IMS

IMS Java Guide and Reference

Version 9

SC18-7821-03

IMS

IMS Java Guide and Reference

Version 9

SC18-7821-03

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 173.

Fourth Edition (August 2005) (Softcopy Only)

This edition replaces or makes obsolete the previous edition, SC18-7821-02. This edition is available in softcopy format only. The technical changes for this version are summarized under “Summary of Changes” on page xvii.

Contents

Figures ix

Tables xi

About This Book xiii
Prerequisite Knowledge xiii
IBM Product Names Used in This Information xiii
How to Read Syntax Diagrams xiv
How to Send Your Comments xvi

Summary of Changes xvii
Changes to This Edition of This Book xvii
Changes to This Book for IMS Version 9 xvii
Library Changes for IMS Version 9 xix
New and Revised Titles xix
Organizational Changes xix
Terminology Changes xix
Accessibility Enhancements xx
User Assistive Technologies xx
Accessible Information xx
Keyboard Navigation of the User Interface xx

Chapter 1. Getting Started with IMS Java 1
IMS Java System Requirements 1
Installing IMS Java 2
Downloading Apache Open Source XML Libraries 3
Administering IMS Java 4
IMS Java Class Library Summary 5
General Restrictions for Applications 6
Where to Find More Information about IMS Java 6

Chapter 2. JMP and JBP Applications 9
Running the IMS Java IVP in a JMP Region 10
Running the IMS Java IVP in a JBP Region 13
Running the IMS Java Sample Application from a JMP Region 14
Configuring JMP and JBP Regions for DB2 UDB for z/OS Database Access 16
Developing JMP Applications 17
Subclassing the IMSFieldMessage Class to Define Input Messages 18
Subclassing the IMSFieldMessage Class to Define Output Messages 18
Implementing the main Method 19
JMP Programming Models 20
JMP Application Without Rollback 20
JMP Application that Uses Rollback 21
JMP Application that Accesses IMS or DB2 UDB for z/OS Data 21
Additional Message Handling Considerations for JMP Applications 22
Conversational Transactions 22
Handling Multi-Segment Messages 24
Coding and Accessing Messages with Repeating Structures 25
Flexible Reading of Multiple Input Messages 26
Developing JBP Applications 28
Symbolic Checkpoint and Restart 28
JBP Programming Models 29
JBP Application without Rollback 29

JBP Application with Symbolic Checkpoint and Restart 29
JBP Application using Rollback 30
JBP Application that Accesses DB2 UDB for z/OS or IMS Data 30
Enterprise COBOL Interoperability with JMP and JBP Applications 31
Enterprise COBOL as a Back-End Application in a JMP or JBP Region . . . 32
Enterprise COBOL as a Front-End Application in a JMP or JBP Region . . . 32
Performance Consideration for OO COBOL in a JMP or JBP Region 33
Recommendation against Accessing Databases with Both Java and COBOL 33
Accessing DB2 UDB for z/OS Databases from JMP or JBP Applications 34
Program Switching in JMP and JBP Applications 35
Immediate Program Switching for JMP and JBP Applications 35
Deferred Program Switching for Conversational JMP Applications 35

Chapter 3. WebSphere Application Server for z/OS Applications 37
Configuring WebSphere Application Server for z/OS for IMS Java 38
Configuring WebSphere Application Server V5 for z/OS 39
Configuring WebSphere Application Server V5 for z/OS to Access IMS 39
Adding the Required XML Files to the WebSphere Application Server V5 for z/OS Classpath 39
Installing the IMS JDBC Resource Adapter on WebSphere Application Server V5 for z/OS 40
Installing the Custom Service on WebSphere Application Server V5 for z/OS 41
Configuring WebSphere Application Server V6 for z/OS 42
Configuring WebSphere Application Server V6 for z/OS to Access IMS 42
Installing the IMS JDBC Resource Adapter on WebSphere Application Server V6 for z/OS 43
Installing the Custom Service on WebSphere Application Server V6 for z/OS 43
Running the IMS Java IVP on WebSphere Application Server for z/OS 44
Running the IMS Java IVP on WebSphere Application Server V5 for z/OS 45
Installing the Data Source for the IMS Java IVP on WebSphere Application Server V5 for z/OS 45
Installing the IMS Java IVP on WebSphere Application Server V5 for z/OS 46
Adding the XML Files to the IVP Classpath on WebSphere Application Server V5 for z/OS 47
Testing the IMS Java IVP on WebSphere Application Server V5 for z/OS 48
Running the IMS Java IVP on WebSphere Application Server V6 for z/OS 48
Installing the Data Source for the IMS Java IVP on WebSphere Application Server V6 for z/OS 49
Installing the IMS Java IVP on WebSphere Application Server V6 for z/OS 50
Testing the IMS Java IVP on WebSphere Application Server V6 for z/OS 50
Running the IMS Java Sample Applications on WebSphere Application Server for z/OS 51
Running the IMS Java Sample Applications on WebSphere Application Server V5 for z/OS 52
Installing the Data Source for the IMS Java Samples on WebSphere Application Server V5 for z/OS 52
Installing the IMS Java Sample Applications on WebSphere Application Server V5 for z/OS 54
Testing the IMS Java Sample Applications on WebSphere Application Server V5 for z/OS 55
Running the IMS Java Sample Applications on WebSphere Application Server V6 for z/OS 56
Installing the Data Source for the IMS Java Samples on WebSphere Application Server V6 for z/OS 56

Installing the IMS Java Sample Applications on WebSphere Application Server V6 for z/OS . . . 57
Testing the IMS Java Sample Applications on WebSphere Application Server V6 for z/OS . . . 58
Running Your Applications on WebSphere Application Server for z/OS . . . 59
Running Your Applications on WebSphere Application Server V5 for z/OS 60
Setting the WebSphere Application Server V5 for z/OS Classpath . . . 60
Installing the Data Source for Your Application on WebSphere Application Server V5 for z/OS . . . 60
Installing Your Application on WebSphere Application Server V5 for z/OS 61
Adding the XML Files to the Application Classpath on WebSphere Application Server V5 for z/OS . . . 62
Enabling J2EE Tracing with WebSphere Application Server V5 for z/OS 63
Running Your Applications on WebSphere Application Server V6 for z/OS 65
Setting the WebSphere Application Server V6 for z/OS Classpath . . . 65
Installing the Data Source for Your Application on WebSphere Application Server V6 for z/OS . . . 65
Installing Your Application on WebSphere Application Server V6 for z/OS 67
Enabling J2EE Tracing with WebSphere Application Server V6 for z/OS 67
Developing Enterprise Applications that Access IMS DB . . . 69
Bean-Managed EJB Programming Model . . . 69
Transaction Demarcation Using the javax.transaction.UserTransaction Interface . . . 69
Transaction Demarcation Using the java.sql.Connection Interface . . . 70
Container-Managed EJB Programming Model . . . 71
Servlet Programming Model . . . 71
Programming Requirements for WebSphere Application Server for z/OS . . . 72
Deployment Descriptor Requirements for IMS Java . . . 72

Chapter 4. Remote Data Access with WebSphere Application Server Applications . . . 75
Downloading IMS Java Files for Remote Database Services . . . 77
Configuring the Application Servers for IMS Java Remote Database Services 77
Configuring the V5 Application Servers for IMS Java Remote Database Services . . . 78
Mapping Hostnames for the Client and Server . . . 78
Installing the Data Source on WebSphere Application Server V5 for z/OS 78
Installing the EAR file on WebSphere Application Server V5 for z/OS . . . 79
Adding the XML Files to the EAR Classpath . . . 80
Installing the IMS Distributed JDBC Resource Adapter on WebSphere Application Server V5 . . . 81
Configuring the V6 Application Servers for IMS Java Remote Database Services . . . 81
Mapping Hostnames for the Client and Server . . . 82
Installing the Data Source on WebSphere Application Server V6 for z/OS 82
Installing the EAR file on WebSphere Application Server V6 for z/OS . . . 83
Installing the IMS Distributed JDBC Resource Adapter on WebSphere Application Server V6 . . . 83
Running the IMS Java IVP for Remote Database Services . . . 84
Running the IMS Java IVP for Remote Database Services on WebSphere Application Server V5 . . . 84
Setting the WebSphere Application Server V5 for z/OS Classpath . . . 85
Installing the Data Source for the IVP on the Client Side . . . 85
Installing the IVP on the Client Side . . . 86
Testing the IVP on WebSphere Application Server V5. . . . 87

Running the IMS Java IVP for Remote Database Services on WebSphere Application Server V6 88
Setting the WebSphere Application Server V6 for z/OS Classpath 88
Installing the Data Source for the IVP on the Client Side 88
Installing the IVP on the Client Side 89
Testing the IVP on WebSphere Application Server V6. 90
Running the IMS Java Sample Applications for Remote Database Services 91
Running the IMS Java Sample Applications on WebSphere Application Server V5. 92
Setting the WebSphere Application Server V5 for z/OS Classpath 92
Installing the Data Source for the IMS Java Samples on the Client Side 92
Installing the IMS Java Sample Applications on the Client Side 93
Testing the Phonebook Sample on WebSphere Application Server V5. . . 95
Testing the Dealership Sample on WebSphere Application Server V5 . . . 95
Running the IMS Java Sample Applications on WebSphere Application Server V6. 96
Setting the WebSphere Application Server V6 for z/OS Classpath 96
Installing the Data Source for the IMS Java Samples on the Client Side 96
Installing the IMS Java Sample Applications on the Client Side 98
Testing the Phonebook Sample on WebSphere Application Server V6. . . 99
Testing the Dealership Sample on WebSphere Application Server V6 . . . 99
Running Your Application on WebSphere Application Server 100
Running Your Application on WebSphere Application Server V5 100
Setting the WebSphere Application Server V5 for z/OS Classpath. . . . 100
Installing the Data Source on the Client Side 101
Installing the Application on the Client Side 102
Enabling J2EE Tracing with WebSphere Application Server V5 103
Running Your Application on WebSphere Application Server V6 104
Setting the WebSphere Application Server V6 for z/OS Classpath. . . . 104
Installing the Data Source on the Client Side 105
Installing the Application on the Client Side 106
Enabling J2EE Tracing with WebSphere Application Server V6 107
WebSphere Application Server EJBs 108
Transaction Semantics and Server-Side EJB Types 109
Client-Side EJB Security Semantics. 109

Chapter 5. DB2 UDB for z/OS Stored Procedures 111
Configuring DB2 UDB for z/OS for IMS Java 111
Running the IMS Java IVP from DB2 UDB for z/OS 113
Running the IMS Java Sample Application on DB2 UDB for z/OS 115
Running Your Stored Procedure from DB2 UDB for z/OS 116
Developing DB2 UDB for z/OS Stored Procedures that Access IMS DB . . . 118

Chapter 6. CICS Applications 119
Configuring CICS for IMS Java 119
Running the IMS Java IVP on CICS. 120
Running the IMS Java Sample Application on CICS 121
Running Your Applications on CICS 122
Developing CICS Applications that Access IMS DB 123

Chapter 7. JDBC Access to IMS Data 125
Comparison of Hierarchical and Relational Databases 125
Supported SQL Keywords 128
SELECT Statement Usage 129
Selecting Multiple Segments 131
Selecting All Fields in a Segment. 131

Segment-Qualified Fields. 132
Retrieving XML Using the SELECT Statement 132
Summary of SELECT Statement Usage 132
INSERT Statement Usage 133
DELETE Statement Usage 133
UPDATE Statement Usage 134
FROM Clause Usage 134
PCB-Qualified SQL Queries. 134
Summary of FROM Clause Usage 134
WHERE Clause Usage 135
Non-DBD-Defined Fields in the WHERE Clause 136
Summary of WHERE Clause Usage 136
Supported SQL Aggregate Functions 136
SQL Extensions for XML Storage and Retrieval 137
retrieveXML UDF 138
storeXML UDF 139
Supported JDBC Interfaces 140
JDBC Prepared Statements for SQL 142
Supported JDBC Data Types 142
General Mappings from COBOL Copybook Types to IMS Java and Java Data Types 144
JDBC Recommendations for IMS Databases 145
Java Metadata Classes for IMS Databases 146
Sample Application that Uses JDBC 148
Imported Packages for JDBC Access to IMS Databases 149
Connections to IMS Databases 149

Chapter 8. XML Storage in IMS Databases 151
Decomposed Storage Mode for XML 152
Intact Storage Mode for XML 154
Side Segments for Secondary Indexing 155
DBDs for Intact XML Storage 155
XML Schema 157
XML Type Representation 157
JDBC Interface for Storing and Retrieving XML 158

Chapter 9. Problem Determination 159
Exceptions 159
How Exceptions Map to DL/I Status Codes 159
SQLException Objects 160
XML Tracing for IMS Java 160
WebSphere Application Server Security Requirements for XML Tracing 161
Enabling XML Tracing 161
Tracing the IMS Java Library Methods 162
Tracing Your Application 162
Debugging an Unresettable JVM in a JMP or JBP Region 163

Preparing to Run the Dealership Samples 165
Modifying IMS Stage 1 Input Statements 165
Loading the Dealership Sample Databases 165

SQL Keywords 169

IMS Java Hierarchical Database Interface. 171
Application Programming Using the DLIConnection Object 171
Creating a DLIConnection Object. 171

Creating an SSAList Object 172
Accessing IMS Data Using SSAs 172

Notices 173
Trademarks 175

Bibliography 177
IMS Version 9 Library 177
Supplementary Publications 178
Publication Collections 178
Accessibility Titles Cited in This Library 178

Index 179

Figures

1. JMP or JBP Application That is Using IMS Java . . . 10
2. IVP Screen for IMS Java JMP . . . 12
3. Subclass IMSFieldMessage: Input Message Sample Code . . . 18
4. Subclass IMSFieldMessage: Output Message Sample Code . . . 19
5. `main` Method Sample Code . . . 20
6. Defining a SPA Message . . . 23
7. Reading a SPA Message . . . 23
8. Writing a SPA Message . . . 23
9. Sample Output Message with Repeating Structures . . . 25
10. Defining the Primary Input Message . . . 26
11. Defining Separate Input Messages for Each Request . . . 27
12. Message-Reading Logic . . . 28
13. WebSphere Application Server for z/OS EJB Using IMS Java . . . 37
14. IMS Java and WebSphere Application Server Components . . . 75
15. DB2 UDB for z/OS Stored Procedure Using IMS Java . . . 111
16. Sample JAVAENV Data Set . . . 112
17. CICS Application Using IMS Java . . . 119
18. Sample Dealership Database . . . 126
19. Relational Representation of the Dealership Database . . . 127
20. Segment Occurrences in the Dealership Database . . . 128
21. Relational Representation of Segment Occurrences in the Dealership Database . . . 128
22. Example of SELECT Statement Query Results . . . 130
23. Sample Relational Database Query . . . 131
24. Sample Hierarchical Database Query . . . 131
25. Simple Way to Select All Fields in a Segment . . . 132
26. Long Way to Select All Fields in a Segment . . . 132
27. Sample INSERT Statement . . . 133
28. Sample DELETE Statement . . . 133
29. Sample UPDATE Statement . . . 134
30. PCB-Qualified SQL Query Example . . . 134
31. Creating XML Using the `retrieveXML` UDF and the `getClob` Method . . . 138
32. Establishing a Connection to the Dealership Database . . . 140
33. Sample PSB for the Sample Dealership Application . . . 146
34. DBD for the Sample Dealership Database . . . 146
35. Sample DLIModel IMS Java Report for the Dealership Sample Database . . . 147
36. Example JDBC Application . . . 149
37. Overview of XML Storage in IMS . . . 152
38. How XML is Decomposed XML and Stored in IMS Segments . . . 153
39. Intact Storage of XML with a Secondary Index . . . 155
40. DBD for Intact XML Storage and No Secondary Indexes . . . 156
41. DBD for Intact XML Storage and Two Secondary Indexes . . . 156
42. Secondary Index DBD for Intact XML Storage . . . 156
43. IMSException Class Example . . . 160
44. Setting a Trace within a Static Method . . . 162
45. Creating a DLIConnection Object . . . 172
46. Creating an SSAList Object . . . 172

Tables

1. Licensed Program Full Names and Short Names xiii
2. Relationship between the Transaction Context and the Transaction Semantics. 109
3. Supported SQL Aggregate Functions and Their Supported Data Types 137
4. Supported JDBC Data Types 142
5. ResultSet.getxxx Methods to Retrieve JDBC Types. 143
6. Mapping from COBOL Formats to DLITypeInfo Constants and Java Data Types 144
7. DLITypeInfo Constants and Java Data Types Based on the PICTURE Clause 144
8. Copybook Formats Mapped to DLITypeInfo Constants 145
9. Primary Intact Field Format 154
10. Overflow XML Data Field Format 154
11. IMS Java-Supported XML Schema Data Types 158

About This Book

This information is available as part of the DB2 Information Management Software Information Center for z/OS Solutions. To view the information within the DB2 Information Management Software Information Center for z/OS Solutions, go to http://publib.boulder.ibm.com/infocenter/dzichelp. This information is also available in PDF and BookManager® formats. To get the most current versions of the PDF and BookManager formats, go to the IMS™ Library page at http://www.ibm.com/software/data/ims/library.html.

This book provides application development and deployment information for IMS Java™, a function of IMS that allows you to write Java application programs that access IMS databases from multiple systems. This book also explains XML support for IMS databases.

Information about IMS Java is also available from the IMS Web site. Go to www.ibm.com/ims and link to the IMS Java page.

Prerequisite Knowledge

To configure your system for IMS Java, you must understand system administration for your system (IMS, WebSphere® Application Server, CICS®, or DB2® UDB for z/OS®). For IMS system administration, you should know the concepts in *IMS Version 9: Administration Guide: System*.

To create IMS Java metadata classes, which is a required step in writing IMS Java applications, you must understand IMS databases. IMS database concepts are described in *IMS Version 9: Administration Guide: Database Manager*.

To write Java applications, you must thoroughly understand the Java language and JDBC. This book assumes that you know Java and JDBC. It does not explain any Java or JDBC concepts.

To write applications that store or retrieve XML, you must understand XML and its related technologies, such as XML schemas.

IBM Product Names Used in This Information

In this information, the licensed programs shown in Table 1 are referred to by their short names.

Table 1. Licensed Program Full Names and Short Names

Licensed program full name	Licensed program short name
IBM® Application Recovery Tool for IMS and DB2	Application Recovery Tool
IBM CICS Transaction Server for OS/390®	CICS
IBM CICS Transaction Server for z/OS	CICS
IBM DB2 Universal Database™	DB2 Universal Database
IBM DB2 Universal Database for z/OS	DB2 UDB for z/OS
IBM Enterprise COBOL for z/OS and OS/390	Enterprise COBOL
IBM Enterprise PL/I for z/OS and OS/390	Enterprise PL/I

Table 1. Licensed Program Full Names and Short Names (continued)

Licensed program full name	Licensed program short name
IBM High Level Assembler for MVS™ & VM & VSE	High Level Assembler
IBM IMS Advanced ACB Generator	IMS Advanced ACB Generator
IBM IMS Batch Backout Manager	IMS Batch Backout Manager
IBM IMS Batch Terminal Simulator	IMS Batch Terminal Simulator
IBM IMS Buffer Pool Analyzer	IMS Buffer Pool Analyzer
IBM IMS Command Control Facility for z/OS	IMS Command Control Facility
IBM IMS Connect for z/OS	IMS Connect
IBM IMS Connector for Java	IMS Connector for Java
IBM IMS Database Control Suite	IMS Database Control Suite
IBM IMS Database Recovery Facility for z/OS	IMS Database Recovery Facility
IBM IMS Database Repair Facility	IMS Database Repair Facility
IBM IMS DataPropagator™ for z/OS	IMS DataPropagator
IBM IMS DEDB Fast Recovery	IMS DEDB Fast Recovery
IBM IMS Extended Terminal Option Support	IMS ETO Support
IBM IMS Fast Path Basic Tools	IMS Fast Path Basic Tools
IBM IMS Fast Path Online Tools	IMS Fast Path Online Tools
IBM IMS Hardware Data Compression-Extended	IMS Hardware Data Compression-Extended
IBM IMS High Availability Large Database (HALDB) Conversion Aid for z/OS	IBM IMS HALDB Conversion Aid
IBM IMS High Performance Change Accumulation Utility for z/OS	IMS High Performance Change Accumulation Utility
IBM IMS High Performance Load for z/OS	IMS HP Load
IBM IMS High Performance Pointer Checker for OS/390	IMS HP Pointer Checker
IBM IMS High Performance Prefix Resolution for z/OS	IMS HP Prefix Resolution
IBM Tivoli® NetView® for z/OS	Tivoli NetView for z/OS
IBM WebSphere Application Server for z/OS and OS/390	WebSphere Application Server for z/OS
IBM WebSphere MQ for z/OS	WebSphere MQ
IBM WebSphere Studio Application Developer Integration Edition	WebSphere Studio
IBM z/OS	z/OS

How to Read Syntax Diagrams

The following rules apply to the syntax diagrams that are used in this information:

- Read the syntax diagrams from left to right, from top to bottom, following the path of the line. The following conventions are used:
  - The >>--- symbol indicates the beginning of a syntax diagram.

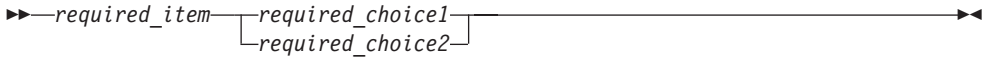
  - The ---> symbol indicates that the syntax diagram is continued on the next line.
  - The >--- symbol indicates that a syntax diagram is continued from the previous line.
  - The --->< symbol indicates the end of a syntax diagram.
- Required items appear on the horizontal line (the main path).

- Optional items appear below the main path.

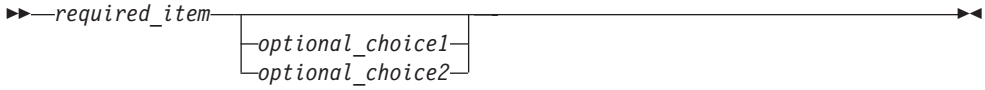
  If an optional item appears above the main path, that item has no effect on the execution of the syntax element and is used only for readability.

- If you can choose from two or more items, they appear vertically, in a stack.

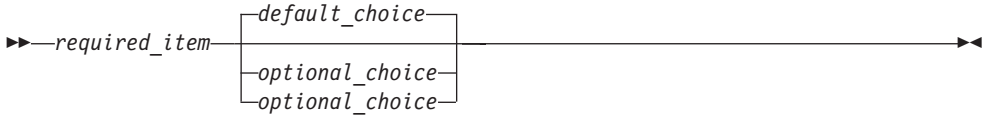
  If you *must* choose one of the items, one item of the stack appears on the main path.



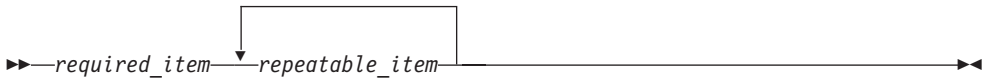
  If choosing one of the items is optional, the entire stack appears below the main path.



  If one of the items is the default, it appears above the main path, and the remaining choices are shown below.



- An arrow returning to the left, above the main line, indicates an item that can be repeated.

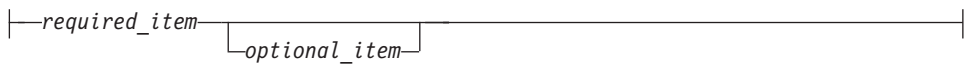


  If the repeat arrow contains a comma, you must separate repeated items with a comma.

A repeat arrow above a stack indicates that you can repeat the items in the stack.

- Sometimes a diagram must be split into fragments. The syntax fragment is shown separately from the main syntax diagram, but the contents of the fragment should be read as if they are on the main path of the diagram.

fragment-name:



- In IMS, a b symbol indicates one blank position.
- Keywords, and their minimum abbreviations if applicable, appear in uppercase. They must be spelled exactly as shown. Variables appear in all lowercase italic letters (for example, *column-name*). They represent user-supplied names or values.
- Separate keywords and parameters by at least one space if no intervening punctuation is shown in the diagram.
- Enter punctuation marks, parentheses, arithmetic operators, and other symbols, exactly as shown in the diagram.
- Footnotes are shown by a number in parentheses, for example (1).

How to Send Your Comments

Your feedback is important in helping us provide the most accurate and highest quality information. If you have any comments about this or any other IMS information, you can take one of the following actions:

- Go to the IMS Library page at www.ibm.com/software/data/ims/library.html and click the Library Feedback link, where you can enter and submit comments.
- Send your comments by e-mail to imspubs@us.ibm.com. Be sure to include the title, the part number of the title, the version of IMS, and, if applicable, the specific location of the text on which you are commenting (for example, a page number in the PDF or a heading in the Information Center).

Summary of Changes

Changes to This Edition of This Book

This edition contains new information about the following enhancements to IMS Version 9:

- Support for WebSphere Application Server V6 for z/OS and WebSphere Application Server V6. See Chapter 3, “WebSphere Application Server for z/OS Applications,” on page 37 and Chapter 4, “Remote Data Access with WebSphere Application Server Applications,” on page 75.

This edition also contains corrected technical information and editorial changes to Chapter 6, “CICS Applications,” on page 119.

Changes to This Book for IMS Version 9

This version contains new information about the following enhancements to IMS Version 9:

- Support for XML storage and retrieval from IMS databases: Chapter 8, “XML Storage in IMS Databases,” on page 151
- Access to IMS databases from WebSphere Application Server on non-z/OS platforms: Chapter 4, “Remote Data Access with WebSphere Application Server Applications,” on page 75
- Symbolic checkpoint and restart for JBP applications: “Symbolic Checkpoint and Restart” on page 28
- Improved IVPs for IMS Java:
  - “Running the IMS Java IVP in a JMP Region” on page 10
  - “Running the IMS Java IVP in a JBP Region” on page 13
  - “Running the IMS Java IVP on WebSphere Application Server for z/OS” on page 44
  - “Running the IMS Java IVP from DB2 UDB for z/OS” on page 113
  - “Running the IMS Java IVP on CICS” on page 120
- PQ93785: GSAM database support for JBP regions: “Symbolic Checkpoint and Restart” on page 28
- PQ97361: SQL search support for non-DBD-defined fields: “Non-DBD-Defined Fields in the WHERE Clause” on page 136
- PK01881: “Deferred Program Switching for Conversational JMP Applications” on page 35

This version also contains the following new information:

- “Installing IMS Java” on page 2
- “Downloading Apache Open Source XML Libraries” on page 3
- “Where to Find More Information about IMS Java” on page 6
- “Configuring DB2 UDB for z/OS for IMS Java” on page 111
- “Configuring CICS for IMS Java” on page 119
- “Running the IMS Java Sample Application from a JMP Region” on page 14
- “Running the IMS Java Sample Application on DB2 UDB for z/OS” on page 115
- “Running the IMS Java Sample Application on CICS” on page 121
- “Developing Enterprise Applications that Access IMS DB” on page 69

- “XML Tracing for IMS Java” on page 160
- “Debugging an Unresettable JVM in a JMP or JBP Region” on page 163

This edition also contains the following new and corrected information:

- “Adding the Required XML Files to the WebSphere Application Server V5 for z/OS Classpath” on page 39
- “Adding the XML Files to the IVP Classpath on WebSphere Application Server V5 for z/OS” on page 47
- “Adding the XML Files to the Application Classpath on WebSphere Application Server V5 for z/OS” on page 62
- “Adding the XML Files to the EAR Classpath” on page 80
- “Specifying at Runtime the Application Server and the Package to Trace” on page 68
- “Deployment Descriptor Requirements for IMS Java” on page 72
- “Program Switching in JMP and JBP Applications” on page 35
- “Enabling XML Tracing” on page 161
- “Running the IMS Java IVP from DB2 UDB for z/OS” on page 113

Information about the DLIModel utility has moved:

- The chapter ″DLIModel Utility″ has moved to the *IMS Version 9: Utilities Reference: System*.
- Information about the DLIModel utility messages, codes, and abends has moved to *IMS Version 9: Messages and Codes, Volume 1*.

The following information has been deleted:

- WebSphere Application Server V4.0.1 for z/OS configuration information. IMS Version 9 requires that you use WebSphere Application Server V5 for z/OS. See Chapter 3, “WebSphere Application Server for z/OS Applications,” on page 37.
- Manually creating IMS Java metadata classes. You should always use the DLIModel utility to generate metadata classes. See the *IMS Version 9: Utilities Reference: System*.
- `IMSTrace` facility. This facility has been deprecated. Use the `XMLTrace` facility instead. See “XML Tracing for IMS Java” on page 160.

This version also contains major organizational changes:

- The parts have been removed.
- The overview information for each environment has been moved to the chapter for each environment.
- The configuration and installation verification tasks for all environments have been split into their own sections.
- Application development information for JMP and JBP applications have been split into their own sections.
- Chapter 7, “JDBC Access to IMS Data,” on page 125 has been reorganized to improve retrievability of reference information.
- Sample JCL jobs have been added throughout the book to provide an alternative to directly using UNIX® System Services.

To get the latest information about IMS Java, including enhancements to the product and corrections to the information, go to http://www.ibm.com/ims and link to the IMS Java page.

Library Changes for IMS Version 9

Changes to the IMS Library for IMS Version 9 include the addition of one title, a change of one title, organizational changes, and a major terminology change. Changes are indicated by a vertical bar (|) to the left of the changed text.

The IMS Version 9 information is now available in the DB2 Information Management Software Information Center for z/OS Solutions, which is available at http://publib.boulder.ibm.com/infocenter/dzichelp. The DB2 Information Management Software Information Center for z/OS Solutions provides a graphical user interface for centralized access to the product information for IMS, IMS Tools, DB2 Universal Database (UDB) for z/OS, DB2 Tools, and DB2 Query Management Facility (QMF™).

New and Revised Titles

The following list details the major changes to the IMS Version 9 library:

- *IMS Version 9: IMS Connect Guide and Reference*

  The library includes new information: *IMS Version 9: IMS Connect Guide and Reference*. This information is available in softcopy format only, as part of the DB2 Information Management Software Information Center for z/OS Solutions, and in PDF and BookManager formats.

  IMS Version 9 provides an integrated IMS Connect function, which offers a functional replacement for the IMS Connect tool (program number 5655-K52). In this information, the term *IMS Connect* refers to the integrated IMS Connect function that is part of IMS Version 9, unless otherwise indicated.
- The information formerly titled *IMS Version 8: IMS Java User's Guide* is now titled *IMS Version 9: IMS Java Guide and Reference*. This information is available in softcopy format only, as part of the DB2 Information Management Software Information Center for z/OS Solutions, and in PDF and BookManager formats.
- To complement the IMS Version 9 library, a new book, *An Introduction to IMS* by Dean H. Meltz, Rick Long, Mark Harrington, Robert Hain, and Geoff Nicholls (ISBN # 0-13-185671-5), is available starting February 2005 from IBM Press. Go to the IMS Web site at www.ibm.com/ims for details.

Organizational Changes

Organization changes to the IMS Version 9 library include changes to:

- *IMS Version 9: IMS Java Guide and Reference*
- *IMS Version 9: Messages and Codes, Volume 1*
- *IMS Version 9: Utilities Reference: System*

The chapter titled ″DLIModel Utility″ has moved from *IMS Version 9: IMS Java Guide and Reference* to *IMS Version 9: Utilities Reference: System*.

The DLIModel utility messages that were in *IMS Version 9: IMS Java Guide and Reference* have moved to *IMS Version 9: Messages and Codes, Volume 1*.

Terminology Changes

IMS Version 9 introduces new terminology for IMS commands:

type-1 command
A command, generally preceded by a leading slash character, that can be

entered from any valid IMS command source. In IMS Version 8, these commands were called *classic* commands.

type-2 command
A command that is entered only through the OM API. Type-2 commands are more flexible than type-2 commands and can have a broader scope. In IMS Version 8, these commands were called *IMSplex* commands or *enhanced* commands.

Accessibility Enhancements

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products. The major accessibility features in z/OS products, including IMS, enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

User Assistive Technologies

Assistive technology products, such as screen readers, function with the IMS user interfaces. Consult the documentation of the assistive technology products for specific information when you use assistive technology to access these interfaces.

Accessible Information

Online information for IMS Version 9 is available in BookManager format, which is an accessible format. All BookManager functions can be accessed by using a keyboard or keyboard shortcut keys. BookManager also allows you to use screen readers and other assistive technologies. The BookManager READ/MVS product is included with the z/OS base product, and the BookManager Softcopy Reader (for workstations) is available on the IMS Licensed Product Kit (CD), which you can download from the Web at www.ibm.com.

Keyboard Navigation of the User Interface

Users can access IMS user interfaces using TSO/E or ISPF. Refer to the *z/OS V1R1.0 TSO/E Primer*, the *z/OS V1R5.0 TSO/E User's Guide*, and the *z/OS V1R5.0 ISPF User's Guide, Volume 1*. These guides describe how to navigate each interface, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

Chapter 1. Getting Started with IMS Java

IMS Java is a function of IMS that allows you to write Java application programs that access IMS databases from many different locations:

- IMS JMP (Java message processing) and JBP (Java batch processing) dependent regions
- IBM WebSphere Application Server for z/OS
- WebSphere Application Server that is running on a non-z/OS platform
- IBM CICS Transaction Server for z/OS
- IBM DB2 Universal Database for z/OS stored procedures

IMS Java implements the JDBC API, which is the standard Java interface for database access. JDBC uses SQL (structured query language) calls. The IMS Java implementation of JDBC supports a selected subset of the full facilities of the JDBC 2.1 API.

IMS Java also extends the JDBC interface for storage and retrieval of XML documents in IMS. For more information, see Chapter 8, “XML Storage in IMS Databases,” on page 151.

In addition to JDBC, IMS Java has another interface to the IMS databases called the *IMS Java hierarchical database interface*. This interface is similar to the standard IMS DL/I database call interface, and provides lower-level access to IMS database functions than the JDBC interface. However, JDBC is the recommended access interface to IMS databases and this book focuses on JDBC. For information about the IMS Java hierarchical database interface, see “IMS Java Hierarchical Database Interface” on page 171.

The following topics provide additional information:

- “IMS Java System Requirements”
- “Installing IMS Java” on page 2
- “Downloading Apache Open Source XML Libraries” on page 3
- “Administering IMS Java” on page 4
- “IMS Java Class Library Summary” on page 5
- “General Restrictions for Applications” on page 6
- “Where to Find More Information about IMS Java” on page 6

IMS Java System Requirements

To use IMS Java to write application programs that access IMS databases, the following software and z/OS components are required:

- IMS Version 9 with the IMS Java FMID
- IBM SDK for z/OS Java 2 Technology Edition, Version 1.3.1 or later
- z/OS Version 1 Release 4 or later
- UNIX System Services available at runtime
- Hierarchic File System (HFS) on z/OS. For information about preparing an HFS, see *z/OS: UNIX System Services File System Interface Reference*.
- Xalan-Java version 2.6.0 or later or equivalent code function. See “Downloading Apache Open Source XML Libraries” on page 3.

