>>
[trader.company_redirected] INSERT request
[trader.company_redirected] **** perform_INSERT BEGINS >>>>
[trader.company_redirected] **** splitRecord BEGINS (VSAM record -> SQL
statement) >>>>
[trader.company_redirected] getString:COMPANY = 'Casey_Import_Export '
[trader.company_redirected] getInt:SHARE_PRICE = 79
[trader.company_redirected] getInt:UNIT_VALUE_7DAYS = 59
[trader.company_redirected] getInt:UNIT_VALUE_6DAYS = 63
[trader.company_redirected] getInt:UNIT_VALUE_5DAYS = 65
[trader.company_redirected] getInt:UNIT_VALUE_4DAYS = 70
[trader.company_redirected] getInt:UNIT_VALUE_3DAYS = 72
[trader.company_redirected] getInt:UNIT_VALUE_2DAYS = 78
[trader.company_redirected] getInt:UNIT_VALUE_1DAYS = 77
[trader.company_redirected] getInt:COMM_COST_SELL = 10
[trader.company_redirected] getInt:COMM_COST_BUY = 7
[trader.company_redirected] --- Database query #2 ---
[trader.company_redirected] --- Database query #3 ---
returning RBA: 0
.o0 DefaultRequestHandler.finished(), STRID=1
...
.o0 DefaultRequestHandler.close()
[trader.company_redirected] **** cleanup BEGINS >>>>
-----
.o0 DefaultRequestHandler.cleanup()
Nov 20, 2003 4:10:23 PM - Connection has been terminated from 9.152.82.82
Nov 20, 2003 4:10:23 PM - Client has been disconnected.

```

12.5.7 Checking the target DB2 table

You may now check the contents of the target DB2 table using native DB2 commands.

Example 12-22 Check contents of target DB2 table

```
db2 => select * from trader.company_redirected
```

UNIQRBACNT	COMPANY	SHARE_PRICE	UNIT_VALUE_7DAYS	UNIT_VALUE_6DAYS	...
0	Casey_Import_Export	79	59	63	...
1	Glass_and_Luget_Plc	19	17	22	...
2	Headworth_Electrical	124	141	138	...
3	IBM	163	157	156	...

4 record(s) selected.

```
db2 =>
```

The first table column, UNIQRBACNT, is used by the Redirector internally.

12.5.8 Trader scenario with redirected VSAM files

Now as we are redirecting the VSAM files, which are used by Trader, the scenario looks as shown in Figure 12-6.

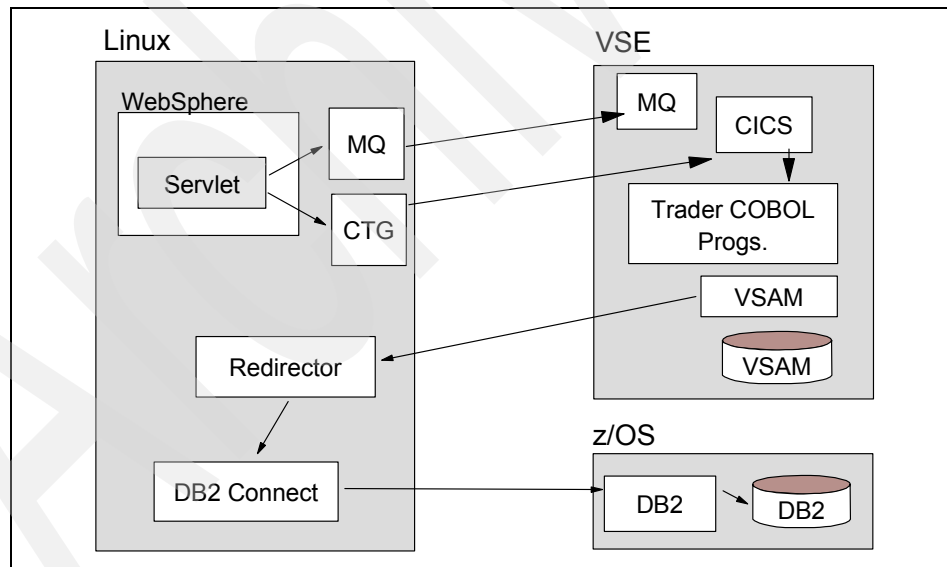


Figure 12-6 Trader scenario with redirected VSAM files

The servlet uses the MQ or CTG connector to call an online COBOL program on VSE, which operates on the Trader VSAM files. All operations to the trader.company file are redirected to the DB2 database running on z/OS.

Archived



A

VSE/ESA code samples

Sample Java SOAP client program

Following is the complete Java source code of the sample SOAP client program used to test the SOAP connection to CICS. The source has been taken from the Apache SOAP sample and has been slightly modified to work with the VSE SOAP service.

Example: A-1 Sample SOAP client Java program

```
/*
 * The Apache Software License, Version 1.1
 *
 *
 * Copyright (c) 2000 The Apache Software Foundation. All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in
 * the documentation and/or other materials provided with the
 * distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 * if any, must include the following acknowledgment:
 * "This product includes software developed by the
 * Apache Software Foundation (http://www.apache.org/)."
 * Alternately, this acknowledgment may appear in the software itself,
 * if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "SOAP" and "Apache Software Foundation" must
 * not be used to endorse or promote products derived from this
 * software without prior written permission. For written
 * permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 * nor may "Apache" appear in their name, without prior written
 * permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
```

```

* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
* USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
* ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
* OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
* OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
* SUCH DAMAGE.
* =====
*
* This software consists of voluntary contributions made by many
* individuals on behalf of the Apache Software Foundation and was
* originally based on software copyright (c) 2000, International
* Business Machines, Inc., http://www.apache.org. For more
* information on the Apache Software Foundation, please see
* <http://www.apache.org/>.
*
* This sample file was changed to call the VSE/ESA SOAP service.
*/

import java.io.*;
import java.net.*;
import java.util.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;

public class GetQuote
{
    public static void main (String[] args) throws Exception
    {
        if (args.length != 2
            && (args.length != 3 || !args[0].startsWith ("-")))
        {
            System.err.println ("Usage: java " + GetQuote.class.getName () + "
[-encodingStyleURI] SOAP-router-URL symbol");
            System.exit (1);
        }

        // Process the arguments.
        int offset = 3 - args.length;
        String encodingStyleURI = args.length == 3
            ? args[0].substring(1)
            : Constants.NS_URI_SOAP_ENC;
        URL url = new URL (args[1 - offset]);
        String symbol = args[2 - offset];

        // 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);

        // make the call: note that the action URI is empty because the
        // XML-SOAP rpc router does not need this. This may change in the
        // future.
        Response resp = call.invoke (/* router URL */ url, /* actionURI */ ""
);

        // Check the response.
        if (resp.generatedFault ()) {
            Fault fault = resp.getFault ();

            System.err.println("Generated fault: " + fault);
        } else {
            Parameter result = resp.getReturnValue ();
            System.out.println (result.getValue ());
        }
    }
}

```

You compile the program using a batch file like the one shown below.

Example: A-2 Compile Java SOAP client program

```

set CLASSPATH=soap.jar;%CLASSPATH%
javac GetQuote.java

```

You run the sample using following batch file. You will have to change the VSE IP address to the one of your VSE system.

Example: A-3 Changing

```

set CLASSPATH=j2ee.jar;soap.jar;xerces.jar;mail.jar;activation.jar;.
java GetQuote http://9.164.155.95:1080/cics/CWBA/IESSOAPS IBM

```

Sample Java program to populate VSAM files on VSE

Following is the complete Java code of the program used to populate the TRADER.COMPANY file on VSE.

Example: A-4 Sample Java program to populate VSAM file

```

/*****
/*          VSE Workdesk - VSE Connector Client          */
/*****
/*
*/
/* MODULE NAME : PutData.java                               */
/*
*/
/* COPYRIGHT NOTICE: Copyright (C) 2003.  IBM Corporation. */
/*
*/
/* This file is used to demonstrate how to utilize IBM Corporation's */
/* VSE/ESA Connector Framework Java classes. */
/*
*/
/* You have a royalty-free right to use, modify, reproduce and */
/* distribute this demonstration file (including any modified */
/* version), provided that you agree that IBM Corporation */
/* has no warranty, implied or otherwise, or liability */
/* for this demonstration file or any modified version. */
/* */
/* COMPILER : */
/* *
/*     Java 1.1.6 or higher */
/*
*/
/*****
/*
*/
/* CHANGE ACTIVITY : */
/* */
/* */
/*****
package com.ibm.vse.samples;
import java.lang.*;
import java.net.*;
import java.io.*;
import java.util.*;

/* Import Common Connector Interface (CCI) classes */
import javax.resource.*;

/* Import VSE Connector classes */
import com.ibm.vse.connector.*;

public class PutData
{

```

```

public static void main(String argv[]) throws IOException, ResourceException
{
    String catalogID = "VSESP.USER.CATALOG";
    String clusterID = "TRADER.COMPANY.COPY";
    String mapname = "COMPANY_MAP";

    VSEConnectionSpec spec;
    VSESystem system;
    byte[] inputArray;
    String ipAddr = "9.152.82.82";
    String userID = "jsch";
    String password = "mypassw";

    /* 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();
    */
    try {
        spec = new VSEConnectionSpec(InetAddress.getByName(ipAddr),
            2893,userID,password);

        system = new VSESystem(spec);
    }
    catch (UnknownHostException e)
    {
        System.out.println("Unknown host : " + e);
        return;
    }

    /* Setup data ... */
    String[][] data = {
        {"Casey_Import_Export
", "0079", "0059", "0063", "0065", "0070", "0072", "0078", "0077", "010", "007"},
        {"Glass_and_Luget_Plc
", "0019", "0017", "0022", "0020", "0016", "0020", "0025", "0022", "002", "002"},