Installing IMS Java

IMS Java is delivered in a separate FMID. Before you can install the IMS Java FMID with SMP/E, you must prepare HFS, which is described in this topic.

Prerequisite: Install IMS Version 9 and run the standard IMS IVPs. For details about how to run the IMS IVPs, see *IMS Version 9: Installation Volume 1: Installation Verification*.

To install IMS Java:

1. Allocate a data set for HFS:

```
//HFSALLOC JOB parameters
//*************************************************************/
//* To run this job:                                          */
//* 1) Add JOB statement parameters to meet your requirements. */
//* 2) For DSNAME, change hfsdsn to the name of the new file   */
//*    system.                                                */
//* 3) For VOLUME, change volid to the volser ID of the DASD   */
//*    that will contain the IMS Java HFS data set.           */
//*************************************************************/
//ALLOCATE EXEC PGM=IDCAMS,DYNAMNBR=200
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
  ALLOCATE -
      DSNAME('hfsdsn') -
      RECFM(U)           -
      LRECL(0)           -
      BLKSIZE(32760) -
      DSORG(PO)          -
      VOLUME(volid)  -
      DSNTYPE(HFS)   -
      NEW CATALOG        -
      SPACE(15,5) CYL -
      DIR(200)     -
      UNIT(SYSALLDA)
/*
```

2. Define the mount point directory to mount the HFS:

```
//HFSMOUNT JOB parameters
//*************************************************************/
//* To run this job:                                          */
//* 1) Add JOB statement parameters to meet your requirements. */
//* 2) For FILESYSTEM, change hfsdsn to the name of the file   */
//*    system that you specified in the HFSALLOC job.         */
//* 3) For MOUNTPOINT, change /PathPrefix to the high-level    */
//*    directory name. The directory name must be preceded with*/
//*    a forward slash (/), for example, /apps or /ims/apps.   */
//*    This string must match the PathPrefix                  */
//*    string in the DFSJSMKD job.                            */
//*************************************************************/
//MOUNT    EXEC PGM=IKJEFT01
//SYSTSPRT DD   SYSOUT=*
//SYSTSIN  DD   *
  MOUNT FILESYSTEM('hfsdsn')    /* MOUNT  HFS    */ +
  MOUNTPOINT('/PathPrefix') TYPE(HFS) MODE(RDWR)
/*
```

3. Run the sample installation job DFSJSMKD. DFSJSMKD runs the DFSJMKDR REXX script, which creates the HFS paths for IMS Java.
4. Using SMP/E, install the IMS Java FMID.

Next: The next step varies. If you are using SDK 1.4.1 or earlier, the next step is "Downloading Apache Open Source XML Libraries." If you are using SDK 1.4.2 or later, the next step depends on the environment that your application will run in:

- JMP region: "Running the IMS Java IVP in a JMP Region" on page 10
- JBP region: "Running the IMS Java IVP in a JBP Region" on page 13
- WebSphere Application Server for z/OS: "Configuring WebSphere Application Server for z/OS for IMS Java" on page 38
- WebSphere Application Server on a non-z/OS platform: "Configuring WebSphere Application Server for z/OS for IMS Java" on page 38
- DB2 UDB for z/OS stored procedure: "Configuring DB2 UDB for z/OS for IMS Java" on page 111
- CICS: "Configuring CICS for IMS Java" on page 119

Downloading Apache Open Source XML Libraries

IMS Java and the DLIModel utility require Xalan-Java version 2.6.0 or later, or equivalent code function.

XSLT 4.3.1, which is equivalent to Xalan-Java version 2.6.1, is included in SDK 1.4.2. If you are using SDK 1.4.2 or later, do not download and install the Apache XML files.

If you are using SDK 1.4.1 or lower, you must install Xalan-Java version 2.6.0 or later from the Apache Software Foundation (www.apache.org) XML Project (xml.apache.org), or equivalent code function. The Apache XML Project is a collaborative software development project that licenses open source software at no charge.

The following open source files (or equivalent code function) are required by IMS Java and the DLIModel utility:

xercesImpl.jar
XML parser that is required for IMS Java and the DLIModel utility

xalan.jar
XSLT processor that is required for IMS Java to create XML and transform XML documents

xml-apis.jar
XML APIs that are required for IMS Java and the DLIModel utility

Prerequisite: "Installing IMS Java" on page 2

To download the required open source files for IMS Java and the DLIModel utility from xml.apache.org:

1. Go to http://xml.apache.org.
2. Follow the links to the Xalan-Java 2 page.
3. Follow the links to download the zipped binary file for Xalan-Java version 2.6.0 or later.
4. Decompress the zipped Xalan-Java file.
5. Move the following files to the HFS directory *pathprefix*/usr/lpp/ims/imsjava91/lib:
   - xercesImpl.jar
   - xalan.jar

- xml-apis.jar

Next: The next step varies depending on the environment that your application will run in:

- JMP region: “Running the IMS Java IVP in a JMP Region” on page 10
- JBP region: “Running the IMS Java IVP in a JBP Region” on page 13
- WebSphere Application Server for z/OS: “Configuring WebSphere Application Server for z/OS for IMS Java” on page 38
- WebSphere Application Server on a non-z/OS platform: “Configuring WebSphere Application Server for z/OS for IMS Java” on page 38
- DB2 UDB for z/OS stored procedure: “Configuring DB2 UDB for z/OS for IMS Java” on page 111
- CICS: “Configuring CICS for IMS Java” on page 119

Administering IMS Java

This topic provides the high-level tasks to administer IMS Java: from installing the IMS Java function to deploying your Java application. This topic does not contain application programming information.

To administer IMS Java:

1. Install and configure the required z/OS software for IMS Java. See “IMS Java System Requirements” on page 1 for a list of required software and z/OS components that must be installed before you can use IMS Java.
2. Install IMS Java. See “Installing IMS Java” on page 2.
3. Download and install the required open source files. See “Downloading Apache Open Source XML Libraries” on page 3.
4. If you are using the remote database services of IMS Java, install additional files from the IMS Java Web site. “Downloading IMS Java Files for Remote Database Services” on page 77.
5. Continue configuration for your environment, if necessary:
   - WebSphere Application Server for z/OS: “Configuring WebSphere Application Server for z/OS for IMS Java” on page 38
   - WebSphere Application Server on a non-z/OS platform: “Configuring WebSphere Application Server for z/OS for IMS Java” on page 38 and “Configuring the Application Servers for IMS Java Remote Database Services” on page 77
   - DB2 UDB for z/OS stored procedure: “Configuring DB2 UDB for z/OS for IMS Java” on page 111
   - CICS: “Configuring CICS for IMS Java” on page 119
6. Run the IVP for your environment:
   - JMP region: “Running the IMS Java IVP in a JMP Region” on page 10
   - JBP region: “Running the IMS Java IVP in a JBP Region” on page 13
   - WebSphere Application Server for z/OS: “Running the IMS Java IVP on WebSphere Application Server for z/OS” on page 44
   - WebSphere Application Server on a non-z/OS platform: “Running the IMS Java IVP on WebSphere Application Server for z/OS” on page 44 and “Running the IMS Java IVP for Remote Database Services” on page 84
   - DB2 UDB for z/OS stored procedure: “Running the IMS Java IVP from DB2 UDB for z/OS” on page 113

- CICS: “Running the IMS Java IVP on CICS” on page 120

7. Write the PSB, and generate the DBDs, PSB, and ACB for the application.
8. Using the DBDs and PSB as input, write control statements for the DLIModel utility. See the *IMS Version 9: Utilities Reference: System*.
9. Run the DLIModel utility, which uses the DBDs, PSB, and other input to generate the Java metadata class that the application uses to access the databases. The DLIModel utility is a Java application, so you can run it from the UNIX System Services prompt, or you can run it using the z/OS-provided BPXBATCH utility. See the *IMS Version 9: Utilities Reference: System*.
10. Compile the Java source file of the Java metadata class that is generated to create a Java class file.
11. Provide the Java metadata classes, the DLIModel IMS Java Report, which provides the information about the IMS database, and optionally the generated XML schema to the Java application developer.
12. Update the IMS system definition with a new APPLCTN macro statement for the Java application.
13. Deploy your application:
    - JMP region: Using the IMS Java IVP and sample application as models, start a JMP region with your specific requirements. See *IMS Version 9: Installation Volume 2: System Definition and Tailoring* for all of the available options.
    - JBP region: Using the IMS Java IVP as a model, start a JBP region with your specific requirements. See *IMS Version 9: Installation Volume 2: System Definition and Tailoring* for all of the available options.
    - WebSphere Application Server for z/OS: “Running Your Applications on WebSphere Application Server for z/OS” on page 59
    - WebSphere Application Server on a non-z/OS platform: “Running Your Application on WebSphere Application Server” on page 100
    - DB2 UDB for z/OS stored procedure: “Running Your Stored Procedure from DB2 UDB for z/OS” on page 116
    - CICS: “Running Your Applications on CICS” on page 122

IMS Java Class Library Summary

Your Java application uses the IMS Java class library, which includes the following packages:

com.ibm.ims.base
: Provides classes for basic IMS Java functions and for problem determination.

com.ibm.connector2.ims.db
: Provides classes for connecting to IMS databases from WebSphere Application Server for z/OS.

com.ibm.ims.application
: Provides classes for processing IMS messages, and performs commits and rollbacks for JMP and JBP applications.

com.ibm.ims.db
: Provides classes for the JDBC driver and for the IMS Java hierarchical database interface.

com.ibm.ims.rds
Provides classes for client-side WebSphere Application Server support of remote database services.

com.ibm.ims.rds.host
Provides classes for server-side WebSphere Application Server support of remote database services.

com.ibm.ims.rds.util
Provides classes for storing data that is passed between the client and server components for remote data access support.

com.ibm.ims.xms
Provides classes for storing and retrieving XML in Java applications.

Related Reading: For more information about the IMS Java class library, see the IMS Java API Specification (Javadoc). Go to the IMS Web site at www.ibm.com/ims and link to the IMS Java page.

General Restrictions for Applications

The following restrictions apply to applications that use IMS Java:

- The z/OS JVM restricts the classpath length to 255 characters. Do not create classpaths longer than 255 characters. This restriction does not apply to classpaths in WebSphere Application Server for z/OS.
- IMS Java applications cannot run in an IMS batch environment.
- For IMS Version 8 and later, IMS Java does not support High Performance Java (HPJ).
- IMS does not support local transactions, but IMS Java emulates local transaction semantics, depending on the type of EJB deployed, with remote database services support. Therefore, the `commit`, `rollback`, and `setAutoCommit` methods on an IMS Java JDBC `Connection` object are not supported and throw an `SQLException` object.

Where to Find More Information about IMS Java

The information in this book is only one of the resources available for IMS Java information.

The IMS Java Web site contains current information about IMS Java and links to the resources described in this section. The Web site also has links to presentation materials from recent conferences, downloads, and announcements about IMS Java enhancements. Go to the IMS Web site at www.ibm.com/ims and link to the IMS Java page.

The IMS Java API specification is available on the IMS Java Web site. The specification contains information about the packages described in “IMS Java Class Library Summary” on page 5.

The IMS Support Web site contains a broad range of information about IMS, including IMS Java. Go to http://www.ibm.com/software/data/ims/support.html.

The following Redbooks™ contain information about IMS Java and related technologies:

- *IMS Version 7 Java Update* (SG24-6536): Contains IMS Version 7 level information about running applications from JMP regions, JBP regions, DB2 stored procedures, and CICS.
- *IMS e-business Connectors: A Guide to IMS Connectivity* (SG24-6514): Contains a chapter on setting up open database access (ODBA).
- *ABCs of System Programming Volume 9* (SG24-6989): Describes UNIX System Services (z/OS UNIX) and how to install, tailor, configure, and use the z/OS Version 1 Release 4 version of z/OS UNIX.

More information about IMS Java

Chapter 2. JMP and JBP Applications

Two IMS dependent regions provide a Java Virtual Machine (JVM) environment for Java applications:

Java message processing (JMP) regions

JMP regions are similar to MPP regions, but JMP regions allow the scheduling only of Java message-processing applications. A JMP application is started when there is a message in the queue for the JMP application and IMS schedules the message to be processed. JMP applications are executed through transaction codes submitted by users at terminals and from other applications. Each transaction code represents a transaction that the JMP application processes. A single application can also be started from multiple transaction codes.

JMP applications are very flexible in how they process transactions and where they send the output. JMP applications send any output messages back to the message queues and process the next message with the same transaction code. The program continues to run until there are no more messages with the same transaction code. JMP applications share the following characteristics:

- They are small.
- They can produce output that is needed immediately.
- They can access IMS or DB2 data in a DB/DC environment and DB2 data in a DCCTL environment.

Java batch processing (JBP) regions

JBP regions run flexible programs that perform batch-type processing online and can access the IMS message queues for output (similar to non-message-driven BMP applications). JBP applications are started by submitting a job with JCL or from TSO. JBP applications are like BMP applications, except that they cannot read input messages from the IMS message queue. For example, there is no IN= parameter in the startup procedure. Similarly to BMP applications, JBP applications can use symbolic checkpoint and restart calls to restart the application after an abend. JBP applications can access IMS or DB2 UDB for z/OS data in a DB/DC or DBCTL environment and DB2 UDB for z/OS data in a DCCTL environment

Important: JMP and JBP regions are not necessary if your application runs in WebSphere Application Server, DB2 UDB for z/OS, or CICS. JMP or JBP regions are needed only if your application is going to run in an IMS dependent region.

Figure 1 on page 10 shows a Java application that is running in a JMP or JBP region. JDBC or IMS Java hierarchical interface calls are passed to the IMS Java layer, which converts the calls to DL/I calls.

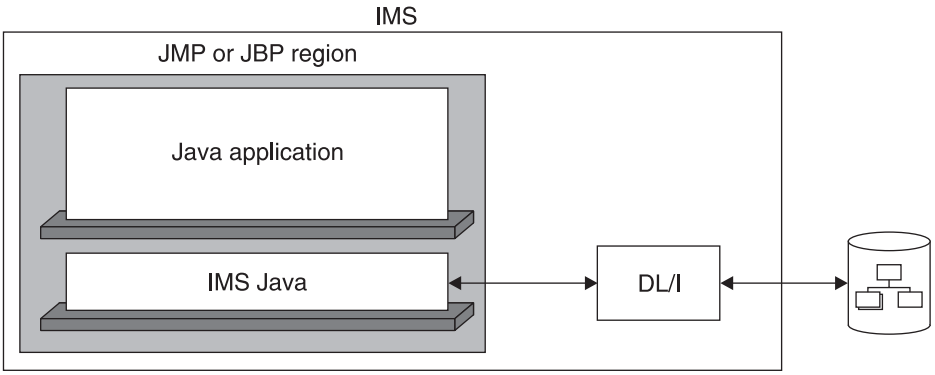


Figure 1. JMP or JBP Application That is Using IMS Java

JMP and JBP regions can run applications written in Java, object-oriented COBOL, or a combination of the two. See “Enterprise COBOL Interoperability with JMP and JBP Applications” on page 31.

JMP and JBP applications can access DB2 UDB for z/OS databases in addition to IMS databases. See “Configuring JMP and JBP Regions for DB2 UDB for z/OS Database Access” on page 16.

This chapter uses the sample applications that are shipped with IMS Java to show how to write and deploy IMS Java applications that run in JMP and JBP regions.

The following topics provide additional information:

- “Running the IMS Java IVP in a JMP Region”
- “Running the IMS Java IVP in a JBP Region” on page 13
- “Running the IMS Java Sample Application from a JMP Region” on page 14
- “Configuring JMP and JBP Regions for DB2 UDB for z/OS Database Access” on page 16
- “Developing JMP Applications” on page 17
- “Developing JBP Applications” on page 28
- “Enterprise COBOL Interoperability with JMP and JBP Applications” on page 31
- “Accessing DB2 UDB for z/OS Databases from JMP or JBP Applications” on page 34
- “Program Switching in JMP and JBP Applications” on page 35

Running the IMS Java IVP in a JMP Region

To verify that IMS Java is properly installed and that the JMP region is properly configured, run the IMS Java IVP. Details about the PROCLIB members and procedure parameters are in *IMS Version 9: Installation Volume 2: System Definition and Tailoring*.

Prerequisites:

- Ensure that the standard IMS IVPs have been run. These IVPs prepare the DBD for the IVP database, named DFSIVD2, and load the IVP database. They also prepare the IMS Java application PSB (named DFSIVP37), build ACBs, prepare the MFS format (named DFSIVF37), and prepare other IMS control

blocks required by the IMS Java IVPs. For details about how to run the IMS IVPs, see *IMS Version 9: Installation Volume 1: Installation Verification*.

- "Installing IMS Java" on page 2

To run the IMS Java IVP in a JMP region:

1. Edit the sample JVM member DFSJVMMS, which is in IMS.SDFSISRC:
   a. For -Dibm.jvm.trusted.middleware.class.path=, change "ImsjavaPath" to *pathprefix*/usr/lpp/ims/imsjava91
   b. For -Dibm.jvm.shareable.application.class.path=, change "SamplePath" to *pathprefix*/usr/lpp/ims/imsjava91
   c. If you are using SDK 1.4.1, add the following JVM property:
      `-Djava.endorsed.dirs=`*`pathprefix`*`/usr/lpp/ims/imsjava91/lib`

      The IVP tests that the required XML files are installed and in the classpath. If you do not do this task, you will receive an error when you run the IVP.
2. Edit the sample JVM member DFSJVMEV, which is in IMS.SDFSISRC:
   a. Change "JavaHome" to the SDK directory. For example: `usr/lpp/java/j1.3`
   b. Change "imsjavaPath" to *pathprefix*/usr/lpp/ims/imsjava91
3. Create two HFS files: one for the JMP output and one for errors. The following sample job creates the files JVM.out and JVM.err:

```
//name     JOB parameters
//TCHMOD   PROC TPARM=
//BPX      EXEC PGM=BPXBATCH,PARM='&TPARM'
//SYSPRINT DD  SYSOUT=*
//STDOUT   DD  PATH='path/tchmod.out',
//         PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
//STDERR   DD  PATH='path/tchmod.err',
//         PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
//         PEND
//STEP1    EXEC TCHMOD,TPARM='sh touch path/JVM.out'
//*
//STEP2    EXEC TCHMOD,
//           TPARM='sh chmod 777 path/JVM.out'
//*
//STEP3    EXEC TCHMOD,TPARM='sh touch path/JVM.err'
//*
//STEP4    EXEC TCHMOD,TPARM='sh chmod 777 path/JVM.err'
```

4. Edit the DFSJMP procedure, which is in IMS.PROCLIB:
   a. Set the JAVAOUT and JAVAERR DD statements to point to the JVM.out and JVM.err files. For example:

```
//JAVAOUT  DD  PATH='/path/JVM.out'
//JAVAERR  DD  PATH='/path/JVM.err'
```

   b. Set the STEPLIB DD statement to point to the SDFSJLIB data set. This data set contains the DFSCLIB member.
   c. Set the PROCLIB DD statement to point to the SDFSISRC data set. This data set contains the DFSJVMMS, DFSJVMWK, DFSJVMEV, and DFSJVMAP members.
   d. Set the following parameters:
      - JVMOPMAS= data set member DFSJVMMS (master JVM options)
      - JVMOPWKR= data set member DFSJVMWK (worker JVM options)
      - ENVIRON= data set member DFSJVMEV (LIBPATH options)
      - XPLINK=Y if you use SDK 1.4.1

e. Set any other parameters that are required by your installation. For complete information about the available parameters and DD statements for the DFSJMP procedure, see the *IMS Version 9: Installation Volume 2: System Definition and Tailoring*.

5. Run the JMP procedure.

   The JMP region is started.

6. From an IMS terminal, invoke the formatted screen for the transaction by issuing the following command:

   ```
   /format IVTCM
   ```

   An input screen, as shown in Figure 2, is displayed.

```
    ************************************************
    *     IMS INSTALLATION VERIFICATION PROCEDURE    *
    ************************************************


                             TRANSACTION TYPE : CONVERSATIONAL
                             DATE             : 12/11/04

PROCESS  CODE  (*1) :
                                                  (*1) PROCESS CODE
LAST  NAME          :                                RUNIVP
                                                     ADD
FIRST  NAME         :                                DELETE
                                                     UPDATE
EXTENSION  NUMBER   :                                DISPLAY
                                                     END
INTERNAL  ZIP CODE  :


                                                  SEGMENT# :
```

Figure 2. IVP Screen for IMS Java JMP

7. In the PROCESS CODE field, type: `RUNIVP`

   If the IVP was successful, it displays IVP PASSED.

   If the IVP was not successful, it displays IVP FAILED or IVP INCOMPLETE.

   See the JVM.out file for the results of the individual tests that are performed by the IVP.

8. Optionally, move the JVM.out and JVM.err files from HFS to a partitioned data set member by submitting the following job:

   ```
   //name     JOB
   //MV2PSD   EXEC PGM=IKJEFT01
   //SYSPRINT DD SYSOUT=*
   //SYSTSPRT DD SYSOUT=*
   //O1       DD DISP=SHR,DSN=hlq.dataset(JVMOUT)
   //I1       DD PATH='pathPrefix/JVM.out'
   //O2       DD DISP=SHR,DSN=hlq.dataset(JVMERR)
   //I2       DD PATH='pathPrefix/JVM.err'
   //SYSTSIN  DD *
   OCOPY INDD(I1) OUTDD(O1)
   OCOPY INDD(I2) OUTDD(O2)
   OCOPY INDD(I3) OUTDD(O3)
   /*
   ```

You can also use this application as a phonebook sample. From the input screen, you can enter the process codes that are listed on the right side of the screen.

Running the IMS Java IVP in a JBP Region

To verify that IMS Java is properly installed and that the JBP region is properly configured, run the IMS Java IVP. Details about the PROCLIB members and procedure parameters are in *IMS Version 9: Installation Volume 2: System Definition and Tailoring*.

Prerequisites:

- Ensure that the standard IMS IVPs have been run. These IVPs prepare the DBD for the IVP database, named DFSIVD2, and load the IVP database. They also prepare the IMS Java application PSB (named DFSIVP67) and prepare other IMS control blocks required by the IMS Java IVPs. For details about how to run the IMS IVPs, see *IMS Version 9: Installation Volume 1: Installation Verification*.
- “Installing IMS Java” on page 2

To run the IMS Java IVP in a JBP region:

1. Edit the sample JVM member DFSJVMMS, which is in IMS.SDFSISRC:
   a. For -Dibm.jvm.trusted.middleware.class.path=, change “ImsjavaPath” to *pathprefix*/usr/lpp/ims/imsjava91
   b. For -Dibm.jvm.shareable.application.class.path=, change “SamplePath” to *pathprefix*/usr/lpp/ims/imsjava91
   c. If you are using SDK 1.4.1, add the following JVM property:
      `-Djava.endorsed.dirs=`*`pathprefix`*`/usr/lpp/ims/imsjava91/lib`

      The IVP tests that the required XML files are installed and in the classpath. If you do not do this task, you will receive an error when you run the IVP.
2. Edit the sample JVM member DFSJVMEV, which is in IMS.SDFSISRC:
   a. Change “JavaHome” to the SDK directory. For example:
      `/usr/lpp/java/j1.3`
   b. Change “imsjavaPath” to *pathprefix*/usr/lpp/ims/imsjava91
3. Create two HFS files: one for the JBP output and one for errors. The following sample job creates the files JVM.out and JVM.err:

```
//name     JOB parameters
//TCHMOD   PROC TPARM=
//BPX      EXEC PGM=BPXBATCH,PARM='&TPARM'
//SYSPRINT DD  SYSOUT=*
//STDOUT   DD  PATH='path/tchmod.out',
//         PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
//STDERR   DD  PATH='path/tchmod.err',
//         PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
//         PEND
//STEP1    EXEC TCHMOD,TPARM='sh touch path/JVM.out'
//*
//STEP2    EXEC TCHMOD,
//           TPARM='sh chmod 777 path/JVM.out'
//*
//STEP3    EXEC TCHMOD,TPARM='sh touch path/JVM.err'
//*
//STEP4    EXEC TCHMOD,TPARM='sh chmod 777 path/JVM.err'
```

4. Edit the DFSJBP procedure, which is in IMS.PROCLIB:
   a. Set the JAVAOUT and JAVAERR DD statements to point to the JVM.out and JVM.err files. For example:

```
//JAVAOUT  DD  PATH='/path/JVM.out'
//JAVAERR  DD  PATH='/path/JVM.err'
```

b. Set the STEPLIB DD statement to point to the SDFSJLIB data set. This data set contains the DFSCLIB member.
c. Set the PROCLIB DD statement to point to the SDFSISRC data set. This data set contains the DFSJVMMS, DFSJVMWK, DFSJVMEV, and DFSJVMAP members.
d. Set the following parameters:
   - JVMOPMAS= data set member DFSJVMMS (master JVM options)
   - JVMOPWKR= data set member DFSJVMWK (worker JVM options)
   - ENVIRON= data set member DFSJVMEV (LIBPATH options)
   - XPLINK=Y if you use SDK 1.4.1
e. Set the following EXEC statement parameters to the following: PSB=DFSIVP67 and MBR=DFSJBP.
f. Set any other parameters that are required by your installation. For complete information about the available parameters and DD statements for the DFSJBP procedure, see the *IMS Version 9: Installation Volume 2: System Definition and Tailoring*.

5. Run the JBP procedure.

   The JBP region is started, the IVP runs, and output is sent to the JVM.out file.

6. Optionally, move the JVM.out and JVM.err files from HFS to a partitioned data set member by submitting the following job:

```
//name     JOB
//MV2PSD   EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//O1       DD DISP=SHR,DSN=hlq.dataset(JVMOUT)
//I1       DD PATH='pathPrefix/JVM.out'
//O2       DD DISP=SHR,DSN=hlq.dataset(JVMERR)
//I2       DD PATH='pathPrefix/JVM.err'
//SYSTSIN  DD *
OCOPY INDD(I1) OUTDD(O1)
OCOPY INDD(I2) OUTDD(O2)
OCOPY INDD(I3) OUTDD(O3)
/*
```

7. Check the JVMOUT data set or JVM.out file.

   If the IVP was successful, it displays IVP PASSED.

   If the IVP was not successful, it displays IVP FAILED or IVP INCOMPLETE.

Running the IMS Java Sample Application from a JMP Region

The IMS Java sample application can run in a JMP region. The sample application processes the sample dealership database based on six process codes. This sample is not MFS-formatted. Therefore, you must run this sample by submitting the transaction code, followed by a process code and other input data, from an IMS terminal.

The source files for the sample application are in the HFS directory *pathprefix*/usr/lpp/ims/imsjava91/samples/dealership/ims.

Prerequisites:

- “Running the IMS Java IVP in a JMP Region” on page 10
- “Preparing to Run the Dealership Samples” on page 165

To run the IMS Java sample application from a JMP region:

1. Edit the sample JVM member DFSJVMAP by adding the following line:

   ```
   AUTPSB11=samples/dealership/ims/IMSAuto
   ```

2. Following the directions provided in the sample JVM members, edit the following three sample JVM members, which are in IMS.SDFSISRC: DFSJVMAP, DFSJVMMS, and DFSJVMEV.

3. Create two HFS files: one for the JMP output and one for errors. The following sample job creates the files JVM.out and JVM.err:

   ```
   //name     JOB parameters
   //TCHMOD   PROC TPARM=
   //BPX      EXEC PGM=BPXBATCH,PARM='&TPARM'
   //SYSPRINT DD  SYSOUT=*
   //STDOUT   DD  PATH='path/tchmod.out',
   //         PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
   //STDERR   DD  PATH='path/tchmod.err',
   //         PATHOPTS=(OWRONLY,OCREAT,OTRUNC),PATHMODE=SIRWXU
   //         PEND
   //STEP1    EXEC TCHMOD,TPARM='sh touch path/JVM.out'
   //*
   //STEP2    EXEC TCHMOD,
   //           TPARM='sh chmod 777 path/JVM.out'
   //*
   //STEP3    EXEC TCHMOD,TPARM='sh touch path/JVM.err'
   //*
   //STEP4    EXEC TCHMOD,TPARM='sh chmod 777 path/JVM.err'
   ```

4. Edit the DFSJMP procedure, which is in IMS.PROCLIB:

   a. Set the JAVAOUT and JAVAERR DD statements to point to the files that are created in step 3. For example:

      ```
      //JAVAOUT  DD  PATH='/path/JVM.out'
      //JAVAERR  DD  PATH='/path/JVM.err'
      ```

   b. Set the STEPLIB DD statement to point to the SDFSJLIB data set. This data set contains the DFSCLIB member.

   c. Set the PROCLIB DD statement to point to the SDFSISRC data set. This data set contains the DFSJVMMS, DFSJVMWK, DFSJVMEV, and DFSJVMAP members.

   d. Set the following parameters:

      - JVMOPMAS= data set member DFSJVMMS (master JVM options)
      - JVMOPWKR= data set member DFSJVMWK (worker JVM options)
      - ENVIRON= data set member DFSJVMEV (LIBPATH options)
      - XPLINK=Y if you use SDK 1.4.1

   e. Set any other parameters that are required by your installation. For complete information about the available parameters and DD statements for the DFSJMP procedure, see the *IMS Version 9: Installation Volume 2: System Definition and Tailoring*.

5. Run the JMP procedure.

   The JMP region is started.

6. From an IMS terminal, run the application by issuing the transaction code AUTRAN11, one of the following six transaction codes, and input data. The spacing is important.

   - LISTMODELS. For example:

     ```
     AUTRAN11 LISTMODELS
     ```

   - FINDCAR. For example:

     ```
     AUTRAN11 FINDCAR            FORD V234567890123456789V
     ```

   - MODELDETAILS. For example:

```
AUTRAN11 MODELDETAILS       VOLVO     S40       2002
```

- RECORDSALE. For example:

```
AUTRAN11 RECORDSALE         1235 S999302042002LAST9                  >
V98765432112345678 2VVOLVO     S40
```

- ACCEPTORDER. For example:

```
AUTRAN11 ACCEPTORDER        123457LAST9                    FIRST9     >
            05-18-200111:23:34
```

- CANCELORDER. For example:

```
AUTRAN11 CANCELORDER        1234571235
```

- RETRIEVEXML. For example:

```
AUTRAN11 RETRIEVEXML        1235
```

Configuring JMP and JBP Regions for DB2 UDB for z/OS Database Access

This topic describes how to set up a JMP or JBP region to access DB2 UDB for z/OS databases. It does not describe how to set up DB2 UDB for z/OS for access from IMS. For information about setting up DB2 UDB for z/OS for Java application access, see *DB2 Universal Database for OS/390 and z/OS: Application Programming Guide and Reference for Java*. Note that you must create a DB2 plan for each PSB (usually each Java application) that is used to access DB2 UDB for z/OS.

JMP and JBP applications can access DB2 UDB for z/OS databases. For JMP or JBP applications to have DB2 UDB for z/OS access, you must attach DB2 UDB for z/OS to IMS using the DB2 Recoverable Resource Manager Services attachment facility (RRSAF). Unlike other dependent regions, JMP and JBP regions do not use the External Subsystem Attach Facility (ESAF).

DB2 UDB for z/OS provides different JDBC drivers:

- JDBC/SQLJ driver for DB2 for OS/390 and z/OS with JDBC 2.0 support (called the DB2 JDBC/SQLJ 2.0 driver), which allows access to DB2 UDB for z/OS databases only when IMS is on the same z/OS image as DB2 UDB for z/OS. This is a type 2 JDBC driver.
- JDBC/SQLJ driver for DB2 for OS/390 and z/OS with JDBC 1.2 support (called the DB2 JDBC/SQLJ 1.2 driver), which allows access to DB2 UDB for z/OS databases only when IMS is on the same z/OS image as DB2 UDB for z/OS. This is a type 2 JDBC driver.
- DB2 Universal JDBC driver, which allows access to DB2 UDB for z/OS databases from IMSs that are on different z/OS images from DB2 UDB for z/OS when you use the Universal Driver type 4 connectivity. You can also use the type 2 implementation of this driver for access to DB2 UDB for z/OS databases when IMS is on the same z/OS image as DB2 UDB for z/OS.

All of these drivers are referred to in this topic as DB2 JDBC drivers.

For type 2 JDBC drivers, you must use the default connection URL in the application program. For example, `jdbc:db2os390:` or `db2:default:connection`.

For type 4 JDBC drivers, you can use a specific connection URL in the application program.

With RRSAF, the dependent region builds an attachment thread to DB2 UDB for z/OS using RRS. RRS coordinates the commits of the updates that the application program makes to both IMS and DB2 UDB for z/OS resources. IMS is a participant, not the coordinator, of these updates and commits.

To attach a DB2 UDB for z/OS subsystem to IMS using RRSAF for JMP or JBP access to DB2 UDB for z/OS databases:

1. Create an IMS.PROCLIB member for information about the DB2 UDB for z/OS subsystem. The member name must follow the same naming conventions you follow when you attach DB2 UDB for z/OS with ESAF.

   In the IMS.PROCLIB member, define the following three parameters for the DB2 subsystem that JMP and JBP applications need access to:

   ```
   SST=DB2,SSN=db2name,COORD=RRS
   ```

2. In the trusted middleware class path of the DFSJVMMS member, which is IMS.SDFSISRC, add the following paths:
   - Path to the ZIP file of the DB2 JDBC driver
   - Path to the ZIP file and ZIP file name of the DB2 JDBC driver

   For example:

   ```
   -Dibm.jvm.trusted.middleware.class.path=>
   /usr/lpp/db2/db2710/classes: >
   /usr/lpp/db2/db2710/classes/db2j2classes.zip
   ```

3. In the DFSJVMEV member, which is IMS.SDFSISRC, add the path to the SO file of the DB2 JDBC driver to the LIBPATH= environment variable. For example:

   ```
   LIBPATH=/usr/lpp/db2/db2710/lib
   ```

4. Add the following parameters to the IMS control region EXEC statement:

   ```
   SSM=name
   RRS=Y
   ```

5. In the DFSJMP or DFSJBP procedure of the region that has access to DB2 UDB for z/OS, add the DFSDB2AF DD statement to point to the DB2 UDB for z/OS libraries, which must be APF-authorized.

Related Reading: For details about the IMS.PROCLIB member and procedure parameters, see the external subsystem information of *IMS Version 9: Installation Volume 2: System Definition and Tailoring*. For information about the DB2 JDBC drivers, see *DB2 Universal Database for OS/390 and z/OS: Application Programming Guide and Reference for Java*.

Developing JMP Applications

JMP applications access the IMS message queue to receive messages to process and to send output messages. Therefore, you must define input and output message classes by subclassing the `IMSFieldMessage` class. The IMS Java class libraries provide the capability to process `IMSFieldMessage` objects. JMP applications commit or roll back the processing of each message by calling `IMSTransaction.getTransaction().commit()` or `IMSTransaction.getTransaction().rollback()`.