{"Headworth_Electrical", "0124", "0141", "0138", "0137", "0133", "0133", "0133", "0131"
, "012", "011"},
        {"IBM
", "0163", "0157", "0156", "0159", "0161", "0160", "0162", "0163", "010", "015"}
    };

    /* Add data ... */

```

```

        VSEVsamRecord record = new VSEVsamRecord(system, catalogID, clusterID,
mapname);
        for (int i=0;i<data.length;i++)
        {
            record.setKeyField(0, new String(data[i][0]));
            for (int k=1;k<11;k++)
                record.setField(k, new Integer((data[i][k])));
            record.add();
        }
        System.out.println("Records added to file.");
    }
}

```

You compile the program using a batch file like the one shown in the Example A-5.

Example: A-5 Compile sample Java program

```

set CLASSPATH=.;c:\vsecon\cci.jar;c:\vsecon\ibmjsse.jar;
c:\vsecon\VSEConnector.jar;%CLASSPATH%
javac com\ibm\vse\samples\PutData.java
pause

```

IBM-provided SOAP service C program

This C program is shipped with VSE in sublibrary PRD1.BASE. It is also provided with the VSE Connector Client, together with the compile and link jobs.

Example: A-6 IBM provided SOAP service C program

```

/*****
/*          VSE/ESA VSE SOAP Service          */
/*****
/*          */
/* MODULE NAME : GetQuote.c          */
/*          */
/* RELEASE : VSE/ESA Version 2.7.0          */
/*          */
/* COPYRIGHT NOTICE: Copyright (C) 2001.  IBM Corporation.          */
/*          */
/* You have a royalty-free right to use, modify, reproduce and          */
/* distribute this demonstration file (including any modified          */
/* version), provided that you agree that IBM Corporation          */
/* has no warranty, implied or otherwise, or liability          */
/* for this demonstration file or any modified version.          */
/*          */
/* COMPILER :          */

```

```

/*                                                                    */
/*      C Compiler                                                    */
/*                                                                    */
/*****                                                                    */
/*                                                                    */
/* CHANGE ACTIVITY :                                                */
/*                                                                    */
/*                                                                    */
/*****                                                                    */
/* This is a sample SOAP service.                                    */
/*                                                                    */
/* It is an implementation of a service for the GetStockQuote sample*/
/* that is provided by many SOAP projects like Apache SOAP.       */
/* This service expects a stock symbol as input value and will     */
/* return a fictitious value as the stock quote.                   */
/*                                                                    */
/* In the client code you must only change the target URI to       */
/* "urn:iessoapd:getquote" and set the URL to call the IESSOAPS */
/* service (e.g. http://9.164.123.45:1080/cics/CWBA/IESSOAPS).    */
/* The sample code in soapclnt.c is a SOAP client that can be used */
/* to call this service.                                           */
/*****                                                                    */

#include <ctype.h>
#include <string.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdarg.h>
#include "IESSOAPH.H"

/*
function to let the CICS program print debug info to the console
*/
void cicsprintf(char* fmt,...)
{
    va_list args;
    char text[256], text2[256];

    va_start(args,fmt);
    vsprintf(text,fmt,args);
    va_end(args);
    sprintf( text2, "SSRV: %s", text );

    EXEC CICS WRITE OPERATOR TEXT(text2) TEXTLENGTH(strlen(text2));
};

/*
gets the next input parameter,

```



```

    sets pName pPtr and pLen (valid until next call)
    */
int GetNextInParameter(char* inqueue, char** pName,
                      char** pPtr, int* pLen)
{
    int resp, resp2;
    SOAP_PARAM_HDR* param;
    unsigned short len;

    *pName = NULL;
    *pPtr = NULL;
    *pLen = 0;

    EXEC CICS READQ TS QUEUE(inqueue)
           SET(param) LENGTH(len) NEXT
           RESP(resp) RESP2(resp2);

    if(resp!=DFHRESP(NORMAL))
        return 4; // no more params

    if(param->type==SOAP_TYPE_STRUCT)
        return 5; // invalid type (for this service)

    *pName = (char*)&param->name;
    *pPtr = (char*)&param[1];
    *pLen = param->length-sizeof(SOAP_PARAM_HDR);

    return(0);
};

// sets the next output parameter
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);
};

/*
 * this function is called if the requested
 * SOAP method is 'getQuote'
 */
int ProcessGetQuote(char* inqueue,char* outqueue)
{
    int rc,resp,resp2;
    char* name;
    char* data;
    int len;
    unsigned short slen;
    char symbol[9];
    char *quote;

    // first get the input parameters
    cicsprintf("method getQuote");

    // get the symbol name ( first and only element that is expected
    rc = GetNextInParameter(inqueue, &name, &data, &len);
    if( rc != 0 )
        return(rc);

    // check if the provided parameter is named 'symbol'
    if(strncmp(name, "symbol", 6) !=0 )
        return(2); // invalid param name

    memset(symbol,' ',8);
    symbol[8] = '\0';
    if(len>8)
        len = 8;
    memcpy(symbol,data,len);

    // variable symbol now contains the symbol name
    cicsprintf("stock symbol = %s", symbol);

    // return a fictitious value as stock quote
    // special handling for value IBM
    if( strncmp(symbol, "IBM", 3) == 0 )
    {

```

```

        quote = "80.10";
    }
    else
    {
        quote = "100.01";
    }

    // set the stock value as output with the name 'data'
    // and type string
    rc = SetNextOutParameter(outqueue,
        "string", SOAP_TYPE_STRING,
        "data", quote, strlen(quote));
    if(rc!=0)
        return 3;

    return 0;
};

int main()
{
    SOAP_PROG_PARAM* call;
    int i,rc;

    EXEC CICS ADDRESS EIB(dfheiptr);

    cicsprintf("SOAPSERVER called");

    EXEC CICS ADDRESS COMMAREA(call);

    cicsprintf("Method = %.16s",call->method);

    // check if the SOAP method name is 'getQuote'
    if( strcmp(call->method, "getQuote", 8) == 0 )
        rc = ProcessGetQuote(call->inqueue, call->outqueue);
    else
        rc = 1;

    cicsprintf("rc = %d",rc);
    cicsprintf("SOAPSERVER finished");

    call->retcode = rc;

    return(rc);
}

```

IBM-provided include file for SOAP

This C header file is shipped with VSE in sublibrary PRD1.BASE.

Example: A-7 IBM provided include file for SOAP

```
/******  
*           VSE/ESA SOAP Interface                               *  
*****  
*           *                                                   *  
* SOURCE FILE : IESSOAPH.H                                     *  
*           *                                                   *  
*****  
*           *                                                   *  
* CONTENTS:      SOAP Interface                               *  
*           *                                                   *  
*****  
* DESCRIPTIVE NAME : SOAP Interface                           *  
*           *                                                   *  
* COPYRIGHT NOTICE : LICENSED MATERIALS - PROPERTY OF IBM   *  
*                   "RESTRICTED MATERIALS OF IBM"             *  
*                   5686-066 (C) Copyright IBM Corp. 2002.    *  
*           *                                                   *  
* NOTES :                                                   *  
* $Revision: 1.00$                                           *  
* $Date: 21.02.2002$                                         *  
* $Author: IFranzki$                                         *  
*****  
* CHANGE ACTIVITY :                                           *  
*           *                                                   *  
* FLAG REASON  RELEASE DATE ORIGIN  COMMENT                 *  
* -----      - - - - -          - - - - -                   *  
*****/  
  
#ifndef _IESSOAPH_H  
#define _IESSOAPH_H  
  
#include <stdio.h>  
#include <stddef.h>  
  
//*****  
// SOAP Server:  
//  
// The SOAP Server (IESSOAPS) is called from CICS (CWS) using the URL  
// "http://<server>:<port>/CICS/CWBA/IESSOAPS" with HTTP method POST.  
// The SOAP Server receives the XML SOAP Envelope, validates it and  
// uses the URN to call the user program or the SOAP decoder (IESSOAPD)  
// The URN must have the following format "urn:<decoder>Y:<userprog>"  
// The IBM-supplied decoder IESSOAPD converts the XML parameter tree
```

```

// into a series of structured data blocks and passes it to the
// user program via EXEC CICS LINK.
//
// SOAP Client:
//
// A SOAP Client program can either call the IBM supplied SOAP encoder
// IESSOAPE, or directly call the SOAP Client IESSOAPC. The SOAP
// decoder works similar as the SOAP Encoder, it converts the parameters
// from the user program into a XML tree and passes it to the SOAP
// client. The SOAP client establishes a HTTP Connection to the
// server and sends the XML SOAP Envelope.
//
//*****

//*****
// Interface between a user program and the encoder/decoder
//*****

// Parameters are converted by the decoder/encoder into/from
// a control block structure which describe the parameter(s) and
// contains the data. Each parameter starts with a header
// (SOAP_PARAM_HDR). This header specifies the type of parameter.
// If the parameter has type SOAP_TYPE_STRUCT the data of the
// parameter contain one or more headers plus its data.
// The parameters are stored into a CICS TS QUEUE to pass them
// from the user program to the encoder/decoder and vice versa.
typedef _Packed struct tagSOAP_PARAM_HDR
{
    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)
}
    SOAP_PARAM_HDR;
typedef SOAP_PARAM_HDR* LPSOAP_PARAM_HDR;

// 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 // hierarchical 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

//*****
// Assembler DSECT of the above structure

```

```

//
// PARAMHDR DSECT SOAP_PARAM_HDR
// NAME DS CL16 PARAMETER NAME
// TYPENAME DS CL16 DATA TYPE NAME
// LENGTH DS F LENGTH OF BLOCK (INC. HEADER)
// TYPE DS F TYPE (SEE EQUs)
// TUNSPEC EQU 0 UNKNOWN/UNSPECIFIED
// TPRIVATE EQU 1 PRIVATE TYPE
// TSTRUCT EQU 2 HIRACHICAL STRUCTURE
// TSTRING EQU 10 STRING
// TINTEGER EQU 11 INTEGER (4 BYTES)
// TSHORT EQU 12 SHORT (2 BYTES)
// TBYTE EQU 13 BYTE (1 BYTE)
// TBOOLEAN EQU 14 BOOLEAN (1 BYTE)
// TBINARY EQU 15 BINARY
//
//*****