Related Reading: For details about the classes you use to develop a JMP application, see the IMS Java API Specification, which is available on the IMS Java Web site. Go to http://www.ibm.com/ims and link to the IMS Java page.

The following topics provide additional information:

- "Subclassing the IMSFieldMessage Class to Define Input Messages"
- "Subclassing the IMSFieldMessage Class to Define Output Messages"
- "Implementing the main Method" on page 19
- "JMP Programming Models" on page 20
- "Additional Message Handling Considerations for JMP Applications" on page 22

Subclassing the IMSFieldMessage Class to Define Input Messages

Figure 3 gives an example of subclassing the `IMSFieldMessage` class. This class defines an input message that accepts a 2-byte type code of a car model to query a car dealership database for available car models.

This example code subclasses the `IMSFieldMessage` class to make the fields in the message available to the program and creates an array of `DLITypeInfo` objects for the fields in the message. For the `DLITypeInfo` class, the code identifies first the field name, then the data type, the position, and finally the length of the individual fields within the array. This allows the application to use the access functions within the `IMSFieldMessage` class hierarchy to automatically convert the data from its format in the message to a Java type that the application can process. In addition to the message-specific fields it defines, the `IMSFieldMessage` class provides access functions that allow it to determine the transaction code and the length of the message.

```
package dealership.application;
import com.ibm.ims.db.*;
import com.ibm.ims.base.*;
import com.ibm.ims.application.*;

/* Subclasses IMSFieldMessage to define application's input messages */
public class InputMessage extends IMSFieldMessage {

    /* Creates array of DLITypeInfo objects for the fields in message */
    final static DLITypeInfo[]fieldInfo={
        new DLITypeInfo("ModelTypeCode", DLITypeInfo.CHAR, 1, 2)
    };

    public InputMessage() {
        super(fieldInfo, 2, false);
    }
}
```

Figure 3. Subclass IMSFieldMessage: Input Message Sample Code

Subclassing the IMSFieldMessage Class to Define Output Messages

Figure 4 on page 19 gives a sample of subclassing `IMSFieldMessage` to define an output message that displays the available car models from a type code query.

This sample code creates an array of `DLITypeInfo` objects and then passes that array, the byte array length, and the boolean value false, which indicates a non-SPA message, to the `IMSFieldMessage` constructor. For each `DLITypeInfo` object, you must first identify the field data type, then the field name, the field offset in the byte array, and finally the length of the byte array.

```
package dealership.application;
import com.ibm.ims.db.*;
import com.ibm.ims.base.*;
import com.ibm.ims.application.*;

/*Subclasses IMSFieldMessage to define application's output messages */
public class ModelOutput extends IMSFieldMessage {

        /* Creates array of DLITypeInfo objects for the fields in message */
    final static DLITypeInfo[] fieldInfo={
        new DLITypeInfo("Type",         DLITypeInfo.CHAR,    1,  2),
        new DLITypeInfo("Make",         DLITypeInfo.CHAR,    3, 10),
        new DLITypeInfo("Model",        DLITypeInfo.CHAR,   13, 10),
        new DLITypeInfo("Year",         DLITypeInfo.DOUBLE, 23,  4),
        new DLITypeInfo("CityMiles",    DLITypeInfo.CHAR,   27,  4),
        new DLITypeInfo("HighwayMiles", DLITypeInfo.CHAR,   31,  4),
        new DLITypeInfo("Horsepower",   DLITypeInfo.CHAR,   35,  4)
    };

    public ModelOutput() {
        super(fieldInfo, 38,false);
    }

}
```

Figure 4. Subclass IMSFieldMessage: Output Message Sample Code

Implementing the main Method

The `main` method (`public static void main(String[] args)`) is the entry point into all JMP and JBP applications.

The sample code shown in Figure 5 on page 20 demonstrates how to perform the following actions:

1. Query the database for a specific model that matches the input model type code. This method is not implemented yet and is explained more fully in Chapter 7, “JDBC Access to IMS Data,” on page 125.
2. Return detailed information about that specific model as output if it is available at the dealership.
3. Return an error message if the model is not available at the dealership.

```
package dealership.ims;
import com.ibm.ims.application.*;

public static void main(String args[]) {
    IMSMessageQueue messageQueue = null;
    InputMessage inputMessage = null;
    ModelOutput modelOutput = null;

    messageQueue = new IMSMessageQueue();
    inputMessage = new InputMessage();
    modelOutput = new ModelOutput();

    while(messageQueue.getUniqueMessage(inputMessage)) {
        if (!inputMessage.getString
                                    ("ModelTypeCode").trim().equals("")){
                if (getModelDetails(inputMessage, modelOutput))      // 1
                        messageQueue.insertMessage(modelOutput);           // 2
        }

        else {
                reply("Invalid Input");                              // 3
        }

        IMSTransaction.getTransaction().commit();

    }

    public void reply(String errmsg) throws IMSException{
        ErrorMessage errorMessage = new ErrorMessage();
        errorMessage.setString("MessageText",errmsg);
        messageQueue.insertMessage(errorMessage);
   }

}
```

Figure 5. `main` Method Sample Code

Note: The `IMSMessageQueue.getUniqueMessage` method returns **true** if a message was read from the queue and **false** if one was not. Also, the `IMSTransaction.getTransaction().commit` method must be called before receiving subsequent messages from the queue.

JMP Programming Models

JMP applications get input messages from the IMS message queue, access IMS databases, commit transactions, and can send output messages.

JMP applications are started when IMS receives a message with a transaction code for the JMP application and schedules the message. JMP applications end when there are no more messages with that transaction code to process.

JMP Application Without Rollback

A transaction begins when the application gets an input message and ends when the application commits the transaction. To get an input message, the application calls the `getUniqueMessage` method. The application must commit or rollback any database processing. The application must issue a commit call immediately before calling subsequent `getUniqueMessage` methods.

```
public static void main(String args[]) {

    conn = DriverManager.getConnection(...); //Establish DB connection


    while(MessageQueue.getUniqueMessage(...)){ //Get input message, which
                                               //starts transaction
```

```
        results=statement.executeQuery(...); //Perform DB processing
        ...
        MessageQueue.insertMessage(...);     //Send output messages
        ...
        IMSTransaction.getTransaction().commit();  //Commit and end transaction
    }

    conn.close();                            //Close DB connection
    return;
}
```

JMP Application that Uses Rollback

A JMP application can roll back database processing and output messages any number of times during a transaction. A rollback call backs out all database processing and output messages to the most recent commit. The transaction must end with a commit call when the program issues a rollback call, even if no further database or message processing occurs after the rollback call.

```
public static void main(String args[]) {

    conn = DriverManager.getConnection(...); //Establish DB connection

    while(MessageQueue.getUniqueMessage(...)){   //Get input message, which
                                                 //starts transaction

        results=statement.executeQuery(...); //Perform DB processing
        ...
        MessageQueue.insertMessage(...);     //Send output messages
        ...
        IMSTransaction.getTransaction().rollback();  //Roll back DB processing
                                                     //and output messages

        results=statement.executeQuery(...); //Perform more DB processing
                                             //(optional)
        ...
        MessageQueue.insertMessage(...);     //Send more output messages
                                             //(optional)
        ...
        IMSTransaction.getTransaction().commit(); //Commit and end transaction
    }

    conn.close();                            //Close DB connection
    return;
}
```

JMP Application that Accesses IMS or DB2 UDB for z/OS Data

When a JMP application accesses only IMS data, it needs to open a database connection only once to process multiple transactions, as shown in “JMP Application Without Rollback” on page 20. However, a JMP application that accesses DB2 UDB for z/OS data must open and close a database connection for each message that is processed. The following model is valid for DB2 UDB for z/OS database access, IMS database access, or both DB2 UDB for z/OS and IMS database access.

Related Reading: For more information about accessing DB2 data from a JMP application, see “Accessing DB2 UDB for z/OS Databases from JMP or JBP Applications” on page 34.

```
public static void main(String args[]) {

    while(MessageQueue.getUniqueMessage(...)){   //Get input message, which
                                                 //starts transaction

        conn = DriverManager.getConnection(...); //Establish DB connection
```

```
            results=statement.executeQuery(...);     //Perform DB processing
            ...
            MessageQueue.insertMessage(...);         //Send output messages
            ...
            conn.close();                            //Close DB connection

            IMSTransaction.getTransaction().commit(); //Commit & end transaction
        }

        return;
    }
```

Additional Message Handling Considerations for JMP Applications

JMP applications access the IMS message queue in addition to IMS or DB2 UDB for z/OS databases. This topic provides information about specific programming considerations for the IMS message queue.

In this topic:

- "Conversational Transactions"
- "Handling Multi-Segment Messages" on page 24
- "Coding and Accessing Messages with Repeating Structures" on page 25
- "Flexible Reading of Multiple Input Messages" on page 26

Conversational Transactions

A conversational program runs in a JMP region and processes conversational transactions that are made up of several steps. It does not process the entire transaction at the same time. A conversational program divides processing into a connected series of terminal-to-program-to-terminal interactions. Use conversational processing when one transaction contains several parts.

A nonconversational program receives a message from a terminal, processes the request, and sends a message back to the terminal. A conversational program receives a message from a terminal and replies to the terminal, but it saves the data from the transaction in a scratch pad area (SPA). Then, when the person at the terminal enters more data, the program has the data it saved from the last message in the SPA, so it can continue processing the request without the person at the terminal having to enter the data again. The application package classes enable applications to be built using IMS Java.

Related Reading: For more information about conversational and nonconversational transaction processing, see *IMS Version 9: Administration Guide: Transaction Manager*.

Defining a SPA Message in a Conversational Program: To define a SPA message in a conversational program:

1. Define the SPA message (including the boolean as a SPA parameter). By default, all messages going to (input) and from (output) a Java application are transmitted as EBCDIC character data. To use a different type of encoding, you must call the `IMSFieldMessage` class inherited method `setDefaultEncoding` and provide the new encoding type. This encoding can be any Java-supported encoding type. In Figure 6 on page 23, the default encoding is specified as UTF-8.

```
public class SPAMessage extends IMSFieldMessage {
  static DLITypeInfo[] fieldInfo = {
    new DLITypeInfo("SessionNumber",DLITypeInfo.SMALLINT,1, 2),
    new DLITypeInfo("ProcessCode",  DLITypeInfo.CHAR,    3, 8),
    new DLITypeInfo("LastName",     DLITypeInfo.CHAR,   11,10),
    new DLITypeInfo("FirstName",    DLITypeInfo.CHAR,   21,10),
    new DLITypeInfo("Extension",    DLITypeInfo.CHAR,   31,10),
    new DLITypeInfo("ZipCode",      DLITypeInfo.CHAR,   41, 7),
    new DLITypeInfo("Reserved",     DLITypeInfo.CHAR,   48,19) };
public SPAMessage() {
  super(fieldInfo, 66, true);
  setDefaultEncoding("UTF-8");
}
}
```

Figure 6. Defining a SPA Message

2. Read the SPA message before reading the application messages:

```
try {
    // Get the SPA data
    msgReceived = msgQ.getUniqueMessage(spaMessage);
  }
  catch (IMSException e)
  {
    if (e.getStatusCode() !=
                      JavaToDLI.MESSAGE_QUEUED_PRIOR_TO_LAST_START)
      throw e;
  }
  if (!msgReceived)
    outputMessage.setString("Message","UNABLE TO READ SPA");
  else if (!msgQ.getNextMessage(inputMessage))
    // No input message received
    outputMessage.setString("Message","NO INPUT MESSAGE");
  else if ((spaMessage.getShort("SessionNumber")==0)
    && (!inputMessage.getString("ProcessCode").trim().equals("END"))
    && (inputMessage.getString("LastName").trim().equals("")))
    // New Conversation. User has to specify last name.
    outputMessage.setString("Message","LAST NAME WAS NOT SPECIFIED");
  else {
  {
```

Figure 7. Reading a SPA Message

3. Write the SPA message before sending any output messages:

```
 // Set spa data fields
    spaMessage.setString("ProcessCode",
                         inputMessage.getString("ProcessCode"));
    spaMessage.setString("LastName",
                         inputMessage.getString("LastName"));
    spaMessage.setString("FirstName",
                         inputMessage.getString("FirstName"));
    spaMessage.setString("Extension",
                         inputMessage.getString("Extension"));
    spaMessage.setString("ZipCode",
                         inputMessage.getString("ZipCode"));
    spaMessage.incrementSessionNumber();
    msgQ.insertMessage(spaMessage);
```

Figure 8. Writing a SPA Message

4. End the conversation by using the version of the `insertMessage` method that contains a boolean `isLast` argument set to true:

   ```
   msgQ.insertMessage(spaMessage, true);
   ```

Conversational Transaction Sequence of Events: When the message is a conversational transaction, the following sequence of events occurs:

1. IMS removes the transaction code and places it at the beginning of a message segment. The message segment is equal in length to the SPA that was defined for this transaction during system definition. This is the first segment of the input message that is made available to the program. The second through the *n*th segments from the terminal, minus the transaction code, become the remainder of the message that is presented to the application program.
2. After the conversational program prepares its reply, it inserts the SPA to IMS. The program then inserts the actual text of the reply as segments of an output message.
3. IMS saves the SPA and routes the message to the input LTERM (logical terminal).
4. If the SPA insert specifies that another program is to continue the same conversation, the total reply (including the SPA) is retained on the message queue as input to the next program. This program then receives the message in a similar form.
5. A conversational program must be scheduled for each input exchange. The other processing continues while the operator at the input terminal examines the reply and prepares new input messages.
6. To terminate a conversation, the program places blanks in the transaction code field of the SPA and inserts the SPA to IMS. In IMS Java this happens when you call `IMSMessageQueue.insertMessage` with the boolean parameter `isLast` set to true.
7. The conversation can also be terminated if the transaction code in the SPA is replaced by any nonconversational program's transaction code, and the SPA is inserted to IMS. After the next terminal input, IMS routes that message to the other program's queue in the normal way.

Handling Multi-Segment Messages

Message-driven applications can have multi-segment input messages. That is, more than one message needs to be read from the message queue in order to retrieve the entire message. When this occurs, you must provide a mapping for each message that is to be read from the queue and use the appropriate methods available from the `IMSMessageQueue` class.

The following code defines two input messages that comprise a multi-segment message:

```
public class InputMessage1 extends IMSFieldMessage {

    final static DLITypeInfo[] segmentInfo = {
        new DLITypeInfo("Field1", DLITypeInfo.CHAR, 1, 10),
        new DLITypeInfo("Field2", DLITypeInfo.INTEGER, 11, 4)
    };

    public InputMessage1() {
        super(segmentInfo, 14, false);
    }
}

public class InputMessage2 extends IMSFieldMessage {

    final static DLITypeInfo[] segmentInfo = {
        new DLITypeInfo("Field1", DLITypeInfo.CHAR, 1, 10),
        new DLITypeInfo("Field2", DLITypeInfo.CHAR, 11, 8)
    };
```

```
    public InputMessage2() {
        super(segmentInfo, 18, false);
    }
}
```

The following code shows how the message queue is used to retrieve both messages:

```
//Create a message queue
IMSMessageQueue messageQueue = new IMSMessageQueue();
//Create the first input message
InputMessage1 input1 = new InputMessage1();
//Create the second input message
InputMessage2 input2 = new InputMessage2();

try {
    //Read the first message from the queue
    messageQueue.getUniqueMessage(input1);
    ...
    //Read the second message from the queue
    messageQueue.getNextMessage(input2);
    ...
} catch (IMSException e) {
    ...
}
```

Coding and Accessing Messages with Repeating Structures

Messages with repeating structures can be defined by using the `DLITypeInfoList` class. With the `DLITypeInfoList` class, you can specify a repeating list of fields and the maximum number of times the list can be repeated. These repeating structures can contain repeating structures.

Figure 9 is a sample output message that contains a set of Make, Model, and Color fields, with a count field to identify how many occurrences were stored:

```
public class ModelOutput extends IMSFieldMessage {
static DLITypeInfo[] modelTypeInfo = {
     new DLITypeInfo("Make", DLITypeInfo.CHAR, 1, 20),
     new DLITypeInfo("Model", DLITypeInfo.CHAR, 21, 20),
     new DLITypeInfo("Color", DLITypeInfo.CHAR, 41, 20),
     };
static DLITypeInfo[] modelTypeInfoList = {
     new DLITypeInfo("ModelCount", DLITypeInfo.INTEGER, 1, 4),
     new DLITypeInfoList("Models", modelTypeInfo, 5, 60, 100),
     };
public ModelOutput() {
   super(modelOutputTypeInfo, 6004, false);
} }
```

Figure 9. Sample Output Message with Repeating Structures

To access the nested structures that are defined in a `DLITypeInfoList` object, use a dotted notation to specify the fields and the index of the field within a repeating structure. This dotted notation can use either the field names or field indexes. For example, the “Color” field in the fourth “Models” definition in the `ModelOutput` object is accessed as “Models.4.Color” within the `ModelOutput` message. The following code sets the fourth “Color” in the `ModelOutput` message to “Red.”

```
ModelOutput output= new ModelOutput();
output.setString("Models.4.Color", "Red");
```

The following code uses field indexes instead of field names to make the same change to the `ModelOutput` message:

```
ModelOutput output= new ModelOutput();
output.setString("2.4.3", "Red");
```

Flexible Reading of Multiple Input Messages

There are times when an application needs to process multiple input messages that require different input data types. For example, the car dealership sample application supports requests to list models, show model details, find cars, cancel orders, and record sales. Each of these requests requires different input data. The following steps explain how to define the messages to support these requests, and how to access the messages from the application.

1. Define the primary input message. The primary input message is the message that you pass to the `IMSMessageQueue.getUniqueMessage` method to retrieve all of your input messages. Your primary input message must have an I/O area that is large enough to contain any of the input requests that your application might receive. It must also contain at least one field in common with all of your input messages. This common field allows you to determine the input request. In the example in Figure 10, the common field is `CommandCode`, and the maximum length of each message is 64 (the number passed to the `IMSFieldMessage` constructor):

```
public class InputMessage extends IMSFieldMessage {

    final static DLITypeInfo[] fieldInfo = {

    new DLITypeInfo("CommandCode", DLITypeInfo.CHAR, 1, 20), A };

    public InputMessage(DLITypeInfo[] fieldInfo)
    {
        super(fieldInfo, 64, false); B
    }
}
```

Figure 10. Defining the Primary Input Message

2. Define separate input messages for each request. Each of these input messages contains the same `CommandCode` field as its first field. Each of these input messages also uses an `IMSFieldMessage` constructor that takes an `IMSFieldMessage` object and a `DLITypeInfo` array. The `IMSFieldMessage` constructor allows you to remap the contents of the primary input message using the same type of information with each request; therefore, you do not copy the I/O area of the message, only a reference to this area. Figure 11 on page 27 illustrates code that creates the input messages for the requests `ShowModelDetails`, `FindACar`, and `CancelOrder`.

```
public class ShowModelDetailsInput extends IMSFieldMessage {
   final static DLITypeInfo[] fieldInfo = {
      new DLITypeInfo("CommandCode", DLITypeInfo.CHAR, 1, 20), C
      new DLITypeInfo("ModelTypeCode", DLITypeInfo.CHAR, 21, 2),
   };
public ShowModelDetailsInput(InputMessage inputMessage) { D
              super(inputMessage, fieldInfo);
}
}
public class FindACarInput extends IMSFieldMessage {
   final static DLITypeInfo[] fieldInfo = {
      new DLITypeInfo("CommandCode", DLITypeInfo.CHAR, 1, 20), E
      new DLITypeInfo("Make", DLITypeInfo.CHAR, 21, 10),
      new DLITypeInfo("Model", DLITypeInfo.CHAR, 31, 10),
      new DLITypeInfo("Year", DLITypeInfo.CHAR, 41, 4),
      new DLITypeInfo("LowPrice", DLITypeInfo.PACKEDDECIMAL, 45, 5),
      new DLITypeInfo("HighPrice", DLITypeInfo.PACKEDDECIMAL, 50, 5),
      new DLITypeInfo("Color", DLITypeInfo.CHAR, 55, 10),
 };
public FindACarInput(InputMessage inputMessage) { F
   super(inputMessage, fieldInfo);
}
}
public class CancelOrderInput extends IMSFieldMessage {
   final static DLITypeInfo[] fieldInfo = {
      new DLITypeInfo("CommandCode", DLITypeInfo.CHAR, 1, 20), G
      new DLITypeInfo("OrderNumber", DLITypeInfo.CHAR, 21, 6),
      new DLITypeInfo("DealerNumber", DLITypeInfo.CHAR, 21, 6),
   };
public CancelOrderInput(InputMessage inputMessage) H
{
   super(inputMessage, fieldInfo);
}
```

Figure 11. Defining Separate Input Messages for Each Request

Note the following details about Figure 10 on page 26 and Figure 11:

- The CommandCode field is defined within every class at lines A, C, E, and G. This field must be defined in every message that reads the command code. If you do not define the field, you must adjust the offsets of the following fields to account for the existence of the CommandCode in the byte array. For example, you can delete the `DLITypeInfo` entry for CommandCode in the `CancelOrderInput` class, but the OrderNumber field must still start at offset 21.
- The length of the base class `InputMessage` must be large enough to contain any of its subclasses. In this example, the `InputMessage` class is 65 bytes because the fields of the `FindACarInput` method require it B.
- Each `InputMessage` subclass must provide a constructor to create itself from an `InputMessage` object, as in lines D, F, and H. This constructor uses a new constructor in the `IMSFieldMessage` class, called a *copy constructor*.

Given this design, an application can provide message-reading logic similar to that shown in Figure 12.

```
while (getUniqueMessage(inputMessage)) {

    string commandCode=inputMsg.getString("CommandCode").trim();

    if (commandCode.equals("ShowModelDetails")) {
        showModelDetails(new ShowModelDetailsInput(inputMessage));

    } else if(commandCode.equals("FindACar")) {
        findACar(new FindACarInput(inputMessage));

    } else {
        //process an error
    }

}
```

Figure 12. Message-Reading Logic

Developing JBP Applications

JBP applications do not access the IMS message queue, and therefore you do not need to subclass the `IMSFieldMessage` class.

Related Reading: For details about the classes you use to develop a JBP application, see the IMS Java API Specification, which is available on the IMS Java Web site. Go to http://www.ibm.com/ims and link to the IMS Java page.

The following topics provide additional information:

- "Symbolic Checkpoint and Restart"
- "JBP Programming Models" on page 29

Symbolic Checkpoint and Restart

Similarly to BMP applications, JBP applications can use symbolic checkpoint and restart calls to restart the application after an abend. The primary methods for symbolic checkpoint and restart are:

- `IMSTransaction().checkpoint()`
- `IMSTransaction().restart()`

These methods perform functions that are analogous to the DL/I system service calls: (symbolic) `CHKP` and `XRST`.

A JBP application connects to a database, makes a restart call, performs database processing, periodically checkpoints, and disconnects from the database at the end of the program. The program must issue a final commit before ending. On an initial application start, the `IMSTransaction().restart()` method notifies IMS that symbolic checkpoint and restart is to be enabled for the application. The application then issues periodic `IMSTransaction().checkpoint()` calls to take checkpoints. The `IMSTransaction().checkpoint()` method allows the application to provide a `com.ibm.ims.application.SaveArea` object that contains one or more other application Java objects whose state is to be saved with the checkpoint.

If a restart is required, it is initiated in a similar way to BMP applications: the checkpoint ID is provided either with the `IMSTransaction().restart()` call (similarly to providing the ID to the `XRST` call in IMS), or with in the CKPTID= parameter of the

JBP region JCL. The `restart()` method returns a `SaveArea` object that contains the application objects in the same order in which they were originally checkpointed.

Related Reading: For the programming model for symbolic checkpoint and restart, see “JBP Application with Symbolic Checkpoint and Restart.”

JBP Programming Models

JBP applications are similar to JMP applications, except that JBP applications do not receive input messages from the IMS message queue. The program should periodically issue commit calls, except for applications that have the PSB PROCOPT=GO parameter.

Unlike BMP applications, JBP applications must be non-message-driven applications.

JBP Application without Rollback

A JBP application connects to a database, performs database processing, periodically commits, and disconnects from the database at the end of the program. The program must issue a final commit before ending.

```
public static void main(String args[]) {

    conn = DriverManager.getConnection(...);     //Establish DB connection

    repeat {

        repeat {
            results=statement.executeQuery(...); //Perform DB processing
            ...
            MessageQueue.insertMessage(...);     //Send output messages
            ...
        }

        IMSTransaction.getTransaction().commit(); //Periodic commits divide work
    }

    conn.close();                            //Close DB connection
    return;
}
```

JBP Application with Symbolic Checkpoint and Restart

A JBP application connects to a database, makes a restart call, performs database processing, periodically checkpoints, and disconnects from the database at the end of the program. The program must issue a final commit before ending.

```
public static void main(String args[]) {

    conn = DriverManager.getConnection(...);     //Establish DB connection

    IMSTransaction.getTransaction().retart();  //Restart application
                                               //after abend from last
                                               //checkpoint
    repeat {

        repeat {
            results=statement.executeQuery(...); //Perform DB processing
            ...
            MessageQueue.insertMessage(...);     //Send output messages
            ...
        }

        IMSTransaction.getTransaction().checkpoint(); //Periodic checkpoints
                                                      // divide work
```

```
    }

    conn.close();                           //Close DB connection
    return;
}
```

JBP Application using Rollback

Similarly to JMP applications, JBP applications can also roll back database processing and output messages. A final commit call is required before the application can end, even if no further database processing occurs or output messages are sent after the last rollback call.

```
public static void main(String args[]) {

    conn = DriverManager.getConnection(...);     //Establish DB connection

    repeat {

        repeat {

            results=statement.executeQuery(...); //Perform DB processing
            ...
            MessageQueue.insertMessage(...);     //Send output messages
            ...
            IMSTransaction.getTransaction().rollback(); //Roll out DB
                                                         //processing and output
                                                        //messages

            results=statement.executeQuery(...); //Perform more DB
                                                        //processing (optional)
            ...
            MessageQueue.insertMessage(...);     //Send more output
                                                        //messages (optional)
            ...
        }

        IMSTransaction.getTransaction().commit();  //Periodic commits divide work
    }

    conn.close();                           //Close DB connection
    return;
}
```

JBP Application that Accesses DB2 UDB for z/OS or IMS Data

Like a JBP application that accesses IMS data, a JBP application that accesses DB2 UDB for z/OS data connects to a database, performs database processing, periodically commits, and disconnects from the database at the end of the application. However, the application must also issue a final commit after closing the database connection.

The following model is valid for DB2 UDB for z/OS database access, IMS database access, or both DB2 UDB for z/OS and IMS database access.

Related Reading: For more information about accessing DB2 UDB for z/OS data from a JBP application, see “Configuring JMP and JBP Regions for DB2 UDB for z/OS Database Access” on page 16.

```
public void doBegin() ... {                       //Application logic runs
                                                  //doBegin method
   conn = DriverManager.getConnection(...);       //Establish DB connection
        repeat {
           repeat {
                results=statement.executeQuery(...); //Perform DB processing
                ...
                MessageQueue.insertMessage(...);     //Send output messages
```

```
                    ...
                }
                IMSTransaction.getTransaction().commit(); //Periodic commits divide work
            }
            conn.close();                                 //Close DB connection

            IMSTransaction.getTransaction().commit(); //Commit the DB connection close

        return;
    }
```

Enterprise COBOL Interoperability with JMP and JBP Applications

IMS Enterprise COBOL for z/OS and OS/390 Version 3 Release 2 supports interoperation between COBOL and Java languages when running in a JMP or JBP region. With this support, you can:

- Call an object-oriented (OO) COBOL application from an IMS Java application by building the front-end application, which processes messages, in Java, and the back end, which processes databases, in OO COBOL.
- Build an OO COBOL application containing a main routine that can invoke Java routines.

Restriction: COBOL applications that run in an IMS Java dependent region must use the AIB interface, which requires that all PCBs in a PSB definition have a name.

You can access COBOL code in a JMP or JBP region because Enterprise COBOL provides object-oriented language syntax that enables you to:

- Define classes with methods and data implemented in COBOL
- Create instances of Java and COBOL classes
- Invoke methods on Java and COBOL objects
- Write classes that inherit from Java classes or other COBOL classes
- Define and invoke overloaded methods

In Enterprise COBOL programs, you can call the services provided by the JNI to obtain Java-oriented capabilities in addition to the basic OO capabilities available directly in the COBOL language.

In Enterprise COBOL classes, you can code CALL statements that interface with procedural COBOL programs. Therefore, COBOL class definition syntax can be especially useful for writing wrapper classes for procedural COBOL logic, enabling existing COBOL code to be accessed from Java.

Java code can create instances of COBOL classes, invoke methods of these classes, and can extend COBOL classes.

Related Reading: For details building applications that use Enterprise COBOL and that run in an IMS Java dependent region, see *Enterprise COBOL for z/OS and OS/390: Programming Guide*.

The following topics provide additional information:

- “Enterprise COBOL as a Back-End Application in a JMP or JBP Region” on page 32
- “Enterprise COBOL as a Front-End Application in a JMP or JBP Region” on page 32

- "Performance Consideration for OO COBOL in a JMP or JBP Region" on page 33
- "Recommendation against Accessing Databases with Both Java and COBOL" on page 33

Enterprise COBOL as a Back-End Application in a JMP or JBP Region

When you define an OO COBOL class and compile it with the Enterprise COBOL compiler, the compiler generates a Java class definition with native methods and the object code that implements the native methods. After compiling the class, you can create an instance and invoke the methods of the class from a Java program that runs in a JMP or JBP region. For example, you can define an OO COBOL class with the appropriate DL/I call in COBOL to access an IMS database.

When Java is the front-end language, you must perform all message-queue and message-synchronization processing in Java.

For example, you must call both the `IMSMessageQueue.getUniqueMessage` method (to read messages from the queue) and the `IMSTransaction.getTransaction().commit()` method (to commit changes) before reading subsequent messages from the message queue or exiting the application. In the back-end application, you can access IMS databases by either using Java or calling a COBOL routine.

You can use the COBOL STOP RUN statement in the COBOL part of an application that runs in an JMP or JBP region. However, this statement terminates all COBOL and Java routines, including the JVM, and returns control immediately to IMS with both the program and transaction left in a stopped state

Important: Do not mix the languages that are used to read messages from the message queue or to commit resources. The IMS Java library tracks the calls that are made in Java to ensure that the syncpoint rules are followed, but it does not track calls made in COBOL.

For example, you can define an OO COBOL class with the appropriate DL/I call in COBOL to access an IMS database. To make the implementation of this class available to an IMS Java program:

1. Compile the COBOL class with the Enterprise COBOL compiler to generate a Java source file, which contains the class definition, and an object module, which contains the implementation of the native methods.
2. Compile the generated Java source file with the Java compiler to create the application class file.
3. Link the object module into a dynamic link library (DLL) in the HFS file (.so).
4. Update the application class path (ibm.jvm.application.class.path) for the JMP or JBP region to allow access to the Java class file.
5. Update the library path for the JMP or JBP region to allow access to the DLL.

Enterprise COBOL as a Front-End Application in a JMP or JBP Region

The object-oriented syntax of Enterprise COBOL enables you to build COBOL applications with a `main` method, which can be run directly in a JMP or JBP region. The JMP or JBP region locates, instantiates, and invokes this `main` method in the same way it does for the `main` method of a Java application.

You can write an application for an JMP or JBP region entirely with OO COBOL, but a more likely use for a front-end COBOL application is to call a Java routine from a COBOL application.

When running within the JVM of an JMP or JBP region, Enterprise COBOL run-time support automatically locates and uses this JVM to invoke methods on Java classes.

A front-end OO COBOL application with a main routine that runs in a JMP or JBP region has the same requirements as a Java program that runs in a JMP or JBP region.

The COBOL application must commit resources before reading subsequent messages or exiting the application. A COBOL GU call does not implicitly commit resources when the program is running in a JMP or JBP region as it does when the program is running in an MPP region.

Use DI/I calls for message processing (`GU` and `GN`) and transaction synchronization (`CHKP`). A `CHKP` call in a JMP or JBP region does not automatically retrieve a message from the message queue.

You can use the COBOL `STOP RUN` statement in the COBOL part of an application that runs in a JMP or JBP region. However, this statement terminates all COBOL and Java routines, including the JVM, and returns control immediately to IMS with both the program and transaction left in a stopped state.

Performance Consideration for OO COBOL in a JMP or JBP Region

COBOL code in a JMP or JBP dependent region affects performance. Because COBOL class methods are implemented in native code, the JVM cannot be reset after a transaction that uses COBOL routines runs.

IBM's Persistent Reusable Java Virtual Machine is specifically designed to treat applications that invoke native code as untrusted. After a transaction runs that contains COBOL routines, IMS ends the current JVM and creates a fresh JVM before scheduling the next transaction. Only classes in the trusted middleware class path ibm.jvm.middleware.class.path can call native routines without affecting JVM reset.

Related Reading: For more information about the Persistent Reusable Java Virtual Machine, see *IBM Developer Kit for OS/390, Java 2 Technology Edition: New IBM Technology featuring Persistent Reusable Java Virtual Machines*.

Recommendation against Accessing Databases with Both Java and COBOL

IBM recommends that you do not access the same DB PCB from both Java and COBOL. The COBOL and Java parts of an application share a single database pointer (or cursor). If the same DB PCB is accessed by both Java and COBOL, database positioning as a result of calls in one language affect the database positioning for calls in the other language.

For example, if you build a SQL SELECT clause and use JDBC to query and retrieve results, the IMS Java class library constructs the appropriate request to IMS to establish the correct position in the database. If you then call a COBOL routine, which builds an SSA and runs a GU request to IMS against the same DB PCB, the GU request will likely change the position in the database for that DB PCB. If the

position is changed, subsequent JDBC requests using the same SQL SELECT clause to retrieve more records will be wrong because the database position has changed.

If you must access the same DB PCB from multiple languages, establish database positioning again when returning from an inter-language call before accessing more records in the database.

Note: Although IBM advises caution for language interoperability, the behavior described in this section is not related to the programming languages themselves. Two parts of the same application that both access the same DB PCB can have the same behavior described in this section even if both parts are written in the same language.

Accessing DB2 UDB for z/OS Databases from JMP or JBP Applications

A JMP or JBP application can access DB2 UDB for z/OS databases by using the DB2 JDBC/SQLJ 2.0 driver or the DB2 JDBC/SQLJ 1.2 driver. The JMP or JBP region that the application is running in must also be defined with DB2 UDB for z/OS attached by the DB2 Recoverable Resource Manager Services attachment facility (RRSAF).