//*****
// Interface between the IBM supplied SOAP Encoder and the user program
//*****

// The following COMMAREA is used when a user program is called by
// the IBM supplied SOAP Encoder (IESSOAPE). It specifies the
// method name, the names of the CICS TS QUEUE names for input
// and output parameter and the URL of the XML namespace which
// is used by the caller
typedef _Packed struct tagSOAP_PROG_PARAM
{
    char methodY16"; // (in) method name
    char inqueueY8"; // (in) input params
    char outqueueY8"; // (in) output params
    char namespaceurlY128"; // (in/out) private namespace url
    int retcode; // (out) return code
}
SOAP_PROG_PARAM;
typedef SOAP_PROG_PARAM* LPSOAP_PROG_PARAM;

//*****
// Assembler DSECT of the above structure
//
// PROGPARM DSECT SOAP_PROG_PARAM
// METHOD DS CL16 (IN) METHOD NAME
// INQUEUE DS CL8 (IN) INPUT QUEUE NAME
// OUTQUEUE DS CL8 (IN) OUTPUT QUEUE NAME
// NSURL DS CL128 (IN/OUT) PRIVATE NAMESPACE URL
// RETCODE DS F (OUT) RETURN CODE
//

```

```

//*****

//*****
// Interface between the SOAP Server and an encoder
//*****

// The parameter area is used when the SOAP Server (IESSOAPS)
// calls the SOAP Encoder. The SOAP Encoder can either be the
// IBM-supplied encoder IESSOAPE, or a user written program.
typedef _Packed struct tagSOAP_ENC_PARAM
{
    int action; // (in) see SOAP_ENC_ACTION_xxx
    void* userdata; // (out/in) user data
    // SOAP specific
    char* urn; // (in) urn
    int urnlen; // (in) length of the urn
    char* method; // (in) method name
    int methodlen; // (in) length of the method name
    // parameters
    struct _node* inparams; // (in) input xml tree
    struct _node* outparams; // (out) output xml tree
}
SOAP_ENC_PARAM;
typedef SOAP_ENC_PARAM* LPSOAP_ENC_PARAM;

// Values for the action field in SOAP_ENC_PARAM
#define SOAP_ENC_ACTION_INIT 0 // Initialize the encoder
#define SOAP_ENC_ACTION_CALL 1 // Process the method call
#define SOAP_ENC_ACTION_DONE 2 // Cleanup the Encoder

// Function prototype for the Encoder
typedef int (*SOAPEncoder)(LPSOAP_ENC_PARAM param);

//*****
// Interface between a user program and the IBM supplied decoder
//*****

// The following COMMAREA is used by a user program to call the
// IBM-supplied decoder (IESSOAPD)
typedef _Packed struct tagSOAP_DEC_PARAM
{
    char urlY128"; // (in) the servers url
    char methodY16"; // (in) method name
    char urnY128"; // (in) the urn
    char inqueueY8"; // (in) input queue name
    char outqueueY8"; // (in) output queue name
    char namespaceurlY128"; // (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
        // return code
        int retcode;      // (out) return code
    }
    SOAP_DEC_PARAM;
typedef SOAP_DEC_PARAM* LPSOAP_DEC_PARAM;

//*****
// Assembler DSECT of the above structure
//
// DECPARM DSECT SOAP_DEC_PARAM
// URL DS CL128 (IN) URL OF THE SERVER
// METHOD DS CL16 (IN) METHOD NAME
// URN DS CL128 (IN) THE URN
// INQUEUE DS CL8 (IN) INPUT QUEUE NAME
// OUTQUEUE DS CL8 (IN) OUTPUT QUEUE NAME
// NSURL DS CL128 (IN/OUT) PRIVATE NAMESPACE URL
// PROXYTYP DS F (IN) PROXY TYPE (SEE EQU)
// PDIRECT EQU 0 DIRECT CONNECTION, NO PROXY/SOCKS
// PPROXY EQU 1 HTTP PROXY
// PSOCKS4 EQU 2 SOCKS V4
// PSOCKS5 EQU 3 SOCKS V5
// PROXY DS CL128 (IN) HOSTNAME OR IP OF THE PROXY SERVER
// PROXPORT DS F (IN) PORT NUMBER OF THE PROXY
// USERID DS CL16 (IN) USER NAME FOR SOCKS
// PASSWORD DS CL16 (IN) PASSWORD FOR SOCKS V5 ONLY
// RETCODE DS F (OUT) RETURN CODE
//
//*****

//*****
// Interface between a user program and the SOAP Client
//*****

// The following control block contains infomations about
// the proxy or socks server that is used to connect to the server.
typedef struct __HTTP_PROXY
{
    int proxytype; // (in) proxy type (HTTP_TYPE_xxx)
    char* proxy; // (in) proxy server name
    int proxylen; // (in) length of server name
    unsigned short proxyport; // (in) port of the proxy
    char* userid; // (in) userid for socks
    int useridlen; // (in) length of the userid

```



```

        char*      password; // (in) password for socks 5
        int       passwordlen; // (in) length of the password
    }
    HTTP_PROXY;
typedef HTTP_PROXY* LPHTTP_PROXY;

// 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

// The following area is used by a user program or decoder
// to call the SOAP Client (IESSOAPC).
typedef _Packed struct tagSOAP_CLNT_PARAM
{
    int  action; // (in) action (see SOAP_CLNT_ACTION_XXX)
    void* userdata; // (in) user data
    // request info
    char* url; // (in) url of the server
    int  urlen; // (in) length of the url
    char* urn; // (in) urn of the SOAP service
    int  urnlen; // (in) length of the urn
    char* method; // (in) method name
    int  methodlen; // (in) length of the method
    // parameters
    struct _node* inparams; // (in) input xml tree
    struct _node* outparams; // (out) output xml tree
    int  isfault; // (out) <=0 if fault
    // proxy info
    HTTP_PROXY* proxy; // (in) proxy info (or NULL)
}
SOAP_CLNT_PARAM;
typedef SOAP_CLNT_PARAM* LPSOAP_CLNT_PARAM;

// SOAP Action codes
#define SOAP_CLNT_ACTION_INIT 0 // Initialize the Client
#define SOAP_CLNT_ACTION_CALL 1 // Process the method call
#define SOAP_CLNT_ACTION_DONE 2 // Cleanup the Client

// SOAP Error codes
#define SOAPERR_NO_ERROR      0 // No error
#define SOAPERR_INVALID_PARAM 1 // invalid param (e.g. NULL)
#define SOAPERR_NULL_POINTER  2 // unexpected null pointer
#define SOAPERR_INVALID_RESPONSE 3 // Invalid SOAP response
#define SOAPERR_INTERNAL_ERR  4 // internal or config error
#define SOAPERR_XML_SYNTAX_ERR 5 // syntax error in XML
#define SOAPERR_INVALID_URN   6 // invalid or no URN specified

```

```
#define SOAPERR_HTTP_ERROR      1000 // HTTP Errors starting here

// Entry point function. Executes a SOAP Call
typedef int (*SOAPClient)(LPSOAP_CLNT_PARAM lpCall);
#define SOAP_CLIENT_PHASE_NAME  "IESSOAPC"

// Macros for fetching and releasing the SOAP Client Phase
#define FETCH_SOAP_CLIENT      (SOAPClient)fetch(SOAP_CLIENT_PHASE_NAME)
#define RELEASE_SOAP_CLIENT(p) release((void(*)())p)