Related Reading: For information about attaching DB2 UDB for z/OS to IMS for JMP or JBP application access to DB2 UDB for z/OS databases, see “Configuring JMP and JBP Regions for DB2 UDB for z/OS Database Access” on page 16.

Accessing DB2 UDB for z/OS data from a JMP or JBP application is similar to accessing IMS data. When writing a JMP or JBP application that accesses DB2 UDB for z/OS data, consider both the differences from IMS database access and the differences from accessing DB2 UDB for z/OS data in other environments:

- You can have only one active DB2 UDB for z/OS connection open at any time.
- For type 2 JDBC drivers, you must use the default connection URL in the application program. For example, `jdbc:db2os390:` or `db2:default:connection`.
- For type 4 JDBC drivers, you can use a specific connection URL in the application program.
- To commit or roll back work, you must use the `IMSTransaction.getTransaction().commit()` method or the `IMSTransaction.getTransaction().rollback()` method. For JMP applications, the`IMSTransaction.getTransaction().commit()` method commits all work: SQL calls and connection closures. For JBP applications, the `IMSTransaction.getTransaction().commit()` method commits SQL calls.
- Because RRS is the coordinator, you cannot use the `Connection.setAutoCommit` or `Connection.commit` method of the DB2 JDBC driver.
- You must always call `IMSTransaction.getTransaction().commit()` after closing a connection to DB2 UDB for z/OS to commit the connection closure.
- You cannot use COBOL to access DB2 UDB for z/OS in a JMP or JBP region.

Related Reading: For a JMP programming model, see “JMP Application that Accesses IMS or DB2 UDB for z/OS Data” on page 21. For a JBP programming model, see “JBP Application that Accesses DB2 UDB for z/OS or IMS Data” on page 30.

Program Switching in JMP and JBP Applications

IMS Java provides an API for immediate program switching in JMP and JBP applications and for deferred program switching in conversational JMP applications.

For more information about program switches, see the *IMS Version 9: Application Programming: Design Guide*.

Immediate Program Switching for JMP and JBP Applications

The `setModifiableAlternatePCB(String)` method of the `com.ibm.ims.application.IMSMessageQueue` class sets the name of the alternate PCB for the program switch. The `setModifiableAlternatePCB(String)` method calls the DL/I `CHNG` call.

To make a program switch in a JMP or JBP application:

1. Call the `setModifiableAlternatePCB(String)` method to set the name of the alternate PCB.
2. Call the `insertMessage(IMSFieldMessage)` method to send the message to the alternate PCB.

For more information about these methods, see the *IMS Java API Specification*.

Deferred Program Switching for Conversational JMP Applications

You can make a deferred program switch in a conversational JMP application. A deferred program switch changes the transaction code in the SPA before the SPA is returned to IMS. When an application makes a deferred program switch, the application replies to the terminal and passes the conversation to another conversational application.

The `setTransactionID(String)` method of the `com.ibm.ims.application.IMSFieldMessage` class specifies the transaction code in the SPA.

To make a deferred program switch:

1. Call the `insertMessage(IMSFieldMessage)` method to send the output message to the terminal.
2. Call the `setTransactionID(String)` method to set the name of the transaction code in the SPA.
3. Call the `insertMessage(IMSFieldMessage)` method to send the SPA to IMS.

For more information about these methods, see the *IMS Java API Specification*.

Chapter 3. WebSphere Application Server for z/OS Applications

You can write applications that run on WebSphere Application Server for z/OS and access IMS databases when WebSphere Application Server for z/OS and IMS are on the same LPAR (logical partition).

To deploy an application on WebSphere Application Server for z/OS, you must install the IMS JDBC resource adaptor (the IMS Java class libraries) on WebSphere Application Server for z/OS, and configure both IMS open database access (ODBA) and the database resource adapter (DRA).

Figure 13 shows an Enterprise JavaBean (EJB) that is accessing IMS data. JDBC or IMS Java hierarchical interface calls are passed to the IMS Java layer, which converts the calls to DL/I calls. The IMS Java layer passes these calls to ODBA, which uses the DRA to access the DL/I region in IMS.

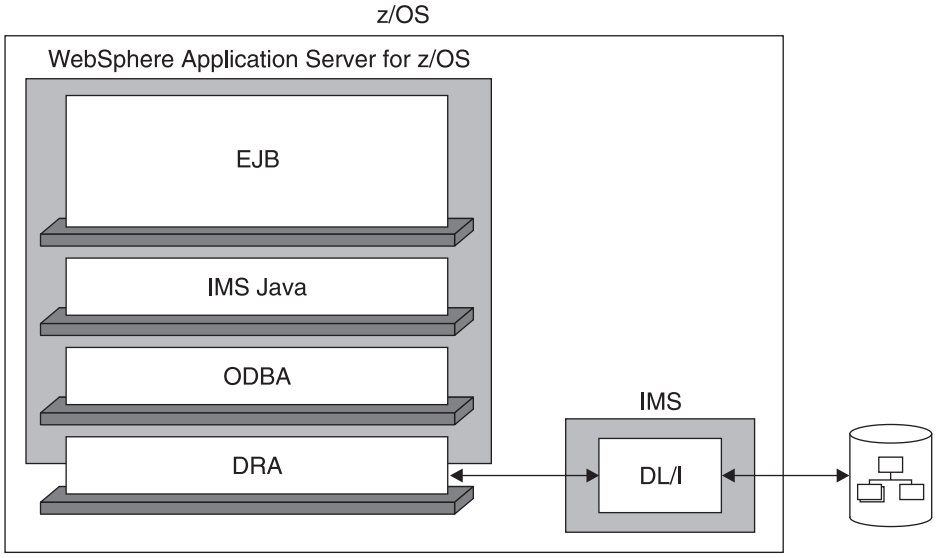


Figure 13. WebSphere Application Server for z/OS EJB Using IMS Java

The following topics provide additional information:

- “Configuring WebSphere Application Server for z/OS for IMS Java” on page 38
- “Running the IMS Java IVP on WebSphere Application Server for z/OS” on page 44
- “Running the IMS Java Sample Applications on WebSphere Application Server for z/OS” on page 51
- “Running Your Applications on WebSphere Application Server for z/OS” on page 59
- “Developing Enterprise Applications that Access IMS DB” on page 69

Configuring WebSphere Application Server for z/OS for IMS Java

To use IMS Java with WebSphere Application Server for z/OS, you must use WebSphere Application Server V5.0 for z/OS or later. If you have WebSphere Application Server V5.0.2 for z/OS, you must install either V5.0.2.1 or APAR PQ81944. You must also use RRS (resource recovery services) for z/OS.

To access IMS databases from WebSphere Application Server on a non-z/OS platform, you must have WebSphere Application Server V5.0 for z/OS installed on the same logical partition (LPAR) as IMS. You must configure WebSphere Application Server for z/OS as well as WebSphere Application Server on the non-z/OS platform. For information about setting up both of these servers, see Chapter 4, “Remote Data Access with WebSphere Application Server Applications,” on page 75.

Before you can deploy an application on WebSphere Application Server for z/OS, you must configure the server. This topic provides the basic steps required to configure the server.

To configure WebSphere Application Server for z/OS:

1. Update the JCL for WebSphere Application Server for z/OS by adding to the STEPLIB the following data sets:
   - The load library that contains the DRA startup table and the ODBA run-time code
   - The SDFSJLIB data set. This data set contains the DFSCLIB member.
2. If you are using WebSphere Application Server V5 for z/OS, add the following required XML files to the server class path:

   ```
   pathprefix/usr/lpp/ims/imsjava91/lib/xml-apis.jar
   pathprefix/usr/lpp/ims/imsjava91/lib/xalan.jar
   pathprefix/usr/lpp/ims/imsjava91/lib/xercesImpl.jar
   ```

3. Install the IMS JDBC resource adapter. The file name of the resource adapter is *pathprefix*/usr/lpp/ims/imsjava91/imsjava91.rar and the file name of the IMS Java library is *pathprefix*/usr/lpp/ims/imsjava91/imsjava.jar.
4. Modify the WebSphere Application Server for z/OS server.policy file, which is in the properties directory of the WebSphere Application Server installation directory, by adding the following code:

   ```
   grant codeBase "file:/pathprefix/usr/lpp/ims/imsjava91/-" {
     //Allows the IMS JDBC resource adapter and the custom service to read and
     //write environment properties    .
     permission java.util.PropertyPermission "*", "read, write";

     //Allows the IMS JDCB resource adapter and the custom service to use
     //the JavTDLI load library during runtime.
         permission java.lang.RuntimePermission "loadLibrary.JavTDLI";
   };
   ```

5. Install the custom service. The class name is com.ibm.connector2.ims.db.IMSJdbcCustomService and the class path is *pathprefix*/usr/lpp/ims/imsjava91.

Detailed steps, specific to the version of WebSphere Application Server for z/OS are also provided:

- “Configuring WebSphere Application Server V5 for z/OS” on page 39
- “Configuring WebSphere Application Server V6 for z/OS” on page 42

Configuring WebSphere Application Server V5 for z/OS

Prerequisite: “Installing IMS Java” on page 2

To configure WebSphere Application Server V5 for z/OS:

1. “Configuring WebSphere Application Server V5 for z/OS to Access IMS”
2. “Adding the Required XML Files to the WebSphere Application Server V5 for z/OS Classpath”
3. “Installing the IMS JDBC Resource Adapter on WebSphere Application Server V5 for z/OS” on page 40
4. “Installing the Custom Service on WebSphere Application Server V5 for z/OS” on page 41

Next: “Running the IMS Java IVP on WebSphere Application Server for z/OS” on page 44

Configuring WebSphere Application Server V5 for z/OS to Access IMS

To use JDBC to access IMS DB from WebSphere Application Server for z/OS, you first must configure WebSphere Application Server for z/OS to access IMS databases using ODBA. ODBA uses the database resource adapter (DRA) to access IMS databases.

Related Reading: For details about the steps in this section, see the ODBA section of *IMS Version 9: Installation Volume 2: System Definition and Tailoring*.

To configure WebSphere Application Server for z/OS to access IMS databases:

1. If not already done, create a DRA startup table. The DRA startup table module name must have the following naming convention:
   - Bytes 1-3: “DFS”
   - Bytes 4-7: 1- to 4-byte ID
   - Byte 8: “0”

   Recommendation: The 1- to 4-byte ID should be the IMS system ID.
2. If not already done, link the DRA startup table into a load library.
3. Update the JCL for WebSphere Application Server for z/OS by adding to the STEPLIB the following data sets:
   - The load library that contains the DRA startup table and the ODBA run-time code
   - The SDFSJLIB data set. This data set contains the DFSCLIB member.
4. Note the DRA name, which is defined by the MBR parameter. You will need to know bytes 4-7, which are usually the IMS system ID, when you install the data source.

Next: “Adding the Required XML Files to the WebSphere Application Server V5 for z/OS Classpath”

Adding the Required XML Files to the WebSphere Application Server V5 for z/OS Classpath

Applications that use the `storeXML()` and `retrieveXML()` UDFs need the XML parser files that are described in “Downloading Apache Open Source XML Libraries” on page 3

on page 3. The IVP checks that you have the correct versions of the XML parser files and that they are in the classpath. If you do not do this task, you will receive an error when you run the IVP that you can choose to ignore if you know that you do not use the XML functions of IMS Java.

Prerequisite: "Downloading Apache Open Source XML Libraries" on page 3

To add the required XML files to the WebSphere Application Server for z/OS classpath:

1. From the WebSphere Application Server for z/OS administrative console, click **Environment**, and then click **Shared Libraries**.
2. Click **New**.
3. In the **Name** field of the Configuration section, type a name for the shared library. For example, type: `XML Shared Library`
4. In the **Classpath** field, type the path, including the file names, to the required jar files. If you have installed SDK 1.4.2 or later, type the path to the file xml.jar in the SDK 1.4.2 lib directory. If you downloaded the required XML files as described in "Downloading Apache Open Source XML Libraries" on page 3, type:

   pathprefix`/usr/lpp/ims/imsjava91/lib/xml-apis.jar`
   pathprefix`/usr/lpp/ims/imsjava91/lib/xalan.jar`
   pathprefix`/usr/lpp/ims/imsjava91/lib/xercesImpl.jar`
5. Click **OK**.
6. Click **Save**.

   The Save page is displayed.
7. Under **Save to Master Configuration**, click **Save**.

Next: "Installing the IMS JDBC Resource Adapter on WebSphere Application Server V5 for z/OS"

Installing the IMS JDBC Resource Adapter on WebSphere Application Server V5 for z/OS

After you configure WebSphere Application Server for z/OS to have access to IMS databases, you must install the IMS JDBC resource adapter on WebSphere Application Server for z/OS.

Prerequisite: "Configuring WebSphere Application Server V5 for z/OS to Access IMS" on page 39

To install the IMS JDBC resource adapter:

1. From the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click **Install RAR**.

   A dialog for installing the resource adapter is displayed.
3. Select **Server path** and type the path to the imsjava91.rar file:
   pathprefix`/usr/lpp/ims/imsjava91/imsjava91.rar`
4. Click **Next**.

   A configuration dialog is displayed.
5. Type the following information:
   Name: a name for the resource adapter

Classpath: the path to imsjava.jar, including the file name:
pathprefix/usr/lpp/ims/imsjava91/imsjava.jar

6. Click **OK**.

   The IMS JDBC resource adapter is listed.
7. Click **Save**.

   The Save page is displayed.
8. Under **Save to Master Configuration**, click **Save** to ensure that the changes have been made.

Next: "Installing the Custom Service on WebSphere Application Server V5 for z/OS"

Installing the Custom Service on WebSphere Application Server V5 for z/OS

Prerequisite: "Installing the IMS JDBC Resource Adapter on WebSphere Application Server V5 for z/OS" on page 40

When WebSphere Application Server for z/OS is started, the custom service initializes the ODBA environment. When the server is stopped, the custom service terminates the ODBA environment. After a server is started, every application that is running in the server uses the initialized ODBA environment.

To install the custom service:

1. Modify the WebSphere Application Server for z/OS server.policy file, which is in the properties directory of the WebSphere Application Server installation directory, by adding the following code:

```
grant codeBase "file:/pathprefix/usr/lpp/ims/imsjava91/-" {
  //Allows the IMS JDBC resource adapter and the custom service to read and
  //write environment properties    .
  permission java.util.PropertyPermission "*", "read, write";

  //Allows the IMS JDCB resource adapter and the custom service to use
  //the JavTDLI load library during runtime.
      permission java.lang.RuntimePermission "loadLibrary.JavTDLI";
};
```

2. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Servers**, and then click **Application Servers**.

   A list of application servers is displayed.
3. Click the name of the server on which you want to deploy your custom service.
4. Under Additional Properties, click **Custom Services**.

   A list of custom services is displayed.
5. Click **New**.

   A configuration dialog is displayed.
6. Select the **Startup** check box.

   If you do not select the **Startup** check box, the custom service is not invoked when you start the server.
7. Type the following information:
   Classname: com.ibm.connector2.ims.db.IMSJdbcCustomService
   Display Name: a name for the custom service
   Classpath: the path to the directory that contains imsjava.jar and libJavTDLI.so: *pathprefix*/usr/lpp/ims/imsjava91
8. Click **OK**.

The custom service is listed.

9. Click **Save**.

   The Save page is displayed.

10. Under **Save to Master Configuration**, click **Save** to ensure that the changes have been made.
11. Restart the server in order for the custom service to take effect.

Next: "Running the IMS Java IVP on WebSphere Application Server for z/OS" on page 44

Configuring WebSphere Application Server V6 for z/OS

This section assumes that you are familiar with WebSphere Application Server V6 for z/OS and its administrative console.

Prerequisite: "Installing IMS Java" on page 2

To configure WebSphere Application Server V6 for z/OS:

1. "Configuring WebSphere Application Server V6 for z/OS to Access IMS"
2. "Adding the Required XML Files to the WebSphere Application Server V5 for z/OS Classpath" on page 39
3. "Installing the IMS JDBC Resource Adapter on WebSphere Application Server V6 for z/OS" on page 43
4. "Installing the Custom Service on WebSphere Application Server V6 for z/OS" on page 43

Configuring WebSphere Application Server V6 for z/OS to Access IMS

To use JDBC to access IMS DB from WebSphere Application Server for z/OS, you first must configure WebSphere Application Server for z/OS to access IMS databases using ODBA. ODBA uses the database resource adapter (DRA) to access IMS databases.

Related Reading: For details about the steps in this section, see the ODBA section of *IMS Version 9: Installation Volume 2: System Definition and Tailoring*.

To configure WebSphere Application Server for z/OS to access IMS databases:

1. If not already done, create a DRA startup table. The DRA startup table module name must have the following naming convention:
   - Bytes 1-3: "DFS"
   - Bytes 4-7: 1- to 4-byte ID
   - Byte 8: "0"

   Recommendation: The 1- to 4-byte ID should be the IMS system ID.
2. If not already done, link the DRA startup table into a load library.
3. Update the JCL for WebSphere Application Server for z/OS by adding to the STEPLIB the following data sets:
   - The load library that contains the DRA startup table and the ODBA run-time code
   - The SDFSJLIB data set. This data set contains the DFSCLIB member.

4. Note the DRA name, which is defined by the MBR parameter. You will need to know bytes 4-7, which are usually the IMS system ID, when you install the data source.

Next: "Installing the IMS JDBC Resource Adapter on WebSphere Application Server V6 for z/OS"

Installing the IMS JDBC Resource Adapter on WebSphere Application Server V6 for z/OS

After you configure WebSphere Application Server for z/OS to have access to IMS databases, you must install the IMS JDBC resource adapter on WebSphere Application Server for z/OS.

Prerequisite: "Configuring WebSphere Application Server V6 for z/OS to Access IMS" on page 42

To install the IMS JDBC resource adapter:

1. From the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click **Install RAR**.

   A dialog for installing the resource adapter is displayed.
3. Select **Server path** and type the path to the imsjava91.rar file:
   pathprefix`/usr/lpp/ims/imsjava91/imsjava91.rar`
4. Click **Next**.

   A configuration dialog is displayed.
5. Type the following information:
   Name: a name for the resource adapter
   Classpath: the path to imsjava.jar, including the file name:
   pathprefix`/usr/lpp/ims/imsjava91/imsjava.jar`
6. Click **OK**.

   The IMS JDBC resource adapter is listed.
7. In the messages box, click **Save**.

   The save page is displayed.
8. Click **Save** to update the master repository with your changes.

Next: "Installing the Custom Service on WebSphere Application Server V6 for z/OS"

Installing the Custom Service on WebSphere Application Server V6 for z/OS

Prerequisite: "Installing the IMS JDBC Resource Adapter on WebSphere Application Server V6 for z/OS"

When WebSphere Application Server for z/OS is started, the custom service initializes the ODBA environment. When the server is stopped, the custom service terminates the ODBA environment. After a server is started, every application that is running in the server uses the initialized ODBA environment.

To install the custom service:

1. Modify the WebSphere Application Server for z/OS server.policy file, which is in the properties directory (for example, *installdir*/profiles/default/properties), by adding the following code:

```
grant codeBase "file:/pathprefix/usr/lpp/ims/imsjava91/-" {
  //Allows the IMS JDBC resource adapter and the custom service to read and
  //write environment properties    .
  permission java.util.PropertyPermission "*", "read, write";

  //Allows the IMS JDCB resource adapter and the custom service to use
  //the JavTDLI load library during runtime.
      permission java.lang.RuntimePermission "loadLibrary.JavTDLI";
};
```

2. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Servers**, and then click **Application servers**.

   A list of application servers is displayed.
3. Click the name of the server on which you want to deploy your custom service.
4. Under Server Infrastructure, click **Administration**, and then click **Custom Services**.

   A list of custom services is displayed.
5. Click **New**.

   A configuration dialog is displayed.
6. Select the **Enable service at server startup** check box.

   If you do not select this check box, the custom service is not invoked when you start the server.
7. Type the following information:
   Classname: `com.ibm.connector2.ims.db.IMSJdbcCustomService`
   Display Name: a name for the custom service
   Classpath: the path to the directory that contains imsjava.jar and libJavTDLI.so: *pathprefix*`/usr/lpp/ims/imsjava91`
8. Click **OK**.

   The custom service is listed.
9. In the messages box, click **Save**.

   The save page is displayed.
10. Click **Save** to update the master repository with your changes.
11. Restart the server in order for the custom service to take effect.

Next: "Running the IMS Java IVP on WebSphere Application Server V6 for z/OS" on page 48

Running the IMS Java IVP on WebSphere Application Server for z/OS

Run the IMS Java IVP on WebSphere Application Server for z/OS to ensure that you have configured IMS and WebSphere Application Server for z/OS properly. This topic provides the high-level tasks required to run the IMS Java IVP.

To run the IMS Java IVP on WebSphere Application Server for z/OS:

1. Install the data source of the IMS Java IVP as a new J2C connection factory. The JNDI name is imsjavaIVP. The database view name is samples.ivp.DFSIVP37DatabaseView.
2. Install the IMS Java IVP EAR file. The file is *pathprefix*/usr/lpp/ims/imsjava91/samples/ivp/was/imsjavaIVP.ear.

3. If you are using WebSphere Application Server V5 for z/OS, add the required XML files to the IVP application class path.
4. Test the IVP. The URL is http://*host_IP_address:port*/IMSJavaIVPWeb/IMSJavaIVP.html.

Detailed information about running the IMS Java IVP on specific versions of WebSphere Application Server for z/OS is available:

- "Running the IMS Java IVP on WebSphere Application Server V5 for z/OS"
- "Running the IMS Java IVP on WebSphere Application Server V6 for z/OS" on page 48

Running the IMS Java IVP on WebSphere Application Server V5 for z/OS

Prerequisites:

- "Configuring WebSphere Application Server for z/OS for IMS Java" on page 38
- Ensure that the standard IMS IVPs have been run. The IMS IVPs prepare the DBD for the IVP database, named IVPDB2, and load the IVP database. They also prepare the IMS Java application PSB (named DFSIVP37), build ACBs, and prepare other IMS control blocks that are required by the IMS Java IVPs. For details about how to run the IMS IVPs, see *IMS Version 9: Installation Volume 1: Installation Verification*.

To run the IMS Java IVP for WebSphere Application Server for z/OS:

1. "Installing the Data Source for the IMS Java IVP on WebSphere Application Server V5 for z/OS"
2. "Installing the IMS Java IVP on WebSphere Application Server V5 for z/OS" on page 46
3. "Adding the XML Files to the IVP Classpath on WebSphere Application Server V5 for z/OS" on page 47
4. "Testing the IMS Java IVP on WebSphere Application Server V5 for z/OS" on page 48

Installing the Data Source for the IMS Java IVP on WebSphere Application Server V5 for z/OS

The `DataSource` facility is a factory for connections to a physical data source, or database. A data source is registered with a naming service based on the Java Naming and Directory (JNDI) API. `DataSource` objects have properties that pertain to the actual data source that an application needs to access.

Requirement: You must use the `DataSource` facility, which replaces the `DriverManager` facility, because the `DriverManager` facility is not supported by the J2EE Connection Architecture Specification.

To install the data source for the IVP:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.

2. Click the name of IMS JDBC resource adapter that you chose when you installed the adapter.

   A configuration dialog is displayed.
3. Under Additional Properties, click **J2C Connection Factories**.
4. Click **New**.

   A configuration dialog is displayed.
5. Type the following information:
   Name: name for the data source
   JNDI Name: `imsjavaIVP`
6. Click **OK**.

   The data source is listed in the J2C Connection Factories.
7. Click the name of the data source that you installed in step 5.
8. Under Additional Properties, click **Custom Properties**.

   Six properties are listed in a table.
9. In the **DRAName** row, click the dash symbol in the **Value** column.
10. In the **Value** field, type bytes 4-7 of the DRA startup table module name (usually the IMS system ID). For more information about the DRA startup table, see "Configuring WebSphere Application Server V6 for z/OS to Access IMS" on page 42.
11. Click **OK**.

    The properties table displays the DRA name that you just entered.
12. In the **DatabaseViewName** row, click the dash symbol in the **Value** column.
13. In the **Value** field, type `samples.ivp.DFSIVP37DatabaseView`
14. Click **OK**.

    The properties table displays the host name that you just entered.
15. Click **Save**.

    The Save page is displayed.
16. Under **Save to Master Configuration**, click **Save**.
17. Restart the server to ensure that the changes have been made.

Next: "Installing the IMS Java IVP on WebSphere Application Server V6 for z/OS" on page 50

Installing the IMS Java IVP on WebSphere Application Server V5 for z/OS

Prerequisite: "Installing the Data Source for the IMS Java IVP on WebSphere Application Server V6 for z/OS" on page 49

This section describes how to deploy the IMS Java IVP on WebSphere Application Server for z/OS.

To install the IMS Java IVP:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Install New Application**.

   A dialog for installing the application is displayed.
2. Select **Server path** and type the path to IMSJavaIVP.ear:
   pathprefix`/usr/lpp/ims/imsjava91/samples/ivp/was/imsjavaIVP.ear`
3. Click **Next**.
4. Accept the defaults and click **Next**.

The Install New Application wizard is started. Step 1, ″Provide options to perform the installation,″ is displayed.

5. Clear the **Create MBeans for Resources** check box.
6. Click **Next**.

   Step 2, ″Provide JNDI Names for Beans,″ is displayed.
7. In the **JNDI Name** field, verify that the name is as follows: `ejb/samples/ivp/was/IMSJavaIVPSessionHome`
8. Click **Next**.

   Step 3, ″Map resource references to resources,″ is displayed.
9. In the JNDI Name field, verify that the name is as follows: `imsjavaIVP`
10. Click **Next**.

    Step 4, ″Map virtual hosts for web modules,″ is displayed.
11. Accept the defaults and click **Next**.

    Step 5, ″Map modules to application servers,″ is displayed.
12. Accept the defaults and click **Next**.

    Step 6, ″Ensure all unprotected 2.0 methods have the correct level of protection,″ is displayed.
13. Make any necessary changes and click **Next**.

    The options that you specified are displayed in Step 7, ″Summary,″ of the Install New Application wizard.
14. Verify that the options are correct, and then click **Finish**.

    A message is displayed that indicates first that the application is being installed, and then that the installation was successful.
15. Click **Save to Master Configuration**.

    The Save page is displayed.
16. Under **Save to Master Configuration**, click **Save**.

Next: “Adding the XML Files to the IVP Classpath on WebSphere Application Server V5 for z/OS”

Adding the XML Files to the IVP Classpath on WebSphere Application Server V5 for z/OS

The IVP tests that the required XML files are installed and in the WebSphere classpath. For the IVP to find these files, you must add these files to the application classpath. If you do no not do this task, you will receive an error when you run the IVP.

Prerequisite: “Installing the IMS Java IVP on WebSphere Application Server V6 for z/OS” on page 50

To add the XML files to the application classpath:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Enterprise Applications**.

   The application IMSJava IVP is listed.
2. Click **IMSJava IVP**.
3. Under General Properties, in the **Classloader Mode** field, select PARENT_LAST.
4. Click **Apply**.
5. Under Additional Properties, click **Libraries**.

6. Click **Add**.
7. In the Library **Name** field, select the shared library that you created in "Adding the Required XML Files to the WebSphere Application Server V5 for z/OS Classpath" on page 39. For example, select XML Shared Library.
8. Click **OK**.
9. Click **Save**.

   The Save page is displayed.
10. Under **Save to Master Configuration**, click **Save**.

Next: "Testing the IMS Java IVP on WebSphere Application Server V5 for z/OS"

Testing the IMS Java IVP on WebSphere Application Server V5 for z/OS

Prerequisite: "Adding the XML Files to the IVP Classpath on WebSphere Application Server V5 for z/OS" on page 47

This topic describes how to test the IVP on WebSphere Application Server for z/OS.

To test the IMS Java IVP:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Enterprise Applications**.

   The application IMSJava IVP is listed with a red X, which indicates that the application is stopped.
2. Select **IMSJava IVP**.
3. Click **Start**.

   The application IMSJava IVP is listed with a green arrow, which indicates that the application is started.
4. Open a Web browser.
5. Type the Web address:

   `http://host_IP_address:port/IMSJavaIVPWeb/IMSJavaIVP.html`

   An input Web page opens.
6. Click **Run the IVP**.

   If WebSphere Application Server for z/OS is configured properly, the IVP displays ″The IVP was SUCCESSFUL″ and the results of checks performed by the IVP.

   If WebSphere Application Server for z/OS is not configured properly, the IVP displays ″The IVP was NOT SUCCESSFUL″ and the results of checks performed by the IVP.

Running the IMS Java IVP on WebSphere Application Server V6 for z/OS

Prerequisites:

- "Configuring WebSphere Application Server V6 for z/OS" on page 42
- Ensure that the standard IMS IVPs have been run. The IMS IVPs prepare the DBD for the IVP database, named IVPDB2, and load the IVP database. They also prepare the IMS Java application PSB (named DFSIVP37), build ACBs, and prepare other IMS

control blocks that are required by the IMS Java IVPs. For details about how to run the IMS IVPs, see *IMS Version 9: Installation Volume 1: Installation Verification*.

To run the IMS Java IVP for WebSphere Application Server for z/OS:

1. "Installing the Data Source for the IMS Java IVP on WebSphere Application Server V6 for z/OS"
2. "Installing the IMS Java IVP on WebSphere Application Server V6 for z/OS" on page 50
3. "Testing the IMS Java IVP on WebSphere Application Server V6 for z/OS" on page 50

Installing the Data Source for the IMS Java IVP on WebSphere Application Server V6 for z/OS

The `DataSource` facility is a factory for connections to a physical data source, or database. A data source is registered with a naming service based on the Java Naming and Directory (JNDI) API. `DataSource` objects have properties that pertain to the actual data source that an application needs to access.

Requirement: You must use the `DataSource` facility, which replaces the `DriverManager` facility, because the `DriverManager` facility is not supported by the J2EE Connection Architecture Specification.

To install the data source for the IVP:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the name of IMS JDBC resource adapter that you chose when you installed the adapter.

   A configuration dialog is displayed.
3. Under Additional Properties, click **J2C connection factories**.
4. Click **New**.

   A configuration dialog is displayed.
5. Type the following information:
   Name: name for the data source
   JNDI Name: `imsjavaIVP`
6. Click **OK**.

   The data source is listed in the J2C Connection Factories.
7. Click the name of the data source that you installed in step 5.
8. Under Additional Properties, click **Custom properties**.

   Three properties are listed in a table.
9. Click **DatabaseViewName**.

   A configuration dialog is displayed.
10. In the **Value** field, type: `samples.ivp.DFSIVP37DatabaseView`
11. Click **OK**.

    The properties table displays the DatabaseViewName value that you just entered.
12. Click **DRAName**.

    A configuration dialog is displayed.

13. In the **Value** field, type bytes 4-7 of the DRA startup table module name (usually the IMS system ID). For more information about the DRA startup table, see “Configuring WebSphere Application Server V6 for z/OS to Access IMS” on page 42.
14. Click **OK**.

    The properties table displays the DRA name that you just entered.
15. In the messages box, click **Save**.

    The save page is displayed.
16. Click **Save** to update the master repository with your changes.
17. Restart the server in order for the custom service to take effect.

Next: “Installing the IMS Java IVP on WebSphere Application Server V6 for z/OS”

Installing the IMS Java IVP on WebSphere Application Server V6 for z/OS

Prerequisite: “Installing the Data Source for the IMS Java IVP on WebSphere Application Server V6 for z/OS” on page 49

This section describes how to deploy the IMS Java IVP on WebSphere Application Server V6 for z/OS.

To install the IMS Java IVP:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Install New Application**.

   A dialog for installing the application is displayed.
2. Select **Remote file system** and type the path to IMSJavaIVP.ear: *pathprefix*`/usr/lpp/ims/imsjava91/samples/ivp/was/imsjavaIVP.ear`
3. Click **Next**.
4. Accept the defaults and click **Next**.

   The Install New Application wizard is started.
5. Click **Step 7** to accept the installation defaults. Depending on your specific server configuration, you might have to use the wizard to change some default values.
6. Verify that the options are correct, and then click **Finish**.

   A message is displayed that indicates first that the application is being installed, and then that the installation was successful.
7. Click **Save to Master Configuration**.

   The save page is displayed.
8. Click **Save** to update the master repository with your changes.

Next: “Testing the IMS Java IVP on WebSphere Application Server V6 for z/OS”

Testing the IMS Java IVP on WebSphere Application Server V6 for z/OS

Prerequisite: “Installing the IMS Java IVP on WebSphere Application Server V6 for z/OS”

This topic describes how to test the IVP on WebSphere Application Server V6 for z/OS.

To test the IMS Java IVP:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Enterprise Applications**.

   The application IMSJava IVP is listed with a red X, which indicates that the application is stopped.
2. Select **IMSJava IVP**.
3. Click **Start**.

   The application IMSJava IVP is listed with a green arrow, which indicates that the application is started.
4. Open a Web browser.
5. Type the Web address:

   `http://host_IP_address:port/IMSJavaIVPWeb/IMSJavaIVP.html`

   An input Web page opens.
6. Click **Run the IVP**.

   If WebSphere Application Server for z/OS is configured properly, the IVP displays ″The IVP was SUCCESSFUL″ and the results of checks performed by the IVP.

   If WebSphere Application Server for z/OS is not configured properly, the IVP displays ″The IVP was NOT SUCCESSFUL″ and the results of checks performed by the IVP.

Running the IMS Java Sample Applications on WebSphere Application Server for z/OS

IMS provides two sample Java applications for WebSphere Application Server for z/OS. The phonebook sample application uses the same database as the IVP application, but allows different queries against the database. The dealership sample application queries a sample dealership database.

You do not need to run either sample application to verify the installation of IMS Java for WebSphere Application Server for z/OS. This sample application is provided to show more complex queries and a more complex database than the IVP application.

This topic describes the high-level tasks that you must complete, along with the IMS-specific information required, to run the IMS Java sample applications.

To run the IMS Java sample application on WebSphere Application Server for z/OS:

1. Install the data source for the IMS Java sample application as a J2C connection factory.

   JNDI name:

   - Phonebook sample: imsjavaPhonebook
   - Dealership sample: jdbc/DealershipSample

   Database view name:

   - Phonebook sample: `samples.ivp.DFSIVP37DatabaseView`
   - Dealership sample: `samples.dealership.AUTPSB11DatabaseView`
2. Install and start the IMS Java sample application EAR file.