#endif
```

Code is in Linux 9.12.9.12 in /javacode/TestECl.java.

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG247042>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG247042.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
SG247042.zip	Zipped code samples

System requirements for downloading the Web material

The examples are using WebSphere Studio-IE environment, so all requirements that apply to WebSphere Studio-IE apply to our examples.

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

There is a readme.txt file in each directory to describe the contents and usage.

Also, refer to 3.2, “Running Trader in WSAD.IE test environment” on page 45.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 342. Note that some of the documents referenced here may be available in softcopy only.

- ▶ e-business Connectivity for VSE/ESA, SG24-5950
- ▶ e-business Solutions for VSE/ESA, SG24-5662
- ▶ MQ for VSE/ESA, SG24-5647
- ▶ CICS Transaction Server for VSE/ESA: CICS Web Support , SG24-5997
- ▶ Getting Started with TCP/IP for VSE/ESA 1.4, SG24-5626
- ▶ *WebSphere Version 5 Web Services Handbook*, SG24-6891
- ▶ IBM WebSphere Application Server V5.0 System Management and Configuration, SG24-6195

Other publications

These publications are also relevant as further information sources:

IMS

- ▶ *IMS Connect Guide and Reference V2.1*, SG18-7260

WebSphere

- ▶ *WebSphere Studio Application Developer Integration Edition, 5.0.1*, SC09-7869

CICS

- ▶ *CICS Transaction Gateway V5 The WebSphere Connector for CICS*, SG24-6133
- ▶ *CICS Transaction Gateway V5.0: Unix Administration*, SC34-6190

- ▶ *CICS TS V2.2 External Interfaces Guide*, SC34-6006

MQ

- ▶ *MQSeries Primer*, REDP-0021-00

VSE/ESA

- ▶ *VSE/ESA e-business Connectors User's Guide*, SC33-6719
- ▶ *CICS for VSE/ESA Enhancements Guide*, GC34-5763

Other

- ▶ *UNIX System Services Messages and Codes*, SA22-7807

Online resources

These Web sites and URLs are also relevant as further information sources.

VSE/ESA

- ▶ VSE/ESA Internet page
<http://www.ibm.com/servers/eserver/zseries/os/vse/>
- ▶ VSE e-business connectors and utilities
<http://www.ibm.com/servers/eserver/zseries/os/vse/support/vseconn/>
- ▶ TCP/IP for VSE/ESA and more
<http://www.ibm.com/servers/eserver/zseries/os/vse/support/tcpip/tcpiphome.htm>
- ▶ TCP/IP for VSE/ESA 1.5 documentation
<http://www.ibm.com/servers/eserver/zseries/os/vse/support/tcpip/tcp15.htm>

Linux

- ▶ Linux DeveloperWorks site
<http://www-136.ibm.com/developerworks/linux/>

J2EE connector architecture

- ▶ Sun's J2EE connector architecture
<http://java.sun.com/j2ee/connector/>
- ▶ Overview of the Common Client Interface (CCI)
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Connector5.html#79167

Java

- ▶ Downloading IBM Java developer kits
<http://www-106.ibm.com/developerworks/java/jdk/index.html>
- ▶ Java products and APIs (Sun)
<http://java.sun.com/products/>
- ▶ Java 1.4.2 API documentation
<http://java.sun.com/j2se/1.4.2/docs/api/>
- ▶ Enterprise JavaBeans technology overview
<http://java.sun.com/products/ejb/index.html>
- ▶ Enterprise JavaBeans API documentation
<http://java.sun.com/products/ejb/docs.html>
- ▶ Java developerWorks® site
<http://www-136.ibm.com/developerworks/java/>

WebSphere MQ

- ▶ WebSphere MQ Internet page
<http://www-3.ibm.com/software/integration/wmq/>
- ▶ WebSphere developerworks site
<http://www-136.ibm.com/developerworks/websphere/>

DB2

- ▶ DB2 Server for VSE V7R1 Bookshelf
http://publibz.boulder.ibm.com/cgi-bin/bookmgr_0S390/shelves/ARISBS20
- ▶ IBM Manuals for DB2 Information Management Products
<http://www-3.ibm.com/software/data/db2/library/>
- ▶ DB2 DeveloperWorks site:
<http://www-136.ibm.com/developerworks/db2>

IMS

- ▶ Description1
<http://www.ibm.com/software/data/db2imstools/index.html>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

A

- AAT 252
- adapter 2
- Application Assembly Tool 252
- application contracts 5
- application server 3
- authentication 13
- authorization 13

B

- basic password 16

C

- caller impersonation 16
- CCI 5, 17
- CICS
 - connectors 43, 83
 - SOAP connection 119, 122
 - transaction gateway 83
 - connection factory
 - for VSE 112
 - for z/OS 108
 - deployment to WAS 114
 - problem determination 116
 - setup
 - CICS for VSE 98
 - CICS for z/OS 96
 - CTG 86
 - WAS 105
 - testing the connections 103
- CMP 147
- commarea 18, 121
- common client interface 2, 5, 17
 - enterprise application integration 17
- configured identity 15
- connection factory
 - CICS 48, 108, 112
 - IMS 48, 214
 - MQ 197
 - SSL 273
 - VSE 229
- connection management 5

- connection pooling 5
- ConnectionEventListener 8
- ConnectionFactory 7
- ConnectionManager 7
 - managed environment 6
 - non-managed environment 8
- connectors 2, 25
 - CICS transaction gateway 83
 - DB2 131
 - IMS 207
 - SOAP for CICS 119
 - VSE Java connectors 223
 - VSE VSAM redirector 293
 - WebSphere MQ 153
- containers 4
 - container-managed 4
 - EJB container 4
 - servlet container 4
- credentials 14
 - mapping 16
- CTG 66, 83

D

- datasources 50, 280
- DB2
 - connectors 131
 - datasources 50, 145
 - DB2 Connect 132
 - tables 58, 63
 - DB2 Connect 132
 - setup 132
 - simple connection test 142
- db2profile 144
- deployment 19
 - descriptor 20
 - sample 21
 - resource adapter 22
 - Trader application 59