   EAR file path:

   - Phonebook sample:

pathprefix/usr/lpp/ims/imsjava91/samples/
ivp/was/IMSJavaPhonebook.ear

- Dealership sample:

  pathprefix/usr/lpp/ims/imsjava91/samples/
  dealership/was/imsjavaDealership.ear

EJB home interface:

- Phonebook sample:

  ejb/samples/phonebook/was/IMSJavaPhonebookSessionHome

- Dealership sample:

  samples.dealership.was.DealershipSessionHome

3. Test the sample application.

   Sample application URL:

   - Phonebook sample:

     http://*host_IP_address*:*port*/IMSJavaPhonebookWeb/IMSJavaPhonebook.html

   - Dealership sample:

     http://*host_IP_address*:*port*/IMSDealershipWeb/dealership.html

   Sample input data:

   - Phonebook sample:

     Last Name: LAST1

   - Dealership sample:

     Car Make: FORD

     VIN Number: V234567890123456789V

Detailed information about running the IMS Java sample applications on specific versions of WebSphere Application Server for z/OS is available:

- "Running the IMS Java Sample Applications on WebSphere Application Server V5 for z/OS"
- "Running the IMS Java Sample Applications on WebSphere Application Server V6 for z/OS" on page 56

Running the IMS Java Sample Applications on WebSphere Application Server V5 for z/OS

Prerequisite: "Running the IMS Java IVP on WebSphere Application Server for z/OS" on page 44

To run the IMS Java sample applications on WebSphere Application Server for z/OS:

1. "Installing the Data Source for the IMS Java Samples on WebSphere Application Server V5 for z/OS"
2. "Installing the IMS Java Sample Applications on WebSphere Application Server V5 for z/OS" on page 54
3. "Testing the IMS Java Sample Applications on WebSphere Application Server V5 for z/OS" on page 55

Installing the Data Source for the IMS Java Samples on WebSphere Application Server V5 for z/OS

The DataSource facility is a factory for connections to a physical data source, or database. A data source is registered with a naming service based on the Java

Naming and Directory (JNDI) API. `DataSource` objects have properties that pertain to the actual data source that an application needs to access.

Requirement: You must use the `DataSource` facility, which replaces the `DriverManager` facility, because the `DriverManager` facility is not supported by the J2EE Connection Architecture Specification.

To install the data source for the IMS Java samples:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the name of IMS JDBC resource adapter that you chose when you installed the adapter.

   A configuration dialog is displayed.
3. Under Additional Properties, click **J2C Connection Factories**.
4. Click **New**.

   A configuration dialog is displayed.
5. Type the following information:
   Name: name for the data source
   JNDI Name: path to the data source.
   - For the phonebook sample, type: `imsjavaPhonebook`
   - For the dealership sample, type: `jdbc/DealershipSample`
6. Click **OK**.

   The data source is listed in the J2C Connection Factories.
7. Click the name of the data source that you installed in step 5.
8. Under Additional Properties, click **Custom Properties**.

   Six properties are listed in a table.
9. In the **DRAName** row, click the dash symbol in the **Value** column.
10. In the **Value** field, type bytes 4-7 of the DRA startup table module name (usually the IMS system ID). For more information about the DRA startup table, see “Configuring WebSphere Application Server V6 for z/OS to Access IMS” on page 42.
11. Click **OK**.

    The properties table displays the DRA name that you just entered.
12. In the **DatabaseViewName** row, click the dash symbol in the **Value** column.
13. In the **Value** field, type the fully-qualified `DLIDatabaseView` subclass name.
    - For the phonebook sample, type: `samples.ivp.DFSIVP37DatabaseView`
    - For the dealership sample, type:`samples.dealership.AUTPSB11DatabaseView`
14. Click **OK**.

    The properties table displays the host name that you just entered.
15. Optionally, set the trace level for the applications. See “Enabling J2EE Tracing with WebSphere Application Server V5” on page 103.
16. Click **Save**.

    The Save page is displayed.
17. Under **Save to Master Configuration**, click **Save**.
18. Restart the server.

Next: “Installing the IMS Java Sample Applications on WebSphere Application Server V5 for z/OS” on page 54

Installing the IMS Java Sample Applications on WebSphere Application Server V5 for z/OS

Prerequisite: "Installing the Data Source for the IMS Java Samples on WebSphere Application Server V5 for z/OS" on page 52

This topic describes how to install one of the IMS Java sample applications on WebSphere Application Server for z/OS. The two sample applications are the phonebook sample and the dealership sample. You must perform this task once for each sample.

To install the sample applications:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Install New Application**.

   A dialog for installing a new application is displayed.
2. Select **Server path** and type the path to the EAR file.

   EAR file path:

   - For the phonebook sample, type the path to IMSJavaPhonebook.ear:

     pathprefix`/usr/lpp/ims/imsjava91/samples/ivp/was/IMSJavaPhonebook.ear`
   - For the dealership sample, type the path to imsjavaDealership.ear:

     pathprefix`/usr/lpp/ims/imsjava91/samples/dealership/was/imsjavaDealership.ear`
3. Click **Next**.
4. Accept the defaults and click **Next**.

   The Install New Application wizard is started. Step 1, ″Provide options to perform the installation,″ is displayed.
5. Clear the **Create MBeans for Resources** check box.
6. Click **Next**.

   Step 2, ″Provide JNDI Names for Beans,″ is displayed.
7. In the **JNDI Name** field, type the path to the EJB home interface.
   - For the phonebook sample, verify that name is as follows:

     `ejb/samples/phonebook/was/IMSJavaPhonebookSessionHome`
   - For the dealership sample, type:
     `samples.dealership.was.DealershipSessionHome`
8. Click **Next**.

   Step 3, ″Map resource references to resources,″ is displayed.
9. For the phonebook sample, verify that the JNDI name of resource references of the IMSJava phSample EJB module is imsjavaPhonebook.

   For the dealership sample, in the **JNDI Name** field for the IMSDealershipWeb module, type: `jdbc/DealershipSample`
10. Click **Next**.

    Step 4, ″Map virtual hosts for web modules,″ is displayed.
11. Accept the defaults and click **Next**.

    Step 5, ″Map modules to application servers,″ is displayed.
12. Accept the defaults and click **Next**.

    Step 6, ″Ensure all unprotected 2.0 methods have the correct level of protection,″ is displayed.
13. Make any necessary changes and click **Next**.

The options that you specified are displayed in Step 7, ″Summary,″ of the Install New Application wizard.

14. Verify that the options are correct, and then click **Finish**.

    A message is displayed that indicates first that the application is being installed, and then that the installation was successful.

15. Click **Save to Master Configuration**.

    The Save page is displayed.

16. Under **Save to Master Configuration**, click **Save**.

Next: “Testing the IMS Java Sample Applications on WebSphere Application Server V5 for z/OS”

Testing the IMS Java Sample Applications on WebSphere Application Server V5 for z/OS

Prerequisite: “Installing the IMS Java Sample Applications on WebSphere Application Server V5 for z/OS” on page 54

This section describes how to test the phonebook or dealership sample application on WebSphere Application Server for z/OS.

To test the phonebook or dealership sample:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Enterprise Applications**.

   The application that you installed is listed with a red X, which indicates that the application is stopped.

2. Select the application.
   - For the phonebook sample, select **IMSJava pbSample**.
   - For the dealership sample, select **IMSDealershipEAR**.

3. Click **Start**.

   The application is listed with a green arrow, which indicates that the application is started.

4. Open a Web browser.
5. Type the Web address of the application.
   - For the phonebook sample, type:

     `http://host_IP_address:port/IMSJavaPhonebookWeb/IMSJavaPhonebook.html`
   - For the dealership sample, type:

     `http://host_IP_address:port/IMSDealershipWeb/dealership.html`

   An input Web page opens.
   - For the phonebook sample, the page is titled **WebSphere Phonebook Sample for IMS Java**.
   - For the dealership sample, the page is titled **Find a car in stock**.

6. Type input.
   - For the phonebook, type the following information:

     Last Name: `LAST1`
   - For the dealership sample, verify that **Car Make** and **VIN Number** fields contain the following information:

     Car Make: `FORD`

     VIN Number: `V234567890123456789V`

7. Click **Submit**.

   If WebSphere Application Server for z/OS is configured properly, the output is displayed.

   - For the phonebook, the following information is displayed:

     ```
     Result: Person found! FirstName: FIRST1 LastName: LAST1
     Extension: 8-111-1111 ZipCode: D01/R01
     ```

   - For the dealership sample, a message indicating that the query was successful is displayed.

Running the IMS Java Sample Applications on WebSphere Application Server V6 for z/OS

Prerequisite: "Running the IMS Java IVP on WebSphere Application Server for z/OS" on page 44

To run the IMS Java sample applications on WebSphere Application Server for z/OS:

1. "Installing the Data Source for the IMS Java Samples on WebSphere Application Server V6 for z/OS"
2. "Installing the IMS Java Sample Applications on WebSphere Application Server V6 for z/OS" on page 57
3. "Testing the IMS Java Sample Applications on WebSphere Application Server V6 for z/OS" on page 58

Installing the Data Source for the IMS Java Samples on WebSphere Application Server V6 for z/OS

The `DataSource` facility is a factory for connections to a physical data source, or database. A data source is registered with a naming service based on the Java Naming and Directory (JNDI) API. `DataSource` objects have properties that pertain to the actual data source that an application needs to access.

Requirement: You must use the `DataSource` facility, which replaces the `DriverManager` facility, because the `DriverManager` facility is not supported by the J2EE Connection Architecture Specification.

To install the data source for the IMS Java samples:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.

2. Click the name of IMS JDBC resource adapter that you chose when you installed the adapter.

   A configuration dialog is displayed.

3. Under Additional Properties, click **J2C connection factories**.
4. Click **New**.

   A configuration dialog is displayed.

5. Type the following information:

   Name: name for the data source

   JNDI Name: path to the data source.

   - For the phonebook sample, type: `imsjavaPhonebook`
   - For the dealership sample, type: `jdbc/DealershipSample`

6. Click **OK**.

The data source is listed in the J2C Connection Factories.

7. Click the name of the data source that you installed in step 5.
8. Under Additional Properties, click **Custom properties**.

   Three properties are listed in a table.
9. Click **DatabaseViewName**.

   A configuration dialog is displayed.
10. In the **Value** field, type the fully-qualified `DLIDatabaseView` subclass name.
    - For the phonebook sample, type: `samples.ivp.DFSIVP37DatabaseView`
    - For the dealership sample, type:`samples.dealership.AUTPSB11DatabaseView`
11. Click **OK**.

    The properties table displays the DatabaseViewName value that you just entered.
12. Click **DRAName**.

    A configuration dialog is displayed.
13. In the **Value** field, type bytes 4-7 of the DRA startup table module name (usually the IMS system ID). For more information about the DRA startup table, see “Configuring WebSphere Application Server V6 for z/OS to Access IMS” on page 42.
14. Click **OK**.

    The properties table displays the DRA name that you just entered.
15. In the messages box, click **Save**.

    The save page is displayed.
16. Click **Save** to update the master repository with your changes.
17. Restart the server in order for the custom service to take effect.

Next: “Installing the IMS Java Sample Applications on WebSphere Application Server V6 for z/OS”

Installing the IMS Java Sample Applications on WebSphere Application Server V6 for z/OS

Prerequisite: “Installing the Data Source for the IMS Java Samples on WebSphere Application Server V6 for z/OS” on page 56

This topic describes how to install one of the IMS Java sample applications on WebSphere Application Server for z/OS. The two sample applications are the phonebook sample and the dealership sample. You must perform this task once for each sample.

To install the sample applications:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Install New Application**.

   A dialog for installing a new application is displayed.
2. Select **Remote file system** and type the path to the EAR file:
   - For the phonebook sample, type the path to IMSJavaPhonebook.ear:

     pathprefix`/usr/lpp/ims/imsjava91/samples/phonebook/was/IMSJavaPhonebook.ear`
   - For the dealership sample, type the path to imsjavaDealership.ear:

     pathprefix`/usr/lpp/ims/imsjava91/samples/dealership/was/imsjavaDealership.ear`

3. Click **Next**.
4. Accept the defaults and click **Next**.

   Application security warnings are displayed.
5. Click **Continue**.

   The Install New Application wizard is started.
6. Click **Step 3**.

   Step 3, ″Provide JNDI Names for Beans,″ is displayed.
7. In the **JNDI Name** field, type the path to the EJB home interface.
   - For the phonebook sample, verify that name is ejb/samples/phonebook/was/IMSJavaPhonebookSessionHome.
   - For the dealership sample, type: `samples.dealership.was.DealershipSessionHome`
8. Click **Step 4**.

   Step 4, ″Map resource references to resources,″ is displayed.
9. For the phonebook sample, verify that the JNDI name of resource references of the IMSJava phSample EJB module is imsjavaPhonebook.

   For the dealership sample, in the **JNDI Name** field for the IMSDealershipWeb module, type: `jdbc/DealershipSample`
10. Click **Step 7**.

    If application resource warnings appear, verify that the resource assignments are correct and click **Continue**.
11. On the summary page, verify that the options are correct, and then click **Finish**.

    A message is displayed that indicates first that the application is being installed, and then that the installation was successful.
12. Click **Save to Master Configuration**.

    The save page is displayed.
13. Click **Save** to update the master repository with your changes.

Next: “Testing the IMS Java Sample Applications on WebSphere Application Server V6 for z/OS”

Testing the IMS Java Sample Applications on WebSphere Application Server V6 for z/OS

Prerequisite: “Installing the IMS Java Sample Applications on WebSphere Application Server V6 for z/OS” on page 57

This section describes how to test the phonebook or dealership sample application on WebSphere Application Server for z/OS.

To test the phonebook or dealership sample:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Enterprise Applications**.

   The application that you installed is listed with a red X, which indicates that the application is stopped.
2. Select the application.
   - For the phonebook sample, select **IMSJava pbSample**.
   - For the dealership sample, select **IMSDealershipEAR**.
3. Click **Start**.

The application is listed with a green arrow, which indicates that the application is started.

4. Open a Web browser.
5. Type the Web address of the application.
   - For the phonebook sample, type:

     `http://host_IP_address:port/IMSJavaPhonebookWeb/IMSJavaPhonebook.html`
   - For the dealership sample, type:

     `http://host_IP_address:port/IMSDealershipWeb/dealership.html`

   An input Web page opens.
   - For the phonebook sample, the page is titled **WebSphere Phonebook Sample for IMS Java**.
   - For the dealership sample, the page is titled **Find a car in stock**.
6. Type input.
   - For the phonebook, type the following information:

     Last Name: `LAST1`
   - For the dealership sample, verify that **Car Make** and **VIN Number** fields contain the following information:

     Car Make: `FORD`

     VIN Number: `V234567890123456789V`
7. Click **Submit**.

   If WebSphere Application Server for z/OS is configured properly, the output is displayed.
   - For the phonebook, the following information is displayed:

     ```
     Result: Person found! FirstName: FIRST1 LastName: LAST1
     Extension: 8-111-1111 ZipCode: D01/R01
     ```
   - For the dealership sample, a message indicating that the query was successful is displayed.

Running Your Applications on WebSphere Application Server for z/OS

This topic provides the high-level steps that are required to run an application that accesses IMS DB from WebSphere Application Server for z/OS.

To run your application on WebSphere Application Server for z/OS:

1. Set the WebSphere Application Server for z/OS classpath to point to the IMS Java metadata class.
2. Install the data source for the application as a J2C connection factory.
3. Install the application.
4. If you are using WebSphere Application Server V5 for z/OS, add the required XML files to the application class path.

Detailed information about running an application on specific versions of WebSphere Application Server for z/OS is available:

- “Running Your Applications on WebSphere Application Server V5 for z/OS” on page 60
- “Running Your Applications on WebSphere Application Server V6 for z/OS” on page 65

Running Your Applications on WebSphere Application Server V5 for z/OS

Prerequisite: "Running the IMS Java IVP on WebSphere Application Server for z/OS" on page 44

To run your applications on WebSphere Application Server for z/OS:

1. "Setting the WebSphere Application Server V5 for z/OS Classpath"
2. "Installing the Data Source for Your Application on WebSphere Application Server V5 for z/OS"
3. "Installing Your Application on WebSphere Application Server V5 for z/OS" on page 61
4. "Adding the XML Files to the Application Classpath on WebSphere Application Server V5 for z/OS" on page 62
5. "Enabling J2EE Tracing with WebSphere Application Server V5 for z/OS" on page 63

Setting the WebSphere Application Server V5 for z/OS Classpath

Your application can include the IMS Java metadata class (`DLIDatabaseView` subclass) or the metadata class can be stored elsewhere.

If your application does not include the metadata class, you must set the WebSphere Application Server for z/OS classpath to the location of the IMS Java metadata class that is used by the application.

One way to set the classpath is to add these files to the IMS JDBC resource adapter classpath.

To add the required files to the IMS JDBC resource adapter classpath:

1. From the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the name of the IMS JDBC resource adapter.

   A configuration dialog is displayed.
3. In the **Classpath** field, add the path to the required files. Include the file name for JAR files. Do not delete imsjava.jar.
4. Click **OK**.

Installing the Data Source for Your Application on WebSphere Application Server V5 for z/OS

The `DataSource` facility is a factory for connections to a physical data source, or database. A data source is registered with a naming service based on the Java Naming and Directory (JNDI) API. `DataSource` objects have properties that pertain to the actual data source that an application needs to access.

Requirement: You must use the `DataSource` facility, which replaces the `DriverManager` facility, because the `DriverManager` facility is not supported by the J2EE Connection Architecture Specification.

To install the data source for your application:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

A list of resource adapters is displayed.

2. Click the name of the IMS JDBC resource adapter that you chose when you installed the adapter.

   A configuration dialog is displayed.

3. Under Additional Properties, click **J2C Connection Factories**.
4. Click **New**.

   A configuration dialog is displayed.

5. Type the following information:
   Name: name for the data source
   JNDI Name: path to the data source.
6. Click **OK**.

   The data source is listed in the J2C Connection Factories.

7. Click the name of the data source that you installed in step 5.
8. Under Additional Properties, click **Custom Properties**.

   Six properties are listed in a table.

9. In the **DRAName** row, click the dash symbol in the **Value** column.
10. In the **Value** field, type bytes 4-7 of the DRA startup table module name (usually the IMS system ID). For more information about the DRA startup table, see “Configuring WebSphere Application Server V6 for z/OS to Access IMS” on page 42.
11. Click **OK**.

    The properties table displays the DRA name that you just entered.

12. In the **DatabaseViewName** row, click the dash symbol in the **Value** column.
13. Optional: In the **Value** field, type the fully-qualified `DLIDatabaseView` subclass name.

    If you do set the subclass name, you must either create a data source for every PSB an application accesses, or you must override the `DLIDatabaseView` subclass name in the `DataSource` object by calling the `setDatabaseView` method and providing the fully-qualified name of the subclass.

    If you do not set the subclass name, you need to create a data source only for each IMS. In the application, define the `DLIDatabaseView` subclass name in the `DataSource` object by calling the `setDatabaseView` method and providing the fully-qualified name of the subclass.

14. Click **OK**.

    The properties table displays the host name that you just entered.

15. Optionally, set the trace level for the applications. See “Enabling J2EE Tracing with WebSphere Application Server V5 for z/OS” on page 63.
16. Click **Save**.

    The Save page is displayed.

17. Under **Save to Master Configuration**, click **Save**.
18. Restart the server to ensure that the changes have been made.

Next: “Installing Your Application on WebSphere Application Server V5 for z/OS”

Installing Your Application on WebSphere Application Server V5 for z/OS

Prerequisite: “Installing the Data Source for Your Application on WebSphere Application Server V6 for z/OS” on page 65

This section describes how to deploy an application on WebSphere Application Server for z/OS.

To install your application:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Install New Application**.

   A dialog for installing a new application is displayed.
2. Type the path to the EAR file.
3. Click **Next**.
4. Accept the defaults and click **Next**.

   The Install New Application wizard is started. Step 1, ″Provide options to perform the installation,″ is displayed.
5. Clear the **Create MBeans for Resources** check box.
6. Click **Next**.

   Step 2, ″Provide JNDI Names for Beans,″ is displayed.
7. In the **JNDI Name** field, type the path to the EJB home interface.
8. Click **Next**.

   Step 3, ″Map resource references to resources,″ is displayed.
9. Type the JNDI name for the data source that you created in “Installing the Data Source for Your Application on WebSphere Application Server V6 for z/OS” on page 65.
10. Click **Next**.

    Step 4, ″Map modules to application servers,″ is displayed.
11. Accept the defaults and click **Next**.

    Step 5, ″Correct use of System Identity,″ is displayed.
12. Verify that no role is selected and click **Next**.

    Step 6, ″Ensure all unprotected 2.0 methods have the correct level of protection,″ is displayed.
13. Make any necessary changes and click **Next**.

    The options that you specified are displayed in Step 7, ″Summary,″ of the Install New Application wizard.
14. Verify that the options are correct, and then click **Finish**.

    A message is displayed that indicates first that the application is being installed, and then that the installation was successful.
15. Click **Save to Master Configuration**.

    The Save page is displayed.
16. Under **Save to Master Configuration**, click **Save**.
17. Restart the server to ensure that the changes have been made.

Adding the XML Files to the Application Classpath on WebSphere Application Server V5 for z/OS

If your application uses the `storeXML()` or `retrieveXML()` UDFs, you must add the XML parser files to the application's classpath.

Prerequisite: “Installing the IMS Java IVP on WebSphere Application Server V5 for z/OS” on page 46

To add the XML files to the application classpath:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Enterprise Applications**.

   The application IMSJava IVP is listed.
2. Click the name of your application.
3. Under General Properties, in the **Classloader Mode** field, select PARENT_LAST.
4. Click **Apply**.
5. Under Additional Properties, click **Libraries**.
6. Click **Add**.
7. In the Library **Name** field, select the shared library that you created in "Adding the Required XML Files to the WebSphere Application Server V5 for z/OS Classpath" on page 39. For example, select XML Shared Library.
8. Click **OK**.
9. Click **Save**.

   The Save page is displayed.
10. Under **Save to Master Configuration**, click **Save**.

Next: "Testing the IMS Java IVP on WebSphere Application Server V5 for z/OS" on page 48

Enabling J2EE Tracing with WebSphere Application Server V5 for z/OS

You can trace the IMS library classes by using the WebSphere Application Server for z/OS tracing service.

To enable tracing if you have not yet specified the level of tracing:

1. "Specifying the Level of Tracing"
2. "Specifying the Application Server and the Package to Trace" on page 64

To enable tracing if you have already specified the level of tracing:

1. "Specifying at Runtime the Application Server and the Package to Trace" on page 64

You can also trace the IMS library classes or your applications using the `com.ibm.ims.base.XMLTrace` class. The `XMLTrace` class is an IMS Java-provided class that represents the trace as an XML document. You can trace different levels of the code depending on the trace level. For more information, see the IMS Java API Specification.

Specifying the Level of Tracing: To use the WebSphere Application Server for z/OS tracing service, you must first specify the level of tracing.

To specify the level of tracing:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click **IMS JDBC resource adapter**.

   A configuration dialog is displayed.
3. Under Additional Properties, click **J2C Connection Factories**.

   A list of connection factories is displayed.

4. Click the name of the J2C connection factory for which you want to enable tracing.

   A configuration dialog is displayed.
5. Under Additional Properties, click **Custom Properties**.

   Properties are listed in a table.
6. In the **Trace Level** row, click the number in the **Value** column.
7. In the **Value** field, type the trace level.
8. Click **OK**.

   The properties table displays the trace level that you just entered.
9. Click **Save**.

   The Save page is displayed.
10. Under **Save to Master Configuration**, click **Save** to ensure that the changes are made.

Specifying the Application Server and the Package to Trace: After you specify the level of tracing, specify the application server and package to trace and then restart the server.

To specify the application server and the package to trace:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Servers**, and then click **Application Servers**.

   A list of application servers is displayed.
2. Click the name of the server on which you want to enable tracing.
3. Under Additional Properties, click **Diagnostic Trace Service**.

   A configuration dialog for Diagnostic Trace Service is displayed.
4. Select the **Enable Trace** check box.
5. In the **Trace Specification** field after any other traces that are listed, type:
   `com.ibm.connector2.ims.db.*=all=enabled`
6. Click **Apply**.
7. Click **Save**.

   The Save page is displayed.
8. Under **Save to Master Configuration**, click **Save** to ensure that the changes are made.
9. Restart the server.

Specifying at Runtime the Application Server and the Package to Trace: You can turn tracing on and off by specifying at runtime the server and package to trace. You do not need to restart your server each time.

To specify the application server and the package to trace at runtime:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Servers**, and then click **Application Servers**.

   A list of application servers is displayed.
2. Click the name of the server on which you want to enable tracing.
3. Under Additional Properties, click **Diagnostic Trace Service**.

   A configuration dialog for Diagnostic Trace Service is displayed.
4. Click the **Runtime** tab.
5. In the **Trace Specification** field after any other traces that are listed, type:
   `com.ibm.connector2.ims.db.*=all=enabled`

6. Click **Apply**.

Running Your Applications on WebSphere Application Server V6 for z/OS

Prerequisite: "Running the IMS Java IVP on WebSphere Application Server for z/OS" on page 44

To deploy your applications on WebSphere Application Server for z/OS:

1. "Setting the WebSphere Application Server V6 for z/OS Classpath"
2. "Installing the Data Source for Your Application on WebSphere Application Server V6 for z/OS"
3. "Installing Your Application on WebSphere Application Server V6 for z/OS" on page 67
4. "Enabling J2EE Tracing with WebSphere Application Server V6 for z/OS" on page 67

Setting the WebSphere Application Server V6 for z/OS Classpath

Your application can include the IMS Java metadata class (`DLIDatabaseView` subclass) or the metadata class can be stored elsewhere.

If your application does not include the metadata class, you must set the WebSphere Application Server for z/OS classpath to the location of the IMS Java metadata class that is used by the application.

One way to set the classpath is to add these files to the IMS JDBC resource adapter classpath.

To add the required files to the IMS JDBC resource adapter classpath:

1. From the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the name of the IMS JDBC resource adapter.

   A configuration dialog is displayed.
3. In the **Classpath** field, add the path to the required files. Include the file name for JAR files. Do not delete imsjava.jar.
4. Click **OK**.
5. In the messages box, click **Save**.

   The save page is displayed.
6. Click **Save** to update the master repository with your changes.

Installing the Data Source for Your Application on WebSphere Application Server V6 for z/OS

The `DataSource` facility is a factory for connections to a physical data source, or database. A data source is registered with a naming service based on the Java Naming and Directory (JNDI) API. `DataSource` objects have properties that pertain to the actual data source that an application needs to access.

Requirement: You must use the `DataSource` facility, which replaces the `DriverManager` facility, because the `DriverManager` facility is not supported by the J2EE Connection Architecture Specification.

To install the data source for your application:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the name of the IMS JDBC resource adapter that you chose when you installed the adapter.

   A configuration dialog is displayed.
3. Under Additional Properties, click **J2C connection factories**.
4. Click **New**.

   A configuration dialog is displayed.
5. Type the following information:
   Name: name for the data source
   JNDI Name: the JNDI name for the data source.
6. Click **OK**.

   The data source is listed in the J2C Connection Factories.
7. Click the name of the data source that you installed in step 5.
8. Under Additional Properties, click **Custom Properties**.

   Three properties are listed in a table.
9. Click **DatabaseViewName**.

   A configuration dialog is displayed.
10. Optional: In the **Value** field, type the fully-qualified `DLIDatabaseView` subclass name.

    If you do set the subclass name, you must either create a data source for every PSB an application accesses, or you must override the `DLIDatabaseView` subclass name in the `DataSource` object that is within the application by calling the `setDatabaseView` method and providing the fully-qualified name of the subclass.

    If you do not set the subclass name, you need to create a data source only for each IMS. In the application, define the `DLIDatabaseView` subclass name in the `DataSource` object by calling the `setDatabaseView` method and providing the fully-qualified name of the subclass.
11. Click **OK**.

    The properties table displays the DatabaseViewName value that you just entered.
12. Click **DRAName**.

    A configuration dialog is displayed.
13. In the **Value** field, type bytes 4-7 of the DRA startup table module name (usually the IMS system ID). For more information about the DRA startup table, see “Configuring WebSphere Application Server V6 for z/OS to Access IMS” on page 42.
14. Click **OK**.

    The properties table displays the DRA name that you just entered.
15. In the messages box, click **Save**.

    The save page is displayed.
16. Click **Save** to update the master repository with your changes.
17. Restart the server in order for the custom service to take effect.

Next: “Installing Your Application on WebSphere Application Server V6 for z/OS” on page 67

Installing Your Application on WebSphere Application Server V6 for z/OS

Prerequisite: "Installing the Data Source for Your Application on WebSphere Application Server V6 for z/OS" on page 65

This section describes how to deploy an application on WebSphere Application Server for z/OS.

To install your application:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Install New Application**.

   A dialog for installing a new application is displayed.
2. Type the path to the EAR file.
3. Click **Next**.
4. Accept the defaults and click **Next**.

   The Install New Application wizard is started. Step 1, ″Provide options to perform the installation,″ is displayed.
5. Click **Step 4: Provide JNDI Names for Beans**.
6. In the **JNDI Name** field, type the path to the EJB home interface.
7. Click **Step 7: Summary**.

   The options that you specified are displayed.
8. Verify that the options are correct, and then click **Finish**.

   A message is displayed that indicates first that the application is being installed, and then that the installation was successful.
9. Click **Save to Master Configuration**.

   The Save page is displayed.
10. Under **Save to Master Configuration**, click **Save**.

Enabling J2EE Tracing with WebSphere Application Server V6 for z/OS

You can trace the IMS library classes by using the WebSphere Application Server for z/OS tracing service.

To enable tracing if you have not yet specified the level of tracing:

1. "Specifying the Level of Tracing"
2. "Specifying the Application Server and the Package to Trace" on page 68

To enable tracing if you have already specified the level of tracing:

1. "Specifying at Runtime the Application Server and the Package to Trace" on page 68

You can also trace the IMS library classes or your applications using the `com.ibm.ims.base.XMLTrace` class. The `XMLTrace` class is an IMS Java-provided class that represents the trace as an XML document. You can trace different levels of the code depending on the trace level. For more information, see the IMS Java API Specification.

Specifying the Level of Tracing: To use the WebSphere Application Server for z/OS tracing service, you must first specify the level of tracing.

To specify the level of tracing:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.
   A list of resource adapters is displayed.
2. Click the name of the IMS JDBC resource adapter.
   A configuration dialog is displayed.
3. Under Additional Properties, click **J2C connection factories**.
   A list of connection factories is displayed.
4. Click the name of the J2C connection factory for which you want to enable tracing.
   A configuration dialog is displayed.
5. Under Additional Properties, click **Custom Properties**.
   Properties are listed in a table.
6. Click **TraceLevel** row.
7. In the **Value** field, type the trace level.
8. Click **OK**.
   The properties table displays the trace level that you just entered.
9. In the messages box, click **Save**.
   The save page is displayed.
10. Click **Save** to update the master repository with your changes.

Specifying the Application Server and the Package to Trace: After you specify the level of tracing, specify the application server and package to trace and then restart the server.

To specify the application server and the package to trace:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Servers**, and then click **Application Servers**.
   A list of application servers is displayed.
2. Click the name of the server on which you want to enable tracing.
3. Under Troubleshooting, click **Diagnostic Trace Service**.
   A configuration dialog for Diagnostic Trace Service is displayed.
4. Select the **Enable Log** check box and click **OK**.
5. Under Troubleshooting, click **Change Log Detail Levels**.
6. Click the plus sign (+) next to **com.ibm.connector2**.
7. Click **com.ibm.connector2.ims.***
8. From the list of trace detail levels, click **all**.
9. Verify that `com.ibm.connector2.ims.*=all` appears in the text box and click **OK**.
10. In the messages box, click **Save**.
    The save page is displayed.
11. Click **Save** to update the master repository with your changes.
12. Restart the server.

Specifying at Runtime the Application Server and the Package to Trace: You can turn tracing on and off by specifying at runtime the server and package to trace. You do not need to restart your server each time.

To specify the application server and the package to trace at runtime:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Servers**, and then click **Application Servers**.
   A list of application servers is displayed.
2. Click the name of the server on which you want to enable tracing.
3. Under Troubleshooting, click **Change Log Detail Levels**.
   A configuration dialog for Change Log Detail Levels is displayed.
4. Click the **Runtime** tab.
5. Click the plus sign (+) next to **com.ibm.connector2**.
6. Click **com.ibm.connector2.ims.***
7. From the list of trace detail levels, click **all**.
8. Verify that `com.ibm.connector2.ims.*=all` appears in the text box and click **OK**.
9. Click **OK**.

Developing Enterprise Applications that Access IMS DB

Enterprise applications that access IMS DB can be servlets or EJBs. The EJBs can be bean-managed or container-managed. This topic describes the programming models for these different types of enterprise applications. These programming models apply to enterprise applications that run on either WebSphere Application Server for z/OS or WebSphere Application Server on a non-z/OS platform.

In this topic:

- "Bean-Managed EJB Programming Model"
- "Container-Managed EJB Programming Model" on page 71
- "Servlet Programming Model" on page 71
- "Programming Requirements for WebSphere Application Server for z/OS" on page 72
- "Deployment Descriptor Requirements for IMS Java" on page 72

Bean-Managed EJB Programming Model

In bean-managed EJBs, you programmatically define the transaction boundaries. To define an EJB as bean-managed, set the transaction-type property, which is in the ejb-jar.xml file of the EJB jar file, to Bean. You must manage the scope of the transaction by using either the `javax.transaction.UserTransaction` or `java.sql.Connection` interface. This topic describes how to use both interfaces:

- "Transaction Demarcation Using the javax.transaction.UserTransaction Interface"
- "Transaction Demarcation Using the java.sql.Connection Interface" on page 70

Transaction Demarcation Using the javax.transaction.UserTransaction Interface

The programming model applies either to applications that run on WebSphere Application Server on a non-z/OS platform or to applications that run on WebSphere Application Server for z/OS. With the `javax.transaction.UserTransaction` interface, you can define when the scope of the transaction begins and ends, and when the transaction commits or rolls back. The EJB container supplies the EJB with a `javax.ejb.SessionContext` object that allows the `javax.transaction.UserTransaction` interface to perform the required operations to manage the transaction.

```
try {
    // Use the javax.ejb.SessionContext set by the EJB container to instantiate
    // a new UserTransaction
    javax.transaction.UserTransaction userTransaction =
```

```
                                                sessionContext.getUserTransaction();

        // Begin the scope of this transaction
        userTransaction.begin();

        // Perform JNDI lookup to obtain the data source (the IVP datasource for
        // example) and cast
        javax.sql.DataSource dataSource = (javax.sql.DataSource)
                           initialContext.lookup("java:comp/env/jdbc/IMSIVP");

        // Get a connection to the data source
        java.sql.Connection connection = dataSource.getConnection();

        // Create an SQL statement using the connection
        java.sql.Statement statement = connection.createStatement();

        // Acquire a result set by executing the query using the statement
        java.sql.ResultSet results = statement.executeQuery(...);

        // Commit and complete the scope of this transaction
        userTransaction.commit();

        // Close the connection
        connection.close();

    } catch (Throwable t) {

        // If an exception occurs, roll back the transaction
        userTransaction.rollback();

        // Close the connection
        connection.close();
    }
```

Transaction Demarcation Using the java.sql.Connection Interface

The programming model applies only to applications that run on WebSphere Application Server on a non-z/OS platform and that use the remote database services of IMS Java. With the `java.sql.Connection` interface, you commit or roll back a transaction that is started by the creation of a data source connection. The IMS Java EJB that is on the server side automatically starts a transaction if one does not exist when a connection is created. You can then use this connection to commit or rollback the transaction without using the `javax.transaction.UserTransaction` interface.

Use this programming model only if you do not use the `javax.transaction.UserTransaction` interface.

When you perform the JNDI lookup, specify `"java:comp/env/sourceName"` where `sourceName` is the name of the data source.

```
try {
    // Perform JNDI lookup to obtain the data source (the IVP data source
    // for example) and cast
    javax.sql.DataSource dataSource = (javax.sql.DataSource)
                       initialContext.lookup("java:comp/env/imsjavaRDSIVP");

    // Get a connection to the data source and begin the transaction scope
    java.sql.Connection connection = dataSource.getConnection();

    // Create an SQL statement using the connection
    java.sql.Statement statement = connection.createStatement();

    // Acquire a result set by executing the query using the statement
    java.sql.ResultSet results = statement.executeQuery(...);
```

```
    // Commit and complete the scope of this transaction
    connection.commit();

    // Close the connection
    connection.close();

} catch (Throwable t) {

    // If an exception occurs, rollback the transaction
    connection.rollback();

    // Close the connection
    connection.close();
}
```

Container-Managed EJB Programming Model

In container-managed EJBs, the container manages the transaction demarcation. The demarcation is defined in the ejb-jar.xml file of the EJB. To define an EJB as container-managed, set the transaction-type property, which is in the ejb-jar.xml file of the EJB jar file, to Container. Because the container manages the transaction demarcation, this programming model does not have any transaction logic.

```
try {

    // Perform JNDI lookup to obtain the data source (the IVP data source
    // for example) and cast
    javax.sql.DataSource dataSource = (javax.sql.DataSource)
                          initialContext.lookup("java:comp/env/jdbc/IMSIVP");

    // Get a connection to the data source
    java.sql.Connection connection = dataSource.getConnection();

    // Create an SQL statement using the connection
    java.sql.Statement statement = connection.createStatement();

    // Acquire a result set by executing the query using the statement
    java.sql.ResultSet results = statement.executeQuery(...);

    // Close the connection
    connection.close();

} catch (Throwable t) {

    // Close the connection
    connection.close();
}
```

Servlet Programming Model

Similarly to the bean-managed EJBs, the servlet programming model uses the `UserTransaction` interface to begin, commit, or rollback the transaction. Because the servlet resides outside of the EJB container and cannot use an `EJBContext` object, the initial context requires an additional JNDI lookup to locate and instantiate the `UserTransaction` interface.

```
try {
    // Establish an initial context to manage the environment
    //properties and JNDI names
    javax.naming.InitialContext initialContext = new InitialContext();

    // Locate and instantiate a UserTransaction object that is associated with
    // the initial context using JNDI
    javax.transaction.UserTransaction userTransaction = (UserTransaction)
                                            ic.lookup("java:comp/UserTransaction");
```

```
        // Begin the scope of this transaction
        userTransaction.begin();

        // Perform JNDI lookup to obtain the data source (the IVP data source
        // for example) and cast
        javax.sql.DataSource dataSource = (javax.sql.DataSource)
                                    initialContext.lookup("java:comp/env/jdbc/IMSIVP");

        // Get a connection to the datasource
        java.sql.Connection connection = dataSource.getConnection();

        // Create an SQL statement using the connection
        java.sql.Statement statement = connection.createStatement();

        // Acquire a result set by executing the query using the statement
        java.sql.ResultSet results = statement.executeQuery(...);

        // Commit and complete the scope of this transaction
        userTransaction.commit();

        // Close the connection
        connection.close();

    } catch (Throwable t) {

        // If an exception occurs, roll back the transaction
        userTransaction.rollback();

        // Close the connection
        connection.close();
    }
```

Programming Requirements for WebSphere Application Server for z/OS

The following programming requirements apply to WebSphere Application Server for z/OS EJBs that access IMS databases:

- IMS Java does not support component-managed signon.
- IMS Java does not support shared connections.
- The `java.sql.Connection` object must be acquired, used, and closed within a transaction boundary.
- A global transaction must exist before you create a `Connection` object from a JDBC connection. Either specify container-demarcated transactions in the EJB deployment descriptor or explicitly begin a global transaction by calling the`javax.transaction.UserTransaction` API before creating a JDBC connection.

Deployment Descriptor Requirements for IMS Java

The deployment descriptor for an EJB or servlet has certain requirements for IMS Java. In an EJB, the deployment descriptor is the file ejb-jar.xml. In a servlet, the deployment descriptor is the file web.xml.

You must have a `resource-ref` element in the deployment descriptor. The `resource-ref` element describes external resources. In the `resource-ref` element, you must have the following elements:

```
<res-type>javax.sql.DataSource</res-type>
<res-sharing-scope>Unshareable</res-sharing-scope>
```

The `<res-type>javax.sql.DataSource</res-type>` element specifies the type of data source. The `<res-sharing-scope>Unshareable</res-sharing-scope>` element specifies that the connections are not shareable.

The following example is a resource-ref element from an EJB deployment descriptor:

```
<resource-ref>
 <res-ref-name>jdbc/DealershipSample</res-ref-name>
 <res-type>javax.sql.DataSource</res-type>
 <res-auth>Container</res-auth>
 <res-sharing-scope>Unshareable</res-sharing-scope>
</resource-ref>
```

Chapter 4. Remote Data Access with WebSphere Application Server Applications

With IMS Java remote database services, you can develop and deploy applications that run on non-z/OS platforms and access IMS databases remotely. Unlike other Java solutions for IMS, you do not need to develop a z/OS application or access a legacy z/OS application to have access to IMS data. Therefore, IMS Java is an ideal solution for IMS application development in a WebSphere environment.

Figure 14 shows the components that are required for an enterprise application (in this case, an EJB) on a non-z/OS platform to access IMS DB. The components are described following the figure.

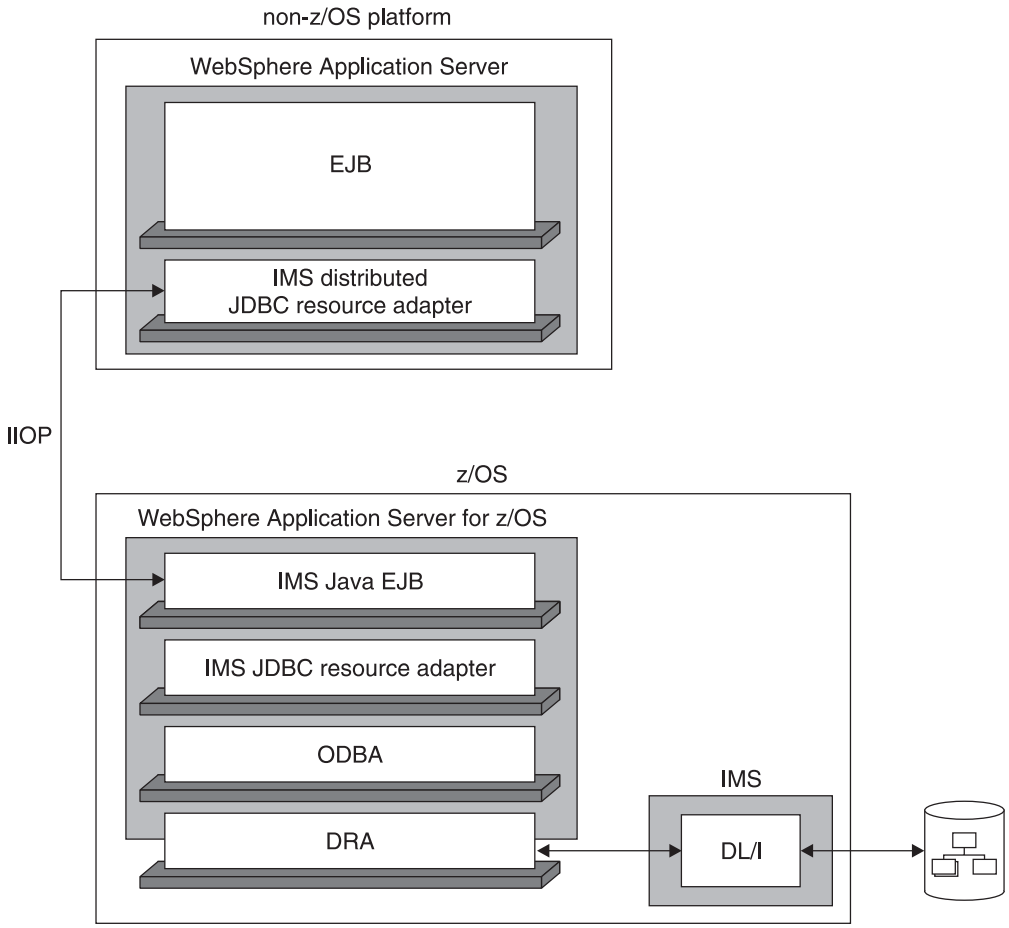


Figure 14. IMS Java and WebSphere Application Server Components

The following components are used for an enterprise application on a non-z/OS platform to access IMS DB:

non-z/OS platform
The operating system that WebSphere Application Server V5 runs on.

WebSphere Application Server
WebSphere Application Server on which the client application runs.

EJB The enterprise application (an EJB in this case) that contains your business logic, and is deployed on WebSphere Application Server. This enterprise

application can be either container managed or bean managed. The enterprise application can be transactional.

IMS distributed JDBC resource adapter
The resource adapter that is deployed on the non-z/OS platform. It contains a type-3 JDBC driver.

IIOP (Internet Inter-ORB Protocol)
IIOP is the protocol that can be used between WebSphere Application Server for z/OS and WebSphere Application Server running on another platform. IIOP allows the servers to exchange data. Data is securely transferred across the Internet using the SSL (Secure Sockets Layer) protocol.

WebSphere Application Server for z/OS
WebSphere Application Server for z/OS is required to manage transaction protocol and communication with RRS. It must reside on the same z/OS LPAR (logical partition) as IMS.

IMS Java EJB
One of two IMS Java-supplied EJBs is the host-side component that facilitates communication with and passes transaction information to the IMS JDBC resource adapter. These EJBs act as listeners for remote requests. Depending on whether there is a transaction context on the non-z/OS platform, either a container-managed or bean-managed IMS Java EJB is used.

IMS JDBC resource adapter
The IMS JDBC resource adapter that is deployed on the z/OS platform. It contains a type-3 JDBC driver.

ODBA Open Database Access is the IMS callable interface for access to IMS DB.

DRA The database resource adapter (DRA) is the bridge between the external subsystem and IMS.

DL/I DL/I is the standard interface to IMS data.

The following topics provide additional information:

- “Downloading IMS Java Files for Remote Database Services” on page 77
- “Configuring the Application Servers for IMS Java Remote Database Services” on page 77
- “Running the IMS Java IVP for Remote Database Services” on page 84
- “Running the IMS Java Sample Applications for Remote Database Services” on page 91
- “Running Your Application on WebSphere Application Server” on page 100
- “WebSphere Application Server EJBs” on page 108

Related Reading: Application programming for distributed enterprise applications is the same as for z/OS enterprise applications. For information on developing enterprise applications for either WebSphere Application Server for z/OS or WebSphere Application Server on a non-z/OS platform, see “Developing Enterprise Applications that Access IMS DB” on page 69.

Downloading IMS Java Files for Remote Database Services

To use IMS Java remote database services to access IMS databases from applications that run on WebSphere Application Server on a non-z/OS platform, you must download IMS Java files from the IMS Java Web site. These files are required in addition to the files that are installed as part of the SMP/E installation of the IMS Java FMID.

To download the required IMS Java files, go to the IMS Web site at http://www.ibm.com/ims and link to the IMS Java Web page for more information.

Configuring the Application Servers for IMS Java Remote Database Services

In addition to the software listed in “IMS Java System Requirements” on page 1, the following software is required:

- WebSphere Application Server V5.0 for z/OS or later. If you have WebSphere Application Server V5.0.2 for z/OS, you must install either V5.0.2.1 or APAR PQ81944.
- Either:
  - WebSphere Application Server V5.0.2.2 with cumulative fixes that include PQ79485
  - WebSphere Application Server V5.0.2.3 or later

The following protocols and z/OS components are required:

- RRS (resource recovery services) for z/OS
- RACF® or equivalent product

Before you can deploy an application on WebSphere Application Server, you must configure the servers. This topic provides the basic steps required to configure the servers.

To configure the servers for IMS Java remote database services:

1. Map the hostnames of the client and server.
2. Install the data source for IMS Java remote database services as a new J2C connection factory on WebSphere Application Server for z/OS. The JNDI name is rdsDataSource. Do not specify a database view name.
3. Install and start the IMS Java remote database services EAR file on WebSphere Application Server for z/OS. The file is imsjavaRDS.ear, which is downloaded from the IMS Java Web site.
4. If you are using WebSphere Application Server V5, add the following required XML files to the server class path:

   ```
   pathprefix/usr/lpp/ims/imsjava91/lib/xml-apis.jar
   pathprefix/usr/lpp/ims/imsjava91/lib/xalan.jar
   pathprefix/usr/lpp/ims/imsjava91/lib/xercesImpl.jar
   ```

5. Install the IMS distributed JDBC resource adapter on WebSphere Application Server. The file name of the resource adapter is imsjavaRDS.rar.

Detailed steps, specific to the version of WebSphere Application Server, are also provided:

- “Configuring the V5 Application Servers for IMS Java Remote Database Services” on page 78

- "Configuring the V6 Application Servers for IMS Java Remote Database Services" on page 81

Configuring the V5 Application Servers for IMS Java Remote Database Services

Prerequisites:

- "Downloading IMS Java Files for Remote Database Services" on page 77
- "Configuring WebSphere Application Server V5 for z/OS" on page 39
- "Running the IMS Java IVP on WebSphere Application Server V5 for z/OS" on page 45

To configure the application servers for IMS Java remote database services:

1. "Mapping Hostnames for the Client and Server"
2. "Installing the Data Source on WebSphere Application Server V5 for z/OS"
3. "Installing the EAR file on WebSphere Application Server V5 for z/OS" on page 79
4. "Adding the XML Files to the EAR Classpath" on page 80
5. "Installing the IMS Distributed JDBC Resource Adapter on WebSphere Application Server V5" on page 81

Next: "Running the IMS Java IVP for Remote Database Services" on page 84

Mapping Hostnames for the Client and Server

Map the hostnames of the client and server so that they can communicate.

To map the hostnames for the client and server:

1. In the z/OS HFS environment, open the file named hosts.
2. In the hosts file, append the client IP address and client hostname. For example:

   ```
   129.42.17.99   IBMCLIENT
   ```
3. On the client, open the file named hosts.
4. In the hosts file, append the server IP address and server hostname. For example:

   ```
   204.146.213.73   IBMSERVER
   ```

Next: "Installing the Data Source on WebSphere Application Server V5 for z/OS"

Installing the Data Source on WebSphere Application Server V5 for z/OS

Unlike the data source for z/OS applications, this data source does not have values for the IMS-specific properties. At runtime, the client application's data source properties will be propagated to an instance of this data source.

Prerequisite: "Mapping Hostnames for the Client and Server"

To install the data source on WebSphere Application Server for z/OS:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.

2. Click the name of IMS JDBC resource adapter that you chose when you installed the adapter.

   A configuration dialog, ″Preparing for the application installation,″ is displayed.
3. Under Additional Properties, click **J2C Connection Factories**.
4. Click **New**.

   A configuration dialog is displayed.
5. Type the following information:
   Name: the name for the data source
   JNDI Name: `rdsDataSource`
6. Click **OK**.

   The data source is listed in the J2C Connection Factories.
7. Click **Save to Master Configuration**.

   The Save to Master Configuration dialog is displayed.
8. Click **Save**.
9. Restart the server.

Next: “Installing the EAR file on WebSphere Application Server V5 for z/OS”

Installing the EAR file on WebSphere Application Server V5 for z/OS

The EAR file contains the two IMS Java-provided EJBs. These stateful session beans act as server-side extensions of the IMS distributed JDBC resource adapter.

Prerequisite: “Installing the Data Source on WebSphere Application Server V5 for z/OS” on page 78

To install the EAR file on WebSphere Application Server for z/OS:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Install New Application**.

   A dialog for installing a new application is displayed.
2. Type the path to the EAR file named imsjavaRDS.ear.
3. Click **Next**.

   A dialog,″ Preparing for application installation,″ is displayed.
4. Accept the defaults and click **Next**.

   The Install New Application wizard starts. Step 1, ″Provide options to perform the installation,″ is displayed.
5. Clear the **Create MBeans for Resources** check box.
6. Click **Next**.

   Step 2, ″Provide JNDI Names for Beans,″ is displayed.
7. In the **JNDI Name** field, verify that the JNDI names are as follows:
   - `ejb/com/ibm/ims/rds/host/HostBeanManagedSessionHome`
   - `ejb/com/ibm/ims/rds/host/HostContainerManagedSessionHome`
8. Click **Next**.

   Step 3, ″Map resource references to resources,″ is displayed.
9. Verify that the JNDI name of the resource reference for both EJBs of the IMS Java Remote Database Services EJB module is rdsDataSource.
10. Click **Next**.

    Step 4, ″Map modules to application servers,″ is displayed.

11. Accept the defaults and click **Next**.

    Step 5, ″Correct use of System Identity,″ is displayed.
12. Verify that no role has been selected and click **Next**.

    Step 6, ″Ensure all unprotected 2.0 methods have the correct level of protection,″ is displayed.
13. Make any necessary changes and click **Next**.

    The options that you specified are displayed in Step 7, ″Summary,″ of the Install New Application wizard.
14. Verify that the options are correct, and then click **Finish**.

    A message is displayed that indicates first that the imsjavaRDS application is being installed, and then that the installation was successful.
15. Click **Save to Master Configuration**.

    The Save page is displayed.
16. Under **Save to Master Configuration**, click **Save**.
17. Restart WebSphere Application Server for z/OS to ensure that changes to the data source have been made and to start the IMS Java Remote Database Services EJBs.
18. Click **Save** to update the master repository with your changes.
19. Click **Applications**, and then click **Enterprise Application**.

    The application IMS Java Remote Database Services is listed with a red X, which indicates that the application is stopped.
20. Select **IMS Java Remote Database Services** and click **Start**.

    The application IMS Java Remote Database Services is listed with a green arrow, which indicates that the application is started.

Next: “Adding the XML Files to the EAR Classpath”

Adding the XML Files to the EAR Classpath

If your application uses the `storeXML()` or `retrieveXML()` UDFs, you must add the XML parser files to the server-side EJB’s classpath.

Prerequisite: “Installing the EAR file on WebSphere Application Server V5 for z/OS” on page 79

To add the XML files to the EJB classpath:

1. From the WebSphere Application Server administrative console, click **Applications**, and then click **Enterprise Applications**.

   The application IMSJava IVP is listed.
2. Click **imsjavaRDS**.
3. Under General Properties, in the **Classloader Mode** field, select PARENT_LAST.
4. Click **Apply**.
5. Under Additional Properties, click **Libraries**.
6. Click **Add**.
7. In the Library **Name** field, select the shared library that you created in “Adding the Required XML Files to the WebSphere Application Server V5 for z/OS Classpath” on page 39. For example, select XML Shared Library.
8. Click **OK**.
9. Click **Save**.

The Save page is displayed.

10. Under **Save to Master Configuration**, click **Save**.

Next: "Installing the IMS Distributed JDBC Resource Adapter on WebSphere Application Server V5"

Installing the IMS Distributed JDBC Resource Adapter on WebSphere Application Server V5

Before deploying applications, you must first set up WebSphere Application Server on the non-z/OS client side by installing the IMS distributed JDBC resource adapter.

The WebSphere Application Server on the client side requires only the IMS distributed JDBC resource adapter, imsjavaRDS.rar.

Prerequisite: "Adding the XML Files to the EAR Classpath" on page 80

To install the IMS distributed JDBC resource adapter:

1. From the client-side WebSphere Application Server administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click **Install RAR**.

   A dialog for installing the resource adapter is displayed.
3. Type the path to the imsjavaRDS.rar file.
4. Click **Next**.

   A configuration dialog is displayed.
5. Click **OK**.

   The IMS distributed JDBC resource adapter is listed.
6. Click **Save to Master Configuration**.

   The Save to Master Configuration dialog is displayed.
7. Click **Save**.

Configuring the V6 Application Servers for IMS Java Remote Database Services

Prerequisites:

- "Downloading IMS Java Files for Remote Database Services" on page 77
- "Configuring WebSphere Application Server V6 for z/OS" on page 42
- "Running the IMS Java IVP on WebSphere Application Server V6 for z/OS" on page 48

To configure the application servers for IMS Java remote database services:

1. "Mapping Hostnames for the Client and Server" on page 82
2. "Installing the Data Source on WebSphere Application Server V6 for z/OS" on page 82
3. "Installing the EAR file on WebSphere Application Server V6 for z/OS" on page 83
4. "Installing the IMS Distributed JDBC Resource Adapter on WebSphere Application Server V6" on page 83

Next: “Running the IMS Java IVP for Remote Database Services” on page 84

Mapping Hostnames for the Client and Server

Map the hostnames of the client and server so that they can communicate.

To map the hostnames for the client and server:

1. In the z/OS HFS environment, open the file named hosts.
2. In the hosts file, append the client IP address and client hostname. For example:

   ```
   129.42.17.99   IBMCLIENT
   ```
3. On the client, open the file named hosts.
4. In the hosts file, append the server IP address and server hostname. For example:

   ```
   204.146.213.73   IBMSERVER
   ```

Next: “Installing the Data Source on WebSphere Application Server V6 for z/OS”

Installing the Data Source on WebSphere Application Server V6 for z/OS

Unlike the data source for z/OS applications, this data source does not have values for the IMS-specific properties. At runtime, the client application’s data source properties will be propagated to an instance of this data source.

Requirement: You must use the `DataSource` facility, which replaces the `DriverManager` facility, because the `DriverManager` facility is not supported by the J2EE Connection Architecture Specification.

Prerequisite: “Mapping Hostnames for the Client and Server”

To install the data source for the IVP:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the name of IMS JDBC resource adapter that you chose when you installed the adapter.

   A configuration dialog is displayed.
3. Under Additional Properties, click **J2C connection factories**.
4. Click **New**.

   A configuration dialog is displayed.
5. Type the following information:
   Name: name for the data source
   JNDI Name: `rdsDataSource`
6. Click **OK**.

   The data source is listed in the J2C Connection Factories.
7. In the messages box, click **Save**.

   The save page is displayed.
8. Click **Save** to update the master repository with your changes.
9. Restart the server.

Next: "Installing the IMS Java IVP on WebSphere Application Server V6 for z/OS" on page 50

Installing the EAR file on WebSphere Application Server V6 for z/OS

The EAR file contains the two IMS Java-provided EJBs. These stateful session beans act as server-side extensions of the IMS distributed JDBC resource adapter.

Prerequisite: "Installing the Data Source on WebSphere Application Server V6 for z/OS" on page 82

To install the IMS Java IVP:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Install New Application**.

   A dialog for installing the application is displayed.
2. Select **Local file system** and type the path to imsjavaRDS.ear.
3. Click **Next**.
4. Accept the defaults and click **Next**.

   The Install New Application wizard is started.
5. Click **Step 7** to accept the installation defaults. Depending on your specific server configuration, you might have to use the wizard to change some default values.
6. Verify that the options are correct, and then click **Finish**.

   A message is displayed that indicates first that the application is being installed, and then that the installation was successful.
7. Click **Save to Master Configuration**.

   The save page is displayed.
8. Click **Save** to update the master repository with your changes.
9. Click **Applications**, and then click **Enterprise Application**.

   The application IMS Java Remote Database Services is listed with a red X, which indicates that the application is stopped.
10. Select **IMS Java Remote Database Services** and click **Start**.

    The application IMS Java Remote Database Services is listed with a green arrow, which indicates that the application is started.

Next: "Testing the IMS Java IVP on WebSphere Application Server V6 for z/OS" on page 50

Installing the IMS Distributed JDBC Resource Adapter on WebSphere Application Server V6

Before deploying applications, you must first set up WebSphere Application Server on the non-z/OS client side by installing the IMS distributed JDBC resource adapter.

Prerequisite: "Installing the EAR file on WebSphere Application Server V6 for z/OS"

To install the IMS distributed JDBC resource adapter:

1. From the WebSphere Application Server administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click **Install RAR**.

A dialog for installing the resource adapter is displayed.

3. Select **Local path** and type the path to the imsjavaRDS.rar file.
4. Click **Next**.

   A configuration dialog is displayed.
5. Click **OK**.

   The IMS distributed JDBC resource adapter is listed.
6. In the messages box, click **Save**.

   The save page is displayed.
7. Click **Save** to update the master repository with your changes.

Running the IMS Java IVP for Remote Database Services

Run the IMS Java IVP for remote database services on WebSphere Application Server to ensure that you have configured IMS and WebSphere Application Server properly. This topic provides the high-level tasks required to run the IMS Java IVP.

To run the IMS Java IVP on WebSphere Application Server:

1. Set the WebSphere Application Server for z/OS classpath to the location of the IMS Java metadata class, which is in the file samples.jar.
2. Install the data source of the IMS Java IVP as a new J2C connection factory on WebSphere Application Server. The JNDI name is imsjavaRDSIVP. The database view name is samples.ivp.DFSIVP37DatabaseView. Also specify the host name and port.
3. Install and start the IMS Java IVP EAR file. The file is IMSJavaRDSIVP.ear.
4. Test the IVP. The URL is http://*host_IP_address:port*/IMSJavaRDSIVPWeb/IMSJavaRDSIVP.html.

Detailed information about running the IMS Java IVP for remote database services on specific versions of WebSphere Application Server is available:

- "Running the IMS Java IVP for Remote Database Services on WebSphere Application Server V5"
- "Running the IMS Java IVP for Remote Database Services on WebSphere Application Server V6" on page 88

Running the IMS Java IVP for Remote Database Services on WebSphere Application Server V5

Prerequisite: "Configuring the Application Servers for IMS Java Remote Database Services" on page 77

To run the IMS Java IVP for remote database services:

- "Setting the WebSphere Application Server V5 for z/OS Classpath" on page 85
- "Installing the Data Source for the IVP on the Client Side" on page 85
- "Installing the IVP on the Client Side" on page 86
- "Testing the IVP on WebSphere Application Server V5" on page 87

Next: "Running the IMS Java Sample Applications for Remote Database Services" on page 91

Setting the WebSphere Application Server V5 for z/OS Classpath

You must set the WebSphere Application Server for z/OS classpath to the location of the IMS Java metadata class, which is in the file samples.jar.

One way to set the classpath is to add samples.jar to the IMS JDBC resource adapter classpath.

To add the samples.jar file to the IMS JDBC resource adapter classpath:

1. From the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the name of the IMS JDBC resource adapter.

   A configuration dialog is displayed.
3. In the **Classpath** field, add *pathprefix*`/usr/lpp/ims/imsjava91/samples/samples.jar`. Do not delete imsjava.jar.
4. Click **OK**.
5. Restart the server.

Installing the Data Source for the IVP on the Client Side

To install the data source on the client side:

1. In the left frame of the client-side WebSphere Application Server administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the IMS distributed JDBC resource adapter.

   A configuration dialog is displayed.
3. Under Additional Properties, click **J2C Connection Factories**.
4. Click **New**.

   A configuration dialog is displayed.
5. Type the following information:

   Name: `imsjavaRDSIVP`
   JNDI Name: `imsjavaRDSIVP`

   Note: To avoid messages J2CA0107I and J2CA0114W, both of which can be ignored, set default values for component-managed authentication alias and container-managed authentication alias.
6. Click **OK**.

   The data source is listed in the J2C Connection Factories.
7. Click the name of the data source that you installed in step 5.
8. Under Additional Properties, click **Custom Properties**.

   Six properties are listed in a table.
9. In the **DRAName** row, click the dash symbol in the **Value** column.
10. In the **Value** field, type bytes 4-7 of the DRA startup table module name (usually the IMS system ID). For more information about the DRA startup table, see “Configuring WebSphere Application Server V6 for z/OS to Access IMS” on page 42.
11. Click **OK**.

    The properties table displays the DRA name that you entered.
12. In the **DatabaseViewName** row, click the dash symbol in the **Value** column.

13. In the **Value** field, type: `samples.ivp.DFSIVP37DatabaseView`
14. In the **HostName** row, click the dash symbol in the **Value** column.
15. In the **Value** field, type the name or IP address of the server.
16. Click **OK**.

    The properties table displays the host name that you entered.
17. In the **PortNumber** row, click the dash symbol in the **Value** column.
18. In the **Value** field, type the IIOP port number of the host machine's server. For example: `2809`
19. Click **OK**.

    The properties table displays the port number that you entered.
20. Optionally, set the trace level for the applications. See "Enabling J2EE Tracing with WebSphere Application Server V5" on page 103.
21. Click **Save**.

    The Save page is displayed.
22. Under **Save to Master Configuration**, click **Save** to ensure that the changes are made.

Next: "Installing the IVP on the Client Side"

Installing the IVP on the Client Side

This topic describes how to deploy the IMS Java IVP for remote database services application on WebSphere Application Server on a non-z/OS platform.

Prerequisite: "Installing the Data Source for the IVP on the Client Side" on page 88

To install the application:

1. From the WebSphere Application Server administrative console, click **Applications**, and then click **Install New Application**.

   A dialog for installing a new application is displayed.
2. Type the path to the EAR file: IMSJavaRDSIVP.ear.
3. Click **Next**.
4. Accept the defaults and click **Next**.

   The Install New Application wizard starts. Step 1, ″Provide options to perform the installation,″ is displayed.
5. Clear the **Create MBeans for Resources** check box.
6. Click **Next**.

   Step 2, ″Provide JNDI Names for Beans,″ is displayed.
7. In the **JNDI Name** fields, verify that the JNDI name is the path to the EJB home interface.
   - For the IVP, verify that the names are as follows:
     - `ejb/samples/ivp/rds/IMSJavaRDSIVPCMSessionHome`
     - `ejb/samples/ivp/rds/IMSJavaRDSIVPBMSessionHome`
8. Click **Next**.

   Step 3, ″Map resource references to resources,″ is displayed.
9. Verify the JNDI name for the resource references.
   - For the IVP, verify that the JNDI name of resource references of the two EJBs within the IMSJavaRDS IVP EJB module are both `imsjavaRDSIVP`.
10. Accept the defaults and click **Next**.

Step 5, ″Map modules to application servers,″ is displayed.

11. Accept the defaults and click **Next**.

    Step 6, ″Ensure all unprotected 2.0 methods have the correct level of protection,″ is displayed.

12. Make any necessary changes and click **Next**.

    The options that you specified are displayed in Step 7, ″Summary,″ of the Install New Application wizard.

13. Verify that the options are correct, and then click **Finish**.

    A message is displayed that indicates first that the application is being installed, and then that the installation was successful.

14. Click **Save to Master Configuration**.

    The Save to Master Configuration dialog is displayed.

15. Click **Save**.
16. Restart the server to ensure that the changes have been made to the data source and to start the IMS Java IVP enterprise application.
17. Click **Applications**, and then click **Enterprise Application**.

    The application IMSJavaRDS IVP is listed with a red X, which indicates that the application is stopped.

18. Select **IMSJavaRDS IVP** and click **Start**.

    The application IMSJavaRDS IVP is listed with a green arrow, which indicates that the application is started.

Next: “Testing the IVP on WebSphere Application Server V5”

Testing the IVP on WebSphere Application Server V5

This section describes how to test the IVP on WebSphere Application Server on a non-z/OS platform. The IVP tests both a container-managed EJB and a bean-managed EJB.

Prerequisite: “Installing the IVP on the Client Side” on page 86

To test the IVP:

1. Open a Web browser.
2. Type the Web address of the IVP:

   `http://host_IP_address:port/IMSJavaRDSIVPWeb/IMSJavaRDSIVP.html`

   An input Web page opens titled IMS Java IVP for Remote Database Services.

3. Select **Container managed** and then click **Submit**.

   If WebSphere Application Server is configured properly, the following information is displayed:

   ```
   Result: IVP successful for the container managed EJB.
   ```

   If WebSphere Application Server is not configured properly, the IVP displays an exception and a stack trace.

4. Select **Bean managed** and then click **Submit**.

   If WebSphere Application Server is configured properly, the following information is displayed:

   ```
   Result: IVP successful for the bean managed EJB.
   ```

   If WebSphere Application Server is not configured properly, the IVP displays an exception and a stack trace.

   If you successfully run the IVP, IMS Java and WebSphere Application Server are installed and configured properly.

Running the IMS Java IVP for Remote Database Services on WebSphere Application Server V6

Prerequisite: "Configuring the Application Servers for IMS Java Remote Database Services" on page 77

To run the IMS Java IVP for remote database services:

- "Setting the WebSphere Application Server V6 for z/OS Classpath"
- "Installing the Data Source for the IVP on the Client Side"
- "Installing the IVP on the Client Side" on page 89
- "Testing the IVP on WebSphere Application Server V6" on page 90

Next: "Running the IMS Java Sample Applications for Remote Database Services" on page 91

Setting the WebSphere Application Server V6 for z/OS Classpath

You must set the WebSphere Application Server for z/OS classpath to the location of the IMS Java metadata class, which is in the file samples.jar.

One way to set the classpath is to add samples.jar to the IMS JDBC resource adapter classpath.

To add the samples.jar file to the IMS JDBC resource adapter classpath:

1. From the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the name of the IMS JDBC resource adapter.

   A configuration dialog is displayed.
3. In the **Classpath** field, add *pathprefix*`/usr/lpp/ims/imsjava91/samples/samples.jar`. Do not delete imsjava.jar.
4. Click **OK**.
5. In the messages box, click **Save**.

   The save page is displayed.
6. Click **Save** to update the master repository with your changes.
7. Restart the server.

Installing the Data Source for the IVP on the Client Side

To install the data source on the client side:

1. In the left frame of the WebSphere Application Server administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the name of IMS JDBC resource adapter that you chose when you installed the adapter.

   A configuration dialog is displayed.
3. Under Additional Properties, click **J2C connection factories**.
4. Click **New**.

   A configuration dialog is displayed.
5. Type the following information:

   Name: name for the data source

JNDI Name: `imsjavaRDSIVP`

6. Click **OK**.

   The data source is listed in the J2C Connection Factories.
7. Click the name of the data source that you installed in step 5.
8. Under Additional Properties, click **Custom properties**.

   Six properties are listed in a table.
9. Click **DatabaseViewName**.

   A configuration dialog is displayed.
10. In the **Value** field, type: `samples.ivp.DFSIVP37DatabaseView`
11. Click **OK**.