- deployment and packaging 5, 19
 - deployment
 - descriptor 20
 - packaging
 - resource adapter 20

E

EAI 17
EIS 1
enterprise application integration 17
ERP 2

G

global transactions 11
 JTA transactions 11
 transaction manager 11
 xa_transaction 12
 XAResource interfaces 11

I

IMS

 connection factory 214
 connectors 43, 207
 deploy TraderIMS in WAS 217
 IMS Connect 208
 configuration 218
 IMS Connector for Java 208
 installation 209
 problem determination 220
 resource adapter 211
 thread identity 222

J

J2C 1–2
J2EE connector architecture 1, 3
 application contracts 5
 common client interface 2
 deployment and packaging 19
 resource adapter 2
 system contracts 3
JAAS 41
Java platform 2
javax.resource.cci.Connection 7
javax.resource.cci.ConnectionFactory 7
javax.resource.spi.ConnectionEventListener 12
javax.resource.spi.LocalTransaction interface 12
javax.transaction.xa.XAResource interface 11
JCA 1
JDBC
 API 17
 connector 43
 driver 145
 provider 145

JMS setup 51
JNDI 7, 41, 50, 197, 203, 206
JTA transactions 11

K

kerbv5 16
Kerebos 16
Keyman/VSE 263
keyring 272

L

LD_LIBRARY_PATH 144
local transactions 12
 javax.resource.spi.LocalTransaction interface 12

M

managed environment 6
ManagedConnection 8
ManagedConnectionFactory 8
message listeners 197

N

non-managed environment 8

O

one-phase commit optimization 13

P

packaging 19
 resource adapter 20
principal 13
principal mapping 16

R

RA 2
Redbooks Web site 342
 Contact us xv
resource adapter 2–3, 20, 22
 CICS 105, 109
 IMS 208, 211
 VSE 227

S

secure communications 13

- secure socket layer 13, 262
- security attributes 13
- security management 13
 - authentication 16
 - authorization 16
 - component-managed 14
 - container-managed 14
 - resource principal 15
 - secure communication 16
 - security model overview 14
- security principal 14
- Simple Object Access Protocol 119
- SOAP 119–120
 - client 127
 - for CICS 122
 - IBM provided include file 330
 - IBM provided service C program 325
 - on VSE 121
 - on z/OS 121
 - programs 129
 - sample Java code 320
- SSL 13, 262
- standard 2
- startServer.sh 144
- system contracts 3, 5
 - connection management 3
 - security management 3
 - transaction management 3
- system functions 4
 - component-managed 4
 - container-managed 4

T

- Trader applications 33
 - components 34
 - configuring DB2 tables 58
 - configuring the test environment 47
 - connectors overview 42
 - datastores 35
 - DB2 table definitions 63
 - dependencies 41
 - deployment in WAS 59
 - TraderCICS 114
 - TraderDB 149
 - TraderIMS 217
 - TraderMQ 196, 198
 - downloading application modules 45
 - front-end GUI 36

- packaging 40
 - running in WSAD.IE 45, 56, 59
 - running the application 60
 - scenarios 31
 - troubleshooting 53
 - VSAM file definitions 65
 - web front-end architecture 38
- Trader Connectors jar 41
- Trader EJB jars 41
- TraderLib.jar 40
- TraderProcess 39
- TraderProcessEJB 39
- TraderServlet 39
- TraderSuperServlet 39–40
- transaction management 9
 - contract 10
 - global transactions 11
 - local transactions 12
 - local transactions versus one phase commit optimization 13
 - no support 11
 - one-phase commit optimization 13
- transaction manager 11
- troubleshooting 53

V

- VSAM
 - files 65, 243, 249
 - populate with sample Java program 322
 - maps 244
 - maptool 242
 - redirector connector 293
 - VSE Java connectors 223
- VSE 223
 - Java connectors 223
 - access DL/1 data 287
 - access VSAM data 241
 - connection factory 229, 273
 - connector client/server 225
 - deployment in WAS 252
 - installation 226
 - JDBC provider 231
 - problem determination 284
 - resource adapter 227
 - SSL JDBC datasource 280
 - navigator 242
 - VSAM redirector connector 293

W

- WAS 67
- WebSphere Application Server 67
 - installing WAS 72
 - installing WebSphere MQ Client 69
 - planning for WAS setup 68
 - starting WAS 78
- WebSphere MQ 44
 - client 68–70
 - configure VSE for MQ 175
 - connection factory 197
 - connectors 153
 - deployment in WebSphere 196
 - MQ-CICS bridge 155
 - MQ-IMS Bridge 156
 - setup
 - MQ
 - on Linux 157
 - on VSE 168
 - on z/OS 162
 - MQ connector 196
 - troubleshooting 195
- WSAD.IE 45, 47, 56, 59, 66, 208

X

- xa_transaction 12
- XAResource interfaces 11



WebSphere V5 for Linux on zSeries Connectivity Handbook

Archived



Redbooks

WebSphere V5 for Linux on zSeries Connectivity Handbook

**WAS for Linux on
zSeries
implementation**

**J2EE Connector
architectures for
Linux on zSeries**

**Linux to z/OS and
VSE connection
scenarios**

This IBM Redbook discusses Linux-based Java applications connecting to z/OS and VSE backend environments on zSeries. This book describes the implementation and deployment of Websphere Application Server, Java frontend applications, and the J2EE connectors needed on Linux for zSeries to connect to backend applications, such as CICS, IMS, DB2 and MQ on zSeries. This book explains the J2EE Connector architecture and provides the following comprehensive connector scenarios for connections to both z/OS and VSE backend environments:

- CICS Transaction Gateway
- IMS Connect
- WebSphere MQ
- JDBC to DB2
- SOAP to CICS

**INTERNATIONAL
TECHNICAL
SUPPORT
ORGANIZATION**

**BUILDING TECHNICAL
INFORMATION BASED ON
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:
ibm.com/redbooks**