    The properties table displays the DatabaseViewName value that you just entered.
12. Click **DRAName**.

    A configuration dialog is displayed.
13. In the **Value** field, type bytes 4-7 of the DRA startup table module name (usually the IMS system ID). For more information about the DRA startup table, see “Configuring WebSphere Application Server V6 for z/OS to Access IMS” on page 42.
14. Click **OK**.

    The properties table displays the DRA name that you just entered.
15. Click **HostName**.

    A configuration dialog is displayed.
16. In the **Value** field, type the name of the host system.
17. Click **OK**.

    The properties table displays the host name that you just entered.
18. Click **PortNumber**.

    A configuration dialog is displayed.
19. In the **Value** field, type the port number of the host system.
20. Click **OK**.

    The properties table displays the port number that you just entered.
21. In the messages box, click **Save**.

    The save page is displayed.
22. Click **Save** to update the master repository with your changes.
23. Restart the server.
24. Click **Applications**, and then click **Enterprise Application**.

    The application IMSJavaRDS IVP is listed with a red X, which indicates that the application is stopped.
25. Select **IMSJavaRDS IVP** and click **Start**.

    The application IMSJavaRDS IVP is listed with a green arrow, which indicates that the application is started.

Next: “Installing the IMS Java IVP on WebSphere Application Server V6 for z/OS” on page 50

Installing the IVP on the Client Side

This section describes how to deploy an application on WebSphere Application Server on a non-z/OS platform.

Prerequisite: “Installing the Data Source for the IVP on the Client Side” on page 88

To install the IMS Java IVP:

1. From the WebSphere Application Server administrative console, click **Applications**, and then click **Install New Application**.

   A dialog for installing the application is displayed.
2. Select **Local file system** and type the path to IMSJavaRDSIVP.ear.
3. Click **Next**.
4. Accept the defaults and click **Next**.

   The Install New Application wizard is started.
5. Click **Step 7** to accept the installation defaults. Depending on your specific server configuration, you might have to use the wizard to change some default values.
6. Verify that the options are correct, and then click **Finish**.

   A message is displayed that indicates first that the application is being installed, and then that the installation was successful.
7. Click **Save to Master Configuration**.

   The save page is displayed.
8. Click **Save** to update the master repository with your changes.
9. Click **Applications**, and then click **Enterprise Application**.

   The application IMSJavaRDS IVP is listed with a red X, which indicates that the application is stopped.
10. Select **IMSJavaRDS IVP** and click **Start**.

    The application IMSJavaRDS IVP is listed with a green arrow, which indicates that the application is started.

Next: “Testing the IVP on WebSphere Application Server V6”

Testing the IVP on WebSphere Application Server V6

This section describes how to test the IVP on WebSphere Application Server on a non-z/OS platform. The IVP tests both a container-managed EJB and a bean-managed EJB.

Prerequisite: “Installing the IVP on the Client Side” on page 89

To test the IVP:

1. Open a Web browser.
2. Type the Web address of the IVP:

   `http://host_IP_address:port/IMSJavaRDSIVPWeb/IMSJavaRDSIVP.html`

   An input Web page opens titled IMS Java IVP for Remote Database Services.
3. Select **Container managed** and then click **Submit**.

   If WebSphere Application Server is configured properly, the following information is displayed:

   ```
   Result: IVP successful for the container managed EJB.
   ```

   If WebSphere Application Server is not configured properly, the IVP displays an exception and a stack trace.
4. Select **Bean managed** and then click **Submit**.

   If WebSphere Application Server is configured properly, the following information is displayed:

```
Result: IVP successful for the bean managed EJB.
```

If WebSphere Application Server is not configured properly, the IVP displays an exception and a stack trace.

If you successfully run the IVP, IMS Java and WebSphere Application Server are installed and configured properly.

Running the IMS Java Sample Applications for Remote Database Services

This topic describes the high-level tasks that you must complete, along with the IMS-specific information required, to run the IMS Java sample applications for remote database services.

To run the IMS Java sample application on WebSphere Application Server:

1. Set the WebSphere Application Server for z/OS classpath to the location of the IMS Java metadata class, which is in the file samples.jar.
2. Install the data source for the IMS Java sample application as a J2C connection factory on WebSphere Application Server.

   JNDI name:

   - Phonebook sample: imsjavaPhonebook
   - Dealership sample: jdbc/DealershipSample

   Database view name:

   - Phonebook sample: `samples.ivp.DFSIVP37DatabaseView`
   - Dealership sample: `samples.dealership.AUTPSB11DatabaseView`

   Also specify a host name and port number.
3. Install and start the IMS Java remote database services sample application EAR file on WebSphere Application Server.

   EAR file name:

   - Phonebook sample: IMSJavaRDSPhonebook.ear
   - Dealership sample: IMSJavaRDSDealership.ear
4. Test the sample application.

   Sample application URL:

   - Phonebook sample:

     `http://`*`host_IP_address`*`:`*`port`*`/IMSJavaRDSPhonebookWeb/IMSJavaRDSPhonebook.html`
   - Dealership sample:

     `http://`*`host_IP_address`*`:`*`port`*`/IMSJavaRDSDealershipWeb/IMSJavaRDSDealership.html`

   Sample input data:

   - Phonebook sample:

     Last Name: `LAST1`
   - Dealership sample:

     Car Make: `FORD`

     VIN Number: `V234567890123456789V`

Detailed information about running the IMS Java sample applications for remote database services on specific versions of WebSphere Application Server is available:

- “Running the IMS Java Sample Applications on WebSphere Application Server V5” on page 92

- "Running the IMS Java Sample Applications on WebSphere Application Server V6" on page 96

Running the IMS Java Sample Applications on WebSphere Application Server V5

Prerequisite: "Running the IMS Java IVP for Remote Database Services" on page 84

To run the IMS Java sample applications for remote database services:

1. "Setting the WebSphere Application Server V5 for z/OS Classpath"
2. "Installing the Data Source for the IMS Java Samples on the Client Side"
3. "Installing the IMS Java Sample Applications on the Client Side" on page 93
4. "Testing the Phonebook Sample on WebSphere Application Server V5" on page 95 or "Testing the Dealership Sample on WebSphere Application Server V5" on page 95

Setting the WebSphere Application Server V5 for z/OS Classpath

You must set the WebSphere Application Server for z/OS classpath to the location of the IMS Java metadata class, which is in the file samples.jar.

One way to set the classpath is to add samples.jar to the IMS JDBC resource adapter classpath.

To add the samples.jar file to the IMS JDBC resource adapter classpath:

1. From the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the name of the IMS JDBC resource adapter.

   A configuration dialog is displayed.
3. In the **Classpath** field, add *pathprefix*`/usr/lpp/ims/imsjava91/samples/samples.jar`. Do not delete imsjava.jar.
4. Click **OK**.
5. In the messages box, click **Save**.

   The save page is displayed.
6. Click **Save** to update the master repository with your changes.
7. Restart the server.

Installing the Data Source for the IMS Java Samples on the Client Side

To install the data source on the client side:

1. In the left frame of the WebSphere Application Server administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the IMS distributed JDBC resource adapter.

   A configuration dialog is displayed.
3. Under Additional Properties, click **J2C Connection Factories**.
4. Click **New**.

   A configuration dialog is displayed.

5. Type the following information:
   Name: name for the data source
   – For the dealership sample, type: `imsjavaRDSDealership`
   – For the phonebook sample, type: `imsjavaRDSPhonebook`
   JNDI Name: path to the data source
   – For the dealership sample, type: `imsjavaRDSDealership`
   – For the phonebook sample, type: `imsjavaRDSPhonebook`

   Note: To avoid messages J2CA0107I and J2CA0114W, both of which can be ignored, set default values for component-managed authentication alias and container-managed authentication alias.
6. Click **OK**.
   The data source is listed in the J2C Connection Factories.
7. Click the name of the data source that you installed in step 5.
8. Under Additional Properties, click **Custom Properties**.
   Six properties are listed in a table.
9. In the **DRAName** row, click the dash symbol in the **Value** column.
10. In the **Value** field, type bytes 4-7 of the DRA startup table module name (usually the IMS system ID). For more information about the DRA startup table, see “Configuring WebSphere Application Server V6 for z/OS to Access IMS” on page 42.
11. Click **OK**.
    The properties table displays the DRA name that you entered.
12. In the **DatabaseViewName** row, click the dash symbol in the **Value** column.
13. In the **Value** field, type the fully-qualified `DLIDatabaseView` subclass name.
    - For the phonebook sample, type: `samples.ivp.DFSIVP37DatabaseView`
    - For the dealership sample, type: `samples.dealership.AUTPSB11DatabaseView`
14. In the **HostName** row, click the dash symbol in the **Value** column.
15. In the **Value** field, type the name or IP address of the host machine.
16. Click **OK**.
    The properties table displays the host name that you entered.
17. In the **PortNumber** row, click the dash symbol in the **Value** column.
18. In the **Value** field, type the IIOP port number of the host machine’s server. For example: `2809`
19. Click **OK**.
    The properties table displays the port number that you entered.
20. Optionally, set the trace level for the applications. See “Enabling J2EE Tracing with WebSphere Application Server V5” on page 103.
21. Click **Save**.
    The Save page is displayed.
22. Under **Save to Master Configuration**, click **Save** to ensure that the changes are made.

Next: “Installing the IMS Java Sample Applications on the Client Side”

Installing the IMS Java Sample Applications on the Client Side

Prerequisite: “Installing the Data Source for the IMS Java Samples on the Client Side” on page 92

This section describes how to deploy an application on WebSphere Application Server on a non-z/OS platform.

To install the application:

1. From the WebSphere Application Server administrative console, click **Applications**, and then click **Install New Application**.

   A dialog for installing a new application is displayed.
2. Type the path to the EAR file.
   - For the phonebook sample, type the path to IMSJavaRDSPhonebook.ear.
   - For the dealership sample, type the path to IMSJavaRDSDealership.ear.
3. Click **Next**.
4. Accept the defaults and click **Next**.

   An application security warning is displayed. This warning indicates that the phonebook or dealership sample will write trace files to the /tmp directory.
5. Click **Continue**.

   The Install New Application wizard starts. Step 1, ″Provide options to perform the installation,″ is displayed.
6. Clear the **Create MBeans for Resources** check box.
7. Click **Next**.

   Step 2, ″Provide JNDI Names for Beans,″ is displayed.
8. In the **JNDI Name** fields, type the path to the EJB home interface.
   - For the dealership sample, verify that the names are as follows:

     `ejb/samples/dealership/rds/IMSJavaRDSDealershipSessionHome`
   - For the phonebook sample, verify that the names are as follows:
     - `ejb/samples/phonebook/rds/IMSJavaRDSPBStatefulCMSessionHome`
     - `ejb/samples/phonebook/rds/IMSJavaRDSPBStatefulBMTXSessionHome`
     - `ejb/samples/phonebook/rds/IMSJavaRDSPBStatefulBMNoTXSessionHome`
     - `ejb/samples/phonebook/rds/IMSJavaRDSPBStatelessCMSessionHome`
9. Click **Next**.

   Step 3, ″Map resource references to resources,″ is displayed.
10. Verify the name of the JNDI name for the resource references.
    - For the dealership sample, verify that the JNDI name of resource reference of the IMSJavaRDS dSample EJB modules is `imsjavaRDSDealership`.
    - For the phonebook sample, verify that the JNDI name of resource references of the two EJBs within the IMSJavaRDS pbSample EJB modules are both `imsjavaRDSPhonebook`.
11. Click **Next**.

    Step 4, ″Map virtual hosts for web modules,″ is displayed.
12. Accept the defaults and click **Next**.

    Step 6, ″Ensure all unprotected 2.0 methods have the correct level of protection,″ is displayed.
13. Make any necessary changes and click **Next**.

    The options that you specified are displayed in Step 7, ″Summary,″ of the Install New Application wizard.
14. Verify that the options are correct, and then click **Finish**.

    A message is displayed that indicates first that the application is being installed, and then that the installation was successful.

15. Click **Save to Master Configuration**.

    The Save to Master Configuration dialog is displayed.
16. Click **Save**.
17. Restart the server to ensure that the changes have been made to the data source and to start the sample enterprise application.
18. Click **Applications**, and then click **Enterprise Application**.

    The sample application is listed with a red X, which indicates that the application is stopped.
19. Select the sample application and click **Start**.

    The sample application is listed with a green arrow, which indicates that the application is started.

Next: "Testing the Phonebook Sample on WebSphere Application Server V5" or "Testing the Dealership Sample on WebSphere Application Server V5"

Testing the Phonebook Sample on WebSphere Application Server V5

This section describes how to test the phonebook sample on WebSphere Application Server on a non-z/OS platform.

Prerequisite: "Installing the IMS Java Sample Applications on the Client Side" on page 93

To test the phonebook sample:

1. Open a Web browser.
2. Type the Web address of the phonebook sample:

   `http://host_IP_address:port/IMSJavaRDSPhonebookWeb/IMSJavaRDSPhonebook.html`

   An input Web page opens titled IMS Java Phonebook Sample for Remote Database Services.
3. Select the type of EJB to test, such as Stateful, Container managed, and type the following information:

   Last Name: `LAST1`
4. Select **Display an entry** and click **Submit**.

   If WebSphere Application Server is configured properly, the following information is displayed:

   ```
   Person found! Last Name: LAST1
   First Name: FIRST1
   Extension: 8-111-1111 Zip code: D01/R01
   ```
5. Optionally, test other EJB types and commands.

Testing the Dealership Sample on WebSphere Application Server V5

This section describes how to test the IMS Java dealership sample on WebSphere Application Server on a non-z/OS platform.

Prerequisite: "Installing the IMS Java Sample Applications on the Client Side" on page 93

To test the dealership sample:

1. Open a Web browser.
2. Type the Web address of the dealership sample:

`http://host_IP_address:port/IMSJavaRDSDealershipWeb/IMSJavaRDSDealership.html`

An input Web page opens that is titled **Find a car in stock**.

3. Verify that **Car Make** and **VIN Number** fields contain the following information:

```
Car Make: FORD
VIN Number: V234567890123456789V
```

4. Click **Submit**.

   A message indicating that the query was successful is displayed.

5. Click on the query options on the left to test the applications. Submit the queries with the default values or enter your own query values.

Running the IMS Java Sample Applications on WebSphere Application Server V6

Prerequisite: “Running the IMS Java IVP for Remote Database Services” on page 84

To run the IMS Java sample applications for remote database services:

1. “Setting the WebSphere Application Server V6 for z/OS Classpath”
2. “Installing the Data Source for the IMS Java Samples on the Client Side”
3. “Installing the IMS Java Sample Applications on the Client Side” on page 98
4. “Testing the Phonebook Sample on WebSphere Application Server V6” on page 99 or “Testing the Dealership Sample on WebSphere Application Server V6” on page 99

Setting the WebSphere Application Server V6 for z/OS Classpath

You must set the WebSphere Application Server for z/OS classpath to the location of the IMS Java metadata class, which is in the file samples.jar.

One way to set the classpath is to add samples.jar to the IMS JDBC resource adapter classpath.

To add the samples.jar file to the IMS JDBC resource adapter classpath:

1. From the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.

2. Click the name of the IMS JDBC resource adapter.

   A configuration dialog is displayed.

3. In the **Classpath** field, add *pathprefix*`/usr/lpp/ims/imsjava91/samples/samples.jar`. Do not delete imsjava.jar.
4. Click **OK**.
5. In the messages box, click **Save**.

   The save page is displayed.

6. Click **Save** to update the master repository with your changes.
7. Restart the server.

Installing the Data Source for the IMS Java Samples on the Client Side

To install the data source on the client side for the IMS Java samples:

1. In the left frame of the WebSphere Application Server administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the name of IMS JDBC resource adapter that you chose when you installed the adapter.

   A configuration dialog is displayed.
3. Under Additional Properties, click **J2C connection factories**.
4. Click **New**.

   A configuration dialog is displayed.
5. Type the following information:
   Name: name for the data source
   JNDI Name: path to the data source.
   – For the phonebook sample, type: `imsjavaRDSPhonebook`
   – For the dealership sample, type: `imsjavaRDSDealership`
6. Click **OK**.

   The data source is listed in the J2C Connection Factories.
7. Click the name of the data source that you installed in step 5.
8. Under Additional Properties, click **Custom properties**.

   Six properties are listed in a table.
9. Click **DatabaseViewName**.

   A configuration dialog is displayed.
10. In the **Value** field, type the fully-qualified `DLIDatabaseView` subclass name.
    • For the phonebook sample, type: `samples.ivp.DFSIVP37DatabaseView`
    • For the dealership sample, type:`samples.dealership.AUTPSB11DatabaseView`
11. Click **OK**.

    The properties table displays the DatabaseViewName value that you just entered.
12. Click **DRAName**.

    A configuration dialog is displayed.
13. In the **Value** field, type bytes 4-7 of the DRA startup table module name (usually the IMS system ID). For more information about the DRA startup table, see “Configuring WebSphere Application Server V6 for z/OS to Access IMS” on page 42.
14. Click **OK**.

    The properties table displays the DRA name that you just entered.
15. Click **HostName**.

    A configuration dialog is displayed.
16. In the **Value** field, type the name of the host system.
17. Click **OK**.

    The properties table displays the host name that you just entered.
18. Click **PortNumber**.

    A configuration dialog is displayed.
19. In the **Value** field, type the port number of the host system.
20. Click **OK**.

    The properties table displays the port number that you just entered.
21. In the messages box, click **Save**.

    The save page is displayed.

22. Click **Save** to update the master repository with your changes.
23. Restart the server.

Next: "Installing the IMS Java Sample Applications on the Client Side"

Installing the IMS Java Sample Applications on the Client Side

Prerequisite: "Installing the Data Source for the IMS Java Samples on the Client Side" on page 96

This section describes how to deploy an application on WebSphere Application Server on a non-z/OS platform.

To install the sample applications:

1. From the WebSphere Application Server administrative console, click **Applications**, and then click **Install New Application**.

   A dialog for installing a new application is displayed.
2. Select **Local file system** and type the path to the EAR file:
   - For the phonebook sample, type the path to IMSJavaPhonebook.ear.
   - For the dealership sample, type the path to imsjavaDealership.ear.
3. Click **Next**.
4. Accept the defaults and click **Next**.

   Application security warnings are displayed.
5. Click **Continue**.

   The Install New Application wizard is started.
6. Click **Step 3**.

   Step 3, ″Provide JNDI Names for Beans,″ is displayed.
7. In the **JNDI Name** field, type the path to the EJB home interface.
   - For the phonebook sample, verify that the names are as follows:
     - `ejb/samples/phonebook/rds/IMSJavaRDSPBStatefulCMSessionHome`
     - `ejb/samples/phonebook/rds/IMSJavaRDSPBStatefulBMTXSessionHome`
     - `ejb/samples/phonebook/rds/IMSJavaRDSPBStatefulBMNoTXSessionHome`
     - `ejb/samples/phonebook/rds/IMSJavaRDSPBStatelessCMSessionHome`
   - For the dealership sample, verify that the names are as follows:

     `ejb/samples/dealership/rds/IMSJavaRDSDealershipSessionHome`
8. Click **Step 4**.

   Step 4, ″Map resource references to resources,″ is displayed.
9. For the phonebook sample, verify that the JNDI name of resource references of the IMSJavaRDS pbSample EJB module are both imsjavaRDSPhonebook.

   For the dealership sample, verify that the JNDI name of resource references of the IMSJavaRDS dSample EJB module is imsjavaRDSDealership.
10. Click **Step 7**.

    If application resource warnings appear, verify that the resource assignments are correct and click **Continue**.
11. On the summary page, verify that the options are correct, and then click **Finish**.

    A message is displayed that indicates first that the application is being installed, and then that the installation was successful.
12. Click **Save to Master Configuration**.

The save page is displayed.

13. Click **Save** to update the master repository with your changes.
14. Click **Applications**, and then click **Enterprise Application**.

    The sample application is listed with a red X, which indicates that the application is stopped.
15. Select the sample application and click **Start**.

    The sample application is listed with a green arrow, which indicates that the application is started.

Next: "Testing the Phonebook Sample on WebSphere Application Server V6" or "Testing the Dealership Sample on WebSphere Application Server V6"

Testing the Phonebook Sample on WebSphere Application Server V6

This section describes how to test the phonebook sample on WebSphere Application Server on a non-z/OS platform.

Prerequisite: "Installing the IMS Java Sample Applications on the Client Side" on page 98

To test the phonebook sample:

1. Open a Web browser.
2. Type the Web address of the phonebook sample:

   `http://host_IP_address:port/IMSJavaRDSPhonebookWeb/IMSJavaRDSPhonebook.html`

   An input Web page opens titled IMS Java Phonebook Sample for Remote Database Services.
3. Select the type of EJB to test, such as Stateful, Container managed, and type the following information:

   Last Name: `LAST1`
4. Select **Display an entry** and click **Submit**.

   If WebSphere Application Server is configured properly, the following information is displayed:

   ```
   Person found! Last Name: LAST1
   First Name: FIRST1
   Extension: 8-111-1111 Zip code: D01/R01
   ```
5. Optionally, test other EJB types and commands.

Testing the Dealership Sample on WebSphere Application Server V6

This section describes how to test the IMS Java dealership sample on WebSphere Application Server on a non-z/OS platform.

Prerequisite: "Installing the IMS Java Sample Applications on the Client Side" on page 98

To test the dealership sample:

1. Open a Web browser.
2. Type the Web address of the dealership sample:

   `http://host_IP_address:port/IMSJavaRDSDealershipWeb/IMSJavaRDSDealership.html`

   An input Web page opens that is titled **Find a car in stock**.
3. Verify that **Car Make** and **VIN Number** fields contain the following information:

```
Car Make: FORD
VIN Number: V234567890123456789V
```

4. Click **Submit**.

   A message indicating that the query was successful is displayed.
5. Click on the query options on the left to test the applications. Submit the queries with the default values or enter your own query values.

Running Your Application on WebSphere Application Server

This topic provides the high-level steps that are required to run an application that accesses IMS DB from WebSphere Application Server.

To run your application on WebSphere Application Server:

1. Set the WebSphere Application Server classpath to point to the IMS Java metadata class.
2. Install the data source for the application as a J2C connection factory.
3. Install and start the application.

Detailed information about running an application on specific versions of WebSphere Application Server is also provided:

- "Running Your Application on WebSphere Application Server V5"
- "Running Your Application on WebSphere Application Server V6" on page 104

Running Your Application on WebSphere Application Server V5

Prerequisite: "Running the IMS Java IVP for Remote Database Services" on page 84

To deploy your own application:

- "Setting the WebSphere Application Server V5 for z/OS Classpath"
- "Installing the Data Source on the Client Side" on page 101
- "Installing the Application on the Client Side" on page 102
- "Adding the XML Files to the EAR Classpath" on page 80

Setting the WebSphere Application Server V5 for z/OS Classpath

You must set the WebSphere Application Server for z/OS classpath to the location of the IMS Java metadata class (`DLIDatabaseView` subclass) that is used by the application.

One way to set the classpath is to add these files to the IMS JDBC resource adapter classpath.

To add the required files to the IMS JDBC resource adapter classpath:

1. From the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the name of the IMS JDBC resource adapter.

   A configuration dialog is displayed.
3. In the **Classpath** field, add the path to the required files. Include the file name for JAR files. Do not delete imsjava.jar.
4. Click **OK**.

Installing the Data Source on the Client Side

To install the data source on the client side:

1. In the left frame of the WebSphere Application Server administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the IMS distributed JDBC resource adapter.

   A configuration dialog is displayed.
3. Under Additional Properties, click **J2C Connection Factories**.
4. Click **New**.

   A configuration dialog is displayed.
5. Type the following information:
   Name: the name for the data source
   JNDI Name: the path to the data source

   Note: To avoid messages J2CA0107I and J2CA0114W, both of which can be ignored, set default values for component-managed authentication alias and container-managed authentication alias.
6. Click **OK**.

   The data source is listed in the J2C Connection Factories.
7. Click the name of the data source that you installed in step 5.
8. Under Additional Properties, click **Custom Properties**.

   Six properties are listed in a table.
9. In the **DRAName** row, click the dash symbol in the **Value** column.
10. In the **Value** field, type bytes 4-7 of the DRA startup table module name (usually the IMS system ID). For more information about the DRA startup table, see “Configuring WebSphere Application Server V6 for z/OS to Access IMS” on page 42.
11. Click **OK**.

    The properties table displays the DRA name that you entered.
12. In the **DatabaseViewName** row, click the dash symbol in the **Value** column.
13. Optional: In the **Value** field, type the fully-qualified `DLIDatabaseView` subclass name.

    If you do set the subclass name, you must either create a data source for every PSB an application accesses, or you must override the `DLIDatabaseView` subclass name in the `DataSource` object by calling the `setDatabaseView` method and providing the fully-qualified name of the subclass.

    If you do not set the subclass name, you need to create a data source only for each IMS. In the application, define the `DLIDatabaseView` subclass name in the `DataSource` object by calling the `setDatabaseView` method and providing the fully-qualified name of the subclass.
14. In the **HostName** row, click the dash symbol in the **Value** column.
15. In the **Value** field, type the name or IP address of the host machine.
16. Click **OK**.

    The properties table displays the host name that you entered.
17. In the **PortNumber** row, click the dash symbol in the **Value** column.
18. In the **Value** field, type the IIOP port number of the host machine’s server. For example: `2809`
19. Click **OK**.

    The properties table displays the port number that you entered.

20. Optionally, set the trace level for the applications. See “Enabling J2EE Tracing with WebSphere Application Server V5” on page 103.
21. Click **Save**.

    The Save page is displayed.
22. Under **Save to Master Configuration**, click **Save** to ensure that the changes are made.

Next: “Installing the Application on the Client Side” on page 106

Installing the Application on the Client Side

Prerequisite: “Installing the Data Source on the Client Side” on page 101

This section describes how to deploy an application on WebSphere Application Server on a non-z/OS platform.

To install the application:

1. From the WebSphere Application Server administrative console, click **Applications**, and then click **Install New Application**.

   A dialog for installing a new application is displayed.
2. Type the path to the EAR file.
3. Click **Next**.
4. Accept the defaults and click **Next**.

   The Install New Application wizard starts. Step 1, ″Provide options to perform the installation,″ is displayed.
5. Clear the **Create MBeans for Resources** check box.
6. Click **Next**.

   Step 2, ″Provide JNDI Names for Beans,″ is displayed.
7. In the **JNDI Name** fields, type the path to the EJB home interface.
8. Click **Next**.

   Step 3, ″Map resource references to resources,″ is displayed.
9. For the module that you want to install, type the JNDI name.
10. Click **Next**.

    Step 4, ″Map virtual hosts for web modules,″ is displayed.
11. Accept the defaults and click **Next**.

    Step 5, ″Map modules to application servers,″ is displayed.
12. Accept the defaults and click **Next**.

    Step 6, ″Ensure all unprotected 2.0 methods have the correct level of protection,″ is displayed.
13. Make any necessary changes and click **Next**.

    The options that you specified are displayed in Step 7, ″Summary,″ of the Install New Application wizard.
14. Verify that the options are correct, and then click **Finish**.

    A message is displayed that indicates first that the application is being installed, and then that the installation was successful.
15. Click **Save to Master Configuration**.

    The Save to Master Configuration dialog is displayed.
16. Click **Save**.
17. Click **Applications**, and then click **Enterprise Application**.

The application is listed with a red X, which indicates that the application is stopped.

18. Select the application and click **Start**.

    The application is listed with a green arrow, which indicates that the application is started.

Enabling J2EE Tracing with WebSphere Application Server V5

You can trace the IMS library classes by using the WebSphere Application Server tracing service.

To enable tracing if you have not yet specified the level of tracing:

1. "Specifying the Level of Tracing"
2. "Specifying the Application Server and the Package to Trace"

To enable tracing if you have already specified the level of tracing:

1. "Specifying at Runtime the Application Server and the Package to Trace" on page 104

Specifying the Level of Tracing: To use the WebSphere Application Server tracing service, you must first specify the level of tracing.

To specify the level of tracing:

1. In the left frame of the WebSphere Application Server administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click **IMS distributed JDBC resource adapter**.

   A configuration dialog is displayed.
3. Under Additional Properties, click **J2C Connection Factories**.

   A list of connection factories is displayed.
4. Click the name of the J2C connection factory for which you want to enable tracing.

   A configuration dialog is displayed.
5. Under Additional Properties, click **Custom Properties**.

   Properties are listed in a table.
6. In the **Trace Level** row, click the number in the **Value** column.
7. In the **Value** field, type the trace level.
8. Click **OK**.

   The properties table displays the trace level that you entered.
9. Click **Save**.

   The Save page is displayed.
10. Under **Save to Master Configuration**, click **Save** to ensure that the changes are made.

Specifying the Application Server and the Package to Trace: After you specify the level of tracing, specify the application server and package to trace and then restart the server.

To specify the application server and the package to trace:

1. In the left frame of the WebSphere Application Server administrative console, click **Servers**, and then click **Application Servers**.

A list of application servers is displayed.
2. Click the name of the server on which you want to enable tracing.
3. Under Additional Properties, click **Diagnostic Trace Service**.
   A list of custom services is displayed.
4. Click **New**.
   A configuration dialog for Diagnostic Trace Service is displayed.
5. Select the **Enable Trace** check box.
6. In the Trace Specification field, add: `com.ibm.ims.rds.*=all=enabled`
7. Click **New**.
8. Click **Save**.
   The Save page is displayed.
9. Under **Save to Master Configuration**, click **Save** to ensure that the changes are made.
10. Restart the server.

Specifying at Runtime the Application Server and the Package to Trace: You can turn tracing on and off by specifying at runtime the server and package to trace. You do not need to restart your server each time.

To specify the application server and the package to trace at runtime:

1. In the left frame of the WebSphere Application Server administrative console, click **Servers**, and then click **Application Servers**.
   A list of application servers is displayed.
2. Click the name of the server on which you want to enable tracing.
3. Under Additional Properties, click **Diagnostic Trace Service**.
   A configuration dialog for Diagnostic Trace Service is displayed.
4. Click the **Runtime** tab.
5. In the **Trace Specification** field after any other traces that are listed, type: `com.ibm.ims.rds.*=all=enabled`
6. Click **Apply**.

Running Your Application on WebSphere Application Server V6

Prerequisite: “Running the IMS Java IVP for Remote Database Services” on page 84

To deploy your own application:

- “Setting the WebSphere Application Server V6 for z/OS Classpath”
- “Installing the Data Source on the Client Side” on page 105
- “Installing the Application on the Client Side” on page 106
- “Adding the XML Files to the EAR Classpath” on page 80

Setting the WebSphere Application Server V6 for z/OS Classpath

You must set the WebSphere Application Server for z/OS classpath to the location of the IMS Java metadata class (`DLIDatabaseView` subclass) that is used by the application.

One way to set the classpath is to add these files to the IMS JDBC resource adapter classpath.

To add the required files to the IMS JDBC resource adapter classpath:

1. From the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the name of the IMS JDBC resource adapter.

   A configuration dialog is displayed.
3. In the **Classpath** field, add the path to the required files. Include the file name for JAR files. Do not delete imsjava.jar.
4. Click **OK**.

Installing the Data Source on the Client Side

To install the data source for your application:

1. In the left frame of the WebSphere Application Server for z/OS administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.
2. Click the name of the IMS JDBC resource adapter that you chose when you installed the adapter.

   A configuration dialog is displayed.
3. Under Additional Properties, click **J2C connection factories**.
4. Click **New**.

   A configuration dialog is displayed.
5. Type the following information:
   Name: name for the data source
   JNDI Name: the JNDI name for the data source.
6. Click **OK**.

   The data source is listed in the J2C Connection Factories.
7. Click the name of the data source that you installed in step 5.
8. Under Additional Properties, click **Custom Properties**.

   Six properties are listed in a table.
9. Click **DatabaseViewName**.

   A configuration dialog is displayed.
10. Optional: In the **Value** field, type the fully-qualified `DLIDatabaseView` subclass name.

    If you do set the subclass name, you must either create a data source for every PSB an application accesses, or you must override the `DLIDatabaseView` subclass name in the `DataSource` object that is within the application by calling the `setDatabaseView` method and providing the fully-qualified name of the subclass.

    If you do not set the subclass name, you need to create a data source only for each IMS. In the application, define the `DLIDatabaseView` subclass name in the `DataSource` object by calling the `setDatabaseView` method and providing the fully-qualified name of the subclass.
11. Click **OK**.

    The properties table displays the DatabaseViewName value that you just entered.
12. Click **DRAName**.

    A configuration dialog is displayed.

13. In the **Value** field, type bytes 4-7 of the DRA startup table module name (usually the IMS system ID). For more information about the DRA startup table, see “Configuring WebSphere Application Server V6 for z/OS to Access IMS” on page 42.
14. Click **OK**.

    The properties table displays the DRA name that you just entered.
15. Click **HostName**.

    A configuration dialog is displayed.
16. In the **Value** field, type the name of the host system.
17. Click **OK**.

    The properties table displays the host name that you just entered.
18. Click **PortNumber**.

    A configuration dialog is displayed.
19. In the **Value** field, type the port number of the host system.
20. Click **OK**.

    The properties table displays the port number that you just entered.
21. In the messages box, click **Save**.

    The save page is displayed.
22. Click **Save** to update the master repository with your changes.
23. Click **Applications**, and then click **Enterprise Application**.

    The application is listed with a red X, which indicates that the application is stopped.
24. Select the application and click **Start**.

    The application is listed with a green arrow, which indicates that the application is started.

Next: “Installing the Application on the Client Side”

Installing the Application on the Client Side

Prerequisite: “Installing the Data Source on the Client Side” on page 105

This section describes how to deploy an application on WebSphere Application Server on a non-z/OS platform.

To install your application:

1. From the WebSphere Application Server for z/OS administrative console, click **Applications**, and then click **Install New Application**.

   A dialog for installing a new application is displayed.
2. Type the path to the EAR file.
3. Click **Next**.
4. Accept the defaults and click **Next**.

   The Install New Application wizard is started. Step 1, ″Provide options to perform the installation,″ is displayed.
5. Click **Step 4: Provide JNDI Names for Beans**.
6. In the **JNDI Name** field, type the path to the EJB home interface.
7. Click **Step 7: Summary**.

   The options that you specified are displayed.
8. Verify that the options are correct, and then click **Finish**.

A message is displayed that indicates first that the application is being installed, and then that the installation was successful.

9. Click **Save to Master Configuration**.

   The Save page is displayed.

10. Under **Save to Master Configuration**, click **Save**.
11. Restart the server to ensure that the changes have been made.

Enabling J2EE Tracing with WebSphere Application Server V6

You can trace the IMS library classes by using the WebSphere Application Server tracing service.

To enable tracing if you have not yet specified the level of tracing:

1. "Specifying the Level of Tracing"
2. "Specifying the Application Server and the Package to Trace" on page 108

To enable tracing if you have already specified the level of tracing:

1. "Specifying at Runtime the Application Server and the Package to Trace" on page 108

You can also trace the IMS library classes or your applications using the `com.ibm.ims.base.XMLTrace` class. The `XMLTrace` class is an IMS Java-provided class that represents the trace as an XML document. You can trace different levels of the code depending on the trace level. For more information, see the IMS Java API Specification.

Specifying the Level of Tracing: To use the WebSphere Application Server tracing service, you must first specify the level of tracing.

To specify the level of tracing:

1. In the left frame of the WebSphere Application Server administrative console, click **Resources**, and then click **Resource Adapters**.

   A list of resource adapters is displayed.

2. Click the name of the IMS JDBC resource adapter.

   A configuration dialog is displayed.

3. Under Additional Properties, click **J2C connection factories**.

   A list of connection factories is displayed.

4. Click the name of the J2C connection factory for which you want to enable tracing.

   A configuration dialog is displayed.

5. Under Additional Properties, click **Custom Properties**.

   Properties are listed in a table.

6. Click **TraceLevel** row.
7. In the **Value** field, type the trace level.
8. Click **OK**.

   The properties table displays the trace level that you just entered.

9. In the messages box, click **Save**.

   The save page is displayed.

10. Click **Save** to update the master repository with your changes.

Specifying the Application Server and the Package to Trace: After you specify the level of tracing, specify the application server and package to trace and then restart the server.

To specify the application server and the package to trace:

1. In the left frame of the WebSphere Application Server administrative console, click **Servers**, and then click **Application Servers**.
   A list of application servers is displayed.
2. Click the name of the server on which you want to enable tracing.
3. Under Troubleshooting, click **Diagnostic Trace Service**.
   A configuration dialog for Diagnostic Trace Service is displayed.
4. Select the **Enable Log** check box and click **OK**.
5. Under Troubleshooting, click **Change Log Detail Levels**.
6. Click the plus sign (+) next to **com.ibm.ims.***.
7. Click **com.ibm.ims.rds.***
8. From the list of trace detail levels, click **all**.
9. Verify that `com.ibm.ims.rds*=all` appears in the text box and click **OK**.
10. In the messages box, click **Save**.
    The save page is displayed.
11. Click **Save** to update the master repository with your changes.
12. Restart the server.

Specifying at Runtime the Application Server and the Package to Trace: You can turn tracing on and off by specifying at runtime the server and package to trace. You do not need to restart your server each time.

To specify the application server and the package to trace at runtime:

1. In the left frame of the WebSphere Application Server administrative console, click **Servers**, and then click **Application Servers**.
   A list of application servers is displayed.
2. Click the name of the server on which you want to enable tracing.
3. Under Troubleshooting, click **Change Log Detail Levels**.
   A configuration dialog for Change Log Detail Levels is displayed.
4. Click the **Runtime** tab.
5. Click the plus sign (+) next to **com.ibm.ims.rds**.
6. Click **com.ibm.ims.rds***
7. From the list of trace detail levels, click **all**.
8. Verify that `com.ibm.ims.rds*=all` appears in the text box and click **OK**.
9. Click **OK**.

WebSphere Application Server EJBs

When you design EJBs that access IMS, there are three IMS-specific considerations:

- Transaction semantics and how that affects commits and rollbacks (“Transaction Semantics and Server-Side EJB Types” on page 109).
- Security semantics and how that affects security identity and application access (“Client-Side EJB Security Semantics” on page 109).

- IMS Java JDBC implementation and how that affects SQL calls (Chapter 7, “JDBC Access to IMS Data,” on page 125).

Restriction: The IMS distributed JDBC resource adapter does not support shared connections.

Transaction Semantics and Server-Side EJB Types

There are two server-side EJB types: container managed and bean managed. These EJBs act as listeners for distributed requests that come from the IMS distributed JDBC resource adapter. The type of EJB that is created depends on whether there is a transaction context when the client-side EJB makes the first SQL call. Applications do not manage these EJBs because they are created and managed by the IMS distributed JDBC resource adapter.

When a client-side EJB executes the first SQL request to a database, the IMS distributed JDBC resource adapter checks to see if there is a transaction started. If there is a transaction context, global transaction semantics are followed. However, if there is no transaction context, then local transaction semantics are followed.

If there is a transaction context on the client side, the IMS distributed JDBC resource adapter creates a container-managed EJB on the server side that joins the existing transaction. Global transaction semantics are followed, meaning that if the client-side application is container-managed, the container commits and rolls back work, and if the client-side EJB is bean managed, the application commits and rolls back work with the `UserTransaction` class. All work is committed and rolled back.

If there is no transaction context on the client side, the IMS distributed JDBC resource adapter starts a bean-managed EJB on the server side, which starts a transaction for each connection. Local transaction semantics are followed meaning that the client application can commit and roll back individual connections using the `java.sql.Connection` object.

Table 2 summarizes the relationship between the transaction context and the transaction semantics.

Table 2. Relationship between the Transaction Context and the Transaction Semantics

Transaction Context?	Server-Side EJB Transaction Type	Transaction Semantics	Transaction Boundary Delimiter
Yes	Container managed	Global	EJB container or `javax.transaction.UserTransaction`
No	Bean managed	Local	`java.sql.Connection`

Related Reading: For more information about transaction contexts, see the *Java Transaction Service (JTA) Specification* and the *Java Transaction API (JTA) Specification*.

Client-Side EJB Security Semantics

There are three areas to consider for client-side EJB security:

- **Access to client-side EJB:** Deploy the client-side EJB with the run-as deployment property set to `system`. Restrict access to the client-side EJB. For information about run-as options and other security issues, see the WebSphere Application Server information center.

- **Network security:** You can use identity assertion or SSL to secure the network communication between the two application servers.
- **Security between WebSphere Application Server for z/OS and IMS:** ODBA requires a pre-verified ACEE (access control environment element), which WebSphere Application Server for z/OS places on the execution thread.

Chapter 5. DB2 UDB for z/OS Stored Procedures

You can write DB2 UDB for z/OS Java stored procedures that access IMS databases.

To deploy a Java stored procedure on DB2 UDB for z/OS, you must configure IMS Java, ODBA, and DRA.

Figure 15 shows a DB2 UDB for z/OS stored procedure using IMS Java, ODBA, and DRA to access IMS databases.

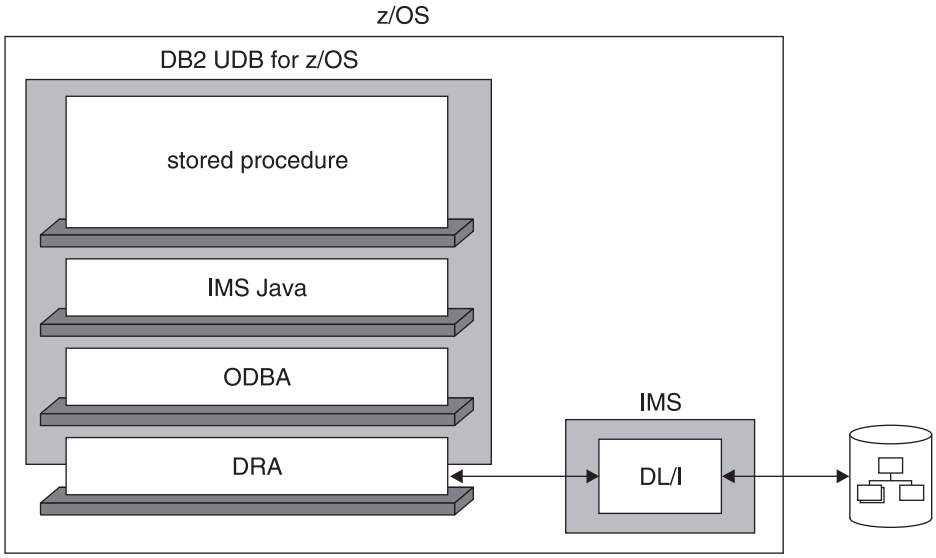


Figure 15. DB2 UDB for z/OS Stored Procedure Using IMS Java

The following topics provide additional information:

- "Configuring DB2 UDB for z/OS for IMS Java"
- "Running the IMS Java IVP from DB2 UDB for z/OS" on page 113
- "Running the IMS Java Sample Application on DB2 UDB for z/OS" on page 115
- "Running Your Stored Procedure from DB2 UDB for z/OS" on page 116
- "Developing DB2 UDB for z/OS Stored Procedures that Access IMS DB" on page 118

Configuring DB2 UDB for z/OS for IMS Java

Access to IMS databases from DB2 UDB for z/OS stored procedures requires IBM DB2 Universal Database for z/OS and OS/390 Version 7 with APARs PQ46673 and PQ50443. You also must have the DB2 for OS/390 and z/OS SQLJ/JDBC driver with APAR PQ48383 installed or the DB2 Universal JDBC Driver.

Prerequisite: "Installing IMS Java" on page 2

To configure DB2 UDB for z/OS for IMS Java:

1. Create a data set with the following attributes. This data set is the JAVAENV DD statement data set.
   - Organization: PS

- Record format: VB
- Record length: 1028
- Block size: 6144

2. In the data set that you created in step 1 on page 111, add the ENVAR keyword with following parameters:

   JAVA_HOME=
   The HFS directory of the JVM.

   DB2_HOME=
   The HFS directory of the JDBC driver for DB2 UDB for z/OS.

   CLASSPATH=
   The HFS directories of the client application Java class files. You do not specify the CLASSPATH= if you specify the client application Java class files in the stored procedure definition.

   LIBPATH=
   The HFS directory of the file libJavTDLI.so.

   TMSUFFIX=
   The HFS directories of the IMS Java and XML class libraries:

   ```
   TMPREFIX=pathprefix/usr/lpp/ims/imsjava91/imsjava.jar
   :pathprefix/usr/lpp/ims/imsjava91/lib/xalan.jar
   :pathprefix/usr/lpp/ims/imsjava91/lib/xml-apis.jar
   :pathprefix/usr/lpp/ims/imsjava91/lib/xercesImpl.jar
   ```

   Figure 16 shows a sample JAVAENV data set.

```
ENVAR("CLASSPATH=/usr/lpp/ims/imsjava91/samples.jar",
      "DB2_HOME=/usr/lpp/db2/db27",
      "JAVA_HOME=/usr/lpp/J1.3",
      "LIBPATH=/usr/lpp/ims/imsjava91",
      "TMSUFFIX=/usr/lpp/ims/imsjava91/imsjava.jar
               :/usr/lpp/ims/imsjava91/lib/xalan.jar
               :/usr/lpp/ims/imsjava91/lib/xml-apis.jar
               :/usr/lpp/ims/imsjava91/lib/xercesImpl.jar")
```

Figure 16. Sample JAVAENV Data Set

3. Set DB2 UDB for z/OS environment variables in UNIX System Services by issuing the following commands:

```
export SQLJ_HOME=location of DB2 SQLJ driver (for example /usr/lpp/db2/db2710)
export JDBC_HOME=location of DB2 JDBC driver (for example /usr/lpp/db2/db2710)
export JAVA_HOME=location of SDK1.3 (for example /usr/lpp/java/J1.3)

export DB2SQLJPROPERTIES=/path/db2sqljjdbc.properties (file created later)

export CLASSPATH=$JDBC_HOME/classes/db2jdbcclasses.zip
export CLASSPATH=$CLASSPATH:$SQLJ_HOME/classes/db2sqljruntime.zip
export CLASSPATH=$CLASSPATH:$SQLJ_HOME/classes/db2sqljclasses.zip
export CLASSPATH=$CLASSPATH:pathprefix/usr/lpp/ims/imsjava91/imsjava.jar
export CLASSPATH=$CLASSPATH:$JAVA_HOME/lib:.

export LIBPATH=$SQLJ_HOME/lib:$JDBC_HOME/lib
export LIBPATH=:$JAVA_HOME/lib:$LIBPATH

export LD_LIBRARY_PATH=.:$SQLJ_HOME/bin:$JDBC_HOME/lib
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$JAVA_HOME/lib

export PATH=$SQLJ_HOME/bin:$PATH
```

```
export STEPLIB=yourDB2HLQ.DSNEXIT:yourDB2HLQ.SDSNLOAD:
export STEPLIB=yourDB2HLQ.SDSNLOD2:yourDB2HLQ.SDSNLINK:$STEPLIB
```

4. If you are using SDK 1.4.1, which does not have the required version of Xalan, you must add the JVM environment variable java.endorsed.dirs and set it to the location of the required XML files (for example, `java.endorsed.dirs=`*`pathprefix`*`/usr/lpp/ims/imsjava91/lib`). IMS Java requires Xalan-Java 2.6.0 or later (or equivalent code function).

Next: "Running the IMS Java IVP from DB2 UDB for z/OS"

Running the IMS Java IVP from DB2 UDB for z/OS

To verify that DB2 UDB for z/OS is configured correctly and that IMS Java is installed correctly, run the IMS Java installation verification program (IVP) on DB2 UDB for z/OS.

The IMS Java IVP for DB2 UDB for z/OS is two programs:

- The Java application DB2IvpClient, which runs under UNIX System Services
- The stored procedure DB2IvpStoredProcedure, which runs in a WLM-managed address space.

Prerequisites:

- "Configuring DB2 UDB for z/OS for IMS Java" on page 111
- Ensure that the standard IMS IVPs have been run. These IVPs prepare the DBD for the IVP database, named IVPDB2, and load the IVP database. They also prepare the IMS Java application PSB (named DFSIVP37), build ACBs, and prepare other IMS control blocks required by the IMS Java IVPs. For details of how to run the IMS IVPs, see *IMS Version 9: Installation Volume 1: Installation Verification*.

To run the IMS Java IVP on DB2 UDB for z/OS:

1. In the JAVAENV data set, modify the CLASSPATH= parameter to *pathprefix*/usr/lpp/ims/imsjava91/samples.jar.
2. Edit the IMS-provided V71AWLM procedure as follows (if IMS.SDFSRESL does not contain the DRA startup table, add that data set to the DFSRESLB DD statement):

```
//V71AWLM PROC RGN=0M,APPLENV=,
//             DB2SSN=,NUMTCB=
//* Define the V71AWLM procedure parameters here on in the service policy.
//IEFPROC EXEC PGM=DSNX9WLM,REGION=&RGN,TIME=NOLIMIT,
//        PARM='&DB2SSN,&NUMTCB,&APPLENV'
//STEPLIB  DD  DISP=SHR,DSN=CEE.SCEERUN
//* DB2 Library
//         DD  DISP=SHR,DSN=yourDB2HLQ.SDSNLOAD
//         DD  DISP=SHR,DSN=yourDB2HLQ.SDSNLOD2
//         DD  DISP=SHR,DSN=yourDB2HLQ.RUNLIB.LOAD
//* DBRM library
//         DD  DISP=SHR,DSN=yourHLQ.SDSNDBRM
//DFSRESLB DD  DISP=SHR,DSN=IMS.SDFSRESL
//JAVAENV  DD  DISP=SHR,DSN=data set with ENVAR settings
//JSPDEBUG DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
```

3. Create a new service policy for WLM. You can define the V71AWLM procedure parameters in the service policy or you can modify the procedure itself.
4. Define the procedure V71AWLM to RACF.
5. Start DB2 UDB for z/OS, the WLM-managed address space, and IMS DB.
6. Define the stored procedure to DB2 UDB for z/OS by running the following job (*Your_WLM_Environment_Name* must match the APPLENV= parameter of the V71AWLM procedure):

```
//CREATIVP JOB ,'name',
//         MSGCLASS=H,TIME=3,
//         USER=SYSADM,PASSWORD=XXXXXXXX,
//         MSGLEVEL=(1)
//CREATJSP EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD  DISP=SHR,DSN=DB2HLQ.DSNEXIT
//         DD  DISP=SHR,DSN=DB2HLQ.SDSNLOAD
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DB2_Subsystem_Name)
  RUN PROGRAM(DSNTIAD) PLAN(DSNTIA71) -
  LIB('DB2HLQ.RUNLIB.LOAD') -
  PARM('RC0')
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSIN    DD  *
  CREATE PROCEDURE IVPStoredProc(VARCHAR (50) IN, VARCHAR(800) OUT)
  FENCED
  NO SQL
  LANGUAGE JAVA
  DYNAMIC RESULT SET 0
  EXTERNAL NAME 'samples.ivp.db2.DB2IvpStoredProcedure.execute'
  PARAMETER STYLE JAVA
  COLLID DSNJDBC
  WLM ENVIRONMENT Your_WLM_Environment_Name;
  GRANT EXECUTE ON PROCEDURE IVPStoredProc TO PUBLIC;
```

7. Create a DB2 plan that runs the client program by running the following job:

```
//BNDIVPCL JOB ,'YOUR NAME',
//         MSGCLASS=H,TIME=3,
//         USER=SYSADM,PASSWORD=XXXXXXXX,
//         MSGLEVEL=(1)
//BINDCLNT EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD  DISP=SHR,DSN=DB2HLQ.DSNEXIT
//         DD  DISP=SHR,DSN=DB2HLQ.SDSNLOAD
//DBRMLIB  DD   DISP=SHR,DSN=DB2HLQ.SDSNDBRM
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
  DSN SYSTEM(DB2ID)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC1) ISOLATION(UR)   -
   ACTION(REPLACE)  VALIDATE(BIND)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC2) ISOLATION(CS)   -
   ACTION(REPLACE)  VALIDATE(BIND)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC3) ISOLATION(RS)   -
   ACTION(REPLACE)  VALIDATE(BIND)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC4) ISOLATION(RR)   -
   ACTION(REPLACE)  VALIDATE(BIND)
   BIND PLAN(DB2IVPCL) KEEPDYNAMIC(YES)  ACTION(REPLACE) -
             PKLIST(DSNJDBC.DSNJDBC1,    -
                    DSNJDBC.DSNJDBC2,    -
                    DSNJDBC.DSNJDBC3,    -
                    DSNJDBC.DSNJDBC4)
 RUN PROGRAM(DSNTEP2) PLAN(DSNTEP71) -
   LIB('DB2HLQ.RUNLIB.LOAD')
END
```

```
//SYSIN    DD  *
GRANT EXECUTE ON PLAN DB2IVPCL TO PUBLIC;
/*
//
```

8. In UNIX System Services, in the directory that you specified by the `export DB2SQLJPROPERTIES` command, create the file db2sqljjdbc.properties that contains the following:

   ```
   DB2SQLJSSID=yourDB2ID
   DB2SQLJPLANNAME=DB2IVPCL
   DB2SQLJATTACHTYPE=RRSAF
   DB2SQLJDBRMLIB=DB2HLQ.SDSNDBRM
   ```

9. Run the client application by issuing following command from UNIX System Services:

   ```
   java samples.ivp.db2.DB2IvpClient IMSID
   ```

   The IVP displays the results of the tests that it performs.

   If the IVP was successful, it displays IVP PASSED.

   If the IVP was not successful, it displays IVP FAILED or IVP INCOMPLETE. Fix any errors and rerun the IVP.

Running the IMS Java Sample Application on DB2 UDB for z/OS

IMS Java provides a sample dealership application in addition to the IVP.

The IMS Java sample application for DB2 UDB for z/OS is two programs:

- The Java application DB2AutoClient, which runs under UNIX System Services
- The stored procedure DB2Auto, which runs in a WLM-managed address space.

Prerequisites:

- “Preparing to Run the Dealership Samples” on page 165
- “Running the IMS Java IVP from DB2 UDB for z/OS” on page 113

To run the IMS Java sample dealership application:

1. Ensure that the DB2 UDB for z/OS environment is configured and running as required by the IVP. If the DB2 UDB for z/OS environment is not configured and running for the IVP, perform steps 1 through 5 in “Running the IMS Java IVP from DB2 UDB for z/OS” on page 113 before continuing.
2. Define the stored procedure to DB2 UDB for z/OS by running the following job (*Your_WLM_Environment_Name* must match the APPLENV= parameter of the V71AWLM procedure):

   ```
   //CREATDLR JOB ,'name',
   //         MSGCLASS=H,TIME=3,
   //         USER=SYSADM,PASSWORD=XXXXXXXX,
   //         MSGLEVEL=(1)
   //CREATJSP EXEC PGM=IKJEFT01,DYNAMNBR=20
   //STEPLIB  DD  DISP=SHR,DSN=DB2HLQ.DSNEXIT
   //         DD  DISP=SHR,DSN=DB2HLQ.SDSNLOAD
   //SYSTSPRT DD  SYSOUT=*
   //SYSTSIN  DD  *
    DSN SYSTEM(DB2_Subsystem_Name)
     RUN PROGRAM(DSNTIAD) PLAN(DSNTIA71) -
     LIB('DB2HLQ.RUNLIB.LOAD') -
     PARM('RC0')
   //SYSPRINT DD  SYSOUT=*
   //SYSUDUMP DD  SYSOUT=*
   //SYSIN    DD  *
   ```

```
CREATE PROCEDURE AutoListModels(VARCHAR (100) IN,VARCHAR (100) OUT,
                                VARCHAR (100) OUT,VARCHAR(100) OUT,
                                VARCHAR (100) OUT,VARCHAR(100) OUT)
FENCED
NO SQL
LANGUAGE JAVA
DYNAMIC RESULT SET 0
EXTERNAL NAME 'samples.dealership.db2.DB2Auto.listModels'
PARAMETER STYLE JAVA
COLLID DSNJDBC
WLM ENVIRONMENT Your_WLM_Environment_Name;
GRANT EXECUTE ON PROCEDURE AutoListModels TO PUBLIC;
```

3. Create a DB2 plan that runs the client program by running the following job:

```
//BNDDLRCL JOB ,'name',
//         MSGCLASS=H,TIME=3,
//         USER=SYSADM,PASSWORD=XXXXXXXX,
//         MSGLEVEL=(1)
//BINDCLNT EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD  DISP=SHR,DSN=DB2HLQ.DSNEXIT
//         DD  DISP=SHR,DSN=DB2HLQ.SDSNLOAD
//DBRMLIB  DD   DISP=SHR,DSN=DB2HLQ.SDSNDBRM
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
  DSN SYSTEM(DB2ID)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC1) ISOLATION(UR)   -
   ACTION(REPLACE)  VALIDATE(BIND)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC2) ISOLATION(CS)   -
   ACTION(REPLACE)  VALIDATE(BIND)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC3) ISOLATION(RS)   -
   ACTION(REPLACE)  VALIDATE(BIND)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC4) ISOLATION(RR)   -
   ACTION(REPLACE)  VALIDATE(BIND)
   BIND PLAN(DB2DLRCL) KEEPDYNAMIC(YES)  ACTION(REPLACE) -
             PKLIST(DSNJDBC.DSNJDBC1,    -
                    DSNJDBC.DSNJDBC2,    -
                    DSNJDBC.DSNJDBC3,    -
                    DSNJDBC.DSNJDBC4)
 RUN PROGRAM(DSNTEP2) PLAN(DSNTEP71) -
   LIB('DB2HLQ.RUNLIB.LOAD')
END
//SYSIN    DD  *
GRANT EXECUTE ON PLAN DB2DLRCL TO PUBLIC;
/*
//
```

4. In UNIX System Services, edit the file db2sqljjdbc.properties by changing the DB2SQLJPLANNAME= parameter to DB2DLRCL:

```
DB2SQLJSSID=yourDB2ID
DB2SQLJPLANNAME=DB2DLRCL
DB2SQLJATTACHTYPE=RRSAF
DB2SQLJDBRMLIB=DB2HLQ.SDSNDBRM
```

5. Run the client application by issuing following command from UNIX System Services:

```
java samples.dealership.db2.DB2AutoCLient IMSID
```

   The sample application displays information about models of cars.

Running Your Stored Procedure from DB2 UDB for z/OS

Prerequisite: “Running the IMS Java IVP from DB2 UDB for z/OS” on page 113

To run your Java application that accesses IMS DB on DB2 UDB for z/OS:

1. In the JAVAENV data set, modify the CLASSPATH= parameter to point to your application files. If your application files are in JAR files, include the JAR file names. If the application files are not in JAR files, do not include the file names.
2. Edit the IMS-provided V71AWLM procedure as follows (if IMS.SDFSRESL does not contain the DRA startup table, add that data set to the DFSRESLB DD statement):

```
//V71AWLM PROC RGN=0M,APPLENV=,
//             DB2SSN=,NUMTCB=
//* Define the V71AWLM procedure parameters here on in the service policy.
//IEFPROC EXEC PGM=DSNX9WLM,REGION=&RGN,TIME=NOLIMIT,
//        PARM='&DB2SSN,&NUMTCB,&APPLENV'
//STEPLIB  DD  DISP=SHR,DSN=CEE.SCEERUN
//* DB2 Library
//         DD  DISP=SHR,DSN=yourDB2HLQ.SDSNLOAD
//         DD  DISP=SHR,DSN=yourDB2HLQ.SDSNLOD2
//         DD  DISP=SHR,DSN=yourDB2HLQ.RUNLIB.LOAD
//* DBRM library
//         DD  DISP=SHR,DSN=yourHLQ.SDSNDBRM
//DFSRESLB DD  DISP=SHR,DSN=IMS.SDFSRESL
//JAVAENV  DD  DISP=SHR,DSN=data set with ENVAR settings
//JSPDEBUG DD  SYSOUT=*
//CEEDUMP  DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
```

3. Create a new service policy for WLM. You can define the V71AWLM procedure parameters in the service policy or you can modify the procedure itself.
4. Define the procedure V71AWLM to RACF.
5. Start DB2 UDB for z/OS, the WLM-managed address space, and IMS DB.
6. Define the stored procedure to DB2 UDB for z/OS by running the following job (*Your_WLM_Environment_Name* must match the APPLENV= parameter of the V71ALWM procedure):

```
//name     JOB ,'name',
//         MSGCLASS=H,TIME=3,
//         USER=user,PASSWORD=XXXXXXXX,
//         MSGLEVEL=(1)
//CREATJSP EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD  DISP=SHR,DSN=DB2HLQ.DSNEXIT
//         DD  DISP=SHR,DSN=DB2HLQ.SDSNLOAD
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 DSN SYSTEM(DB2_Subsystem_Name)
  RUN PROGRAM(DSNTIAD) PLAN(DSNTIA71) -
  LIB('DB2HLQ.RUNLIB.LOAD') -
  PARM('RC0')
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSIN    DD  *
  CREATE PROCEDURE StoredProcName(... IN,... OUT)
  FENCED
  NO SQL
  LANGUAGE JAVA
  DYNAMIC RESULT SET 0
  EXTERNAL NAME 'package.StoredProcedure.targetMethod'
  PARAMETER STYLE JAVA
  COLLID DSNJDBC
  WLM ENVIRONMENT Your_WLM_Environment_Name;
  GRANT EXECUTE ON PROCEDURE StoredProcName TO PUBLIC;
```

7. Create a DB2 plan that runs the client program by running the following job:

```
//name     JOB ,'name',
//         MSGCLASS=H,TIME=3,
//         USER=user,PASSWORD=XXXXXXXX,
```

```
//         MSGLEVEL=(1)
//BINDCLNT EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD  DISP=SHR,DSN=DB2HLQ.DSNEXIT
//         DD  DISP=SHR,DSN=DB2HLQ.SDSNLOAD
//DBRMLIB  DD   DISP=SHR,DSN=DB2HLQ.SDSNDBRM
//SYSTSPRT DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSTSIN  DD  *
  DSN SYSTEM(DB2ID)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC1) ISOLATION(UR)   -
   ACTION(REPLACE)  VALIDATE(BIND)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC2) ISOLATION(CS)   -
   ACTION(REPLACE)  VALIDATE(BIND)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC3) ISOLATION(RS)   -
   ACTION(REPLACE)  VALIDATE(BIND)
   BIND PACKAGE (DSNJDBC) MEMBER(DSNJDBC4) ISOLATION(RR)   -
   ACTION(REPLACE)  VALIDATE(BIND)
   BIND PLAN(plan_name) KEEPDYNAMIC(YES)  ACTION(REPLACE) -
             PKLIST(DSNJDBC.DSNJDBC1,    -
                    DSNJDBC.DSNJDBC2,    -
                    DSNJDBC.DSNJDBC3,    -
                    DSNJDBC.DSNJDBC4)
 RUN PROGRAM(DSNTEP2) PLAN(DSNTEP71) -
   LIB('DB2HLQ.RUNLIB.LOAD')
END
//SYSIN    DD  *
GRANT EXECUTE ON PLAN plan_name TO PUBLIC;
/*
//
```

8. In UNIX System Services, in the directory that you specified by the `export DB2SQLJPROPERTIES` command, create the file db2sqljjdbc.properties that contains the following:

   ```
   DB2SQLJSSID=yourDB2ID
   DB2SQLJPLANNAME=plan_name
   DB2SQLJATTACHTYPE=RRSAF
   DB2SQLJDBRMLIB=DB2HLQ.SDSNDBRM
   ```

9. Run the client application.

Developing DB2 UDB for z/OS Stored Procedures that Access IMS DB

The stored procedure must perform the following tasks in the order listed. An example is given for each step:

1. Load the IMS JDBC driver:

   ```
   Class.forName("com.ibm.ims.db.DLIDriver");
   ```

2. Create an IMS JDBC connection:

   ```
   connection = DriverManager.getConnection
           ("jdbc:dli:package.DatabaseViewName/DRAname");
   ```

3. Create a statement:

   ```
   Statement statement = connection.createStatement();
   ```

4. Query the IMS database:

   ```
   ResultSet results = statement.executeQuery(query);
   ```

5. Move the results to the output parameters:

   ```
   parmOut[...]=...;
   ```

6. Close the connection:

   ```
   connection.close();
   ```

Chapter 6. CICS Applications

Java applications that run on CICS Transaction Server for z/OS can access IMS databases by using IMS Java.

Java applications use the IMS Java class libraries to access IMS. Other than the IMS Java layer, access to IMS from a Java application is the same as for a non-Java application.

Figure 17 shows a JCICS application that is accessing an IMS database using ODBA and IMS Java.

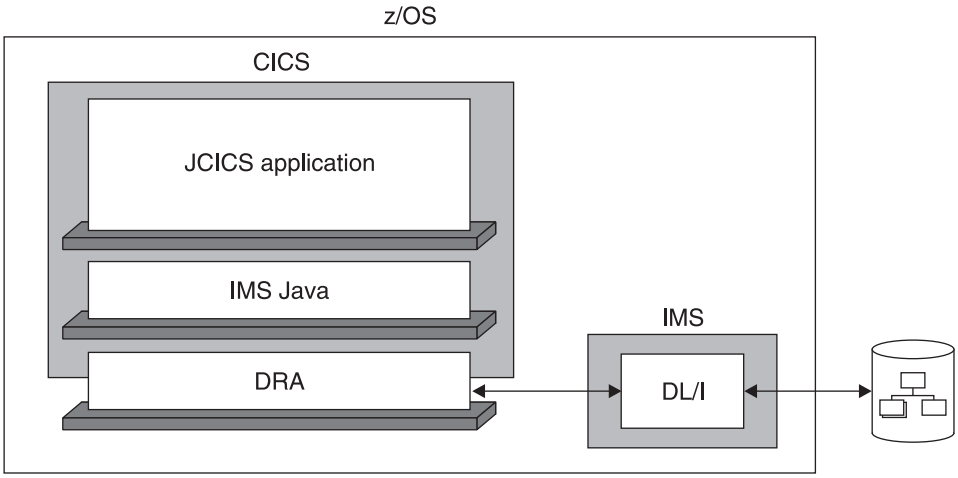


Figure 17. CICS Application Using IMS Java

The following topics provide additional information:

- "Configuring CICS for IMS Java"
- "Running the IMS Java IVP on CICS" on page 120
- "Running the IMS Java Sample Application on CICS" on page 121
- "Running Your Applications on CICS" on page 122
- "Developing CICS Applications that Access IMS DB" on page 123

Configuring CICS for IMS Java

To run Java applications that access IMS databases in a CICS environment, you must have CICS Transaction Server for z/OS Version 2 or later installed.

Prerequisite: "Installing IMS Java" on page 2

To configure CICS for IMS Java:

1. Build the CICSPSB DLL:
   a. Modify the Makefile, which is in the *pathprefix*/usr/lpp/ims/imsjava91/cics directory, by changing both occurrences of your.pdse.loadlib to the data set that will store the CICSPSB module.
   b. Set the CICS library to the _CXX_LSYSLIB environment variable by issuing the following command from the UNIX System Services prompt:

```
export  _CXX_LSYSLIB=CICS.SDFHLOAD:IMS.SDFSRESL:$_CXX_LSYSLIB
```

c. Run the Makefile by issuing the following command from the UNIX System Services prompt:

```
make
```

In the data set that you specified in the Makefile, two data set members named CICSPSB and DFSCLIB are created.

d. Add the data set that you specified in the Makefile to CICS STEPLIB concatenation.

2. Modify the CICS environment member DFHJVMPR, which is the JVM profile:
   a. Add a TMPREFIX variable, which adds libraries to the beginning of the middleware path.
   b. To the TMPREFIX variable, add the path to the IMS Java and XML class libraries as follows:

```
TMPREFIX=pathprefix/usr/lpp/ims/imsjava91/imsjava.jar
:pathprefix/usr/lpp/ims/imsjava91/lib/xalan.jar
:pathprefix/usr/lpp/ims/imsjava91/lib/xml-apis.jar
:pathprefix/usr/lpp/ims/imsjava91/lib/xercesImpl.jar
```

   c. Update the LIBPATH variable so that it contains the path to the file libJavTDLI.so as follows:

```
LIBPATH=pathprefix/usr/lpp/ims/imsjava91
```

3. If you are using SDK 1.4.1, which does not have the required version of Xalan, you must add the JVM environment variable java.endorsed.dirs and set it to the location of the required XML files (for example, `java.endorsed.dirs=pathprefix/usr/lpp/ims/imsjava91/lib`). IMS Java requires Xalan-Java 2.6.0 or later (or equivalent code function).

Next: “Running the IMS Java IVP on CICS”

Related Reading: For detailed information about CICS system definition, see the *CICS Transaction Server for z/OS: CICS System Definition Guide*.

Running the IMS Java IVP on CICS

After you configure CICS to run Java applications that access IMS databases, verify that IMS Java is installed correctly and that CICS is configured correctly by running the IMS Java installation verification program, which is named CICSIVP.

Prerequisites:

- “Configuring CICS for IMS Java” on page 119
- Ensure that the standard IMS IVPs have been run. These IVPs prepare the DBD for the IVP database, named IVPDB2, and load the IVP database. They also prepare the IMS Java application PSB (named DFSIVP37), build ACBs, and prepare other IMS control blocks required by the IMS Java IVPs. For details of how to run the IMS IVP procedures, see *IMS Version 9: Installation Volume 1: Installation Verification*.

To run the IMS Java IVP on CICS:

1. Create an HFS file named dfjjvmpr.props that contains the following class path:

```
ibm.jvm.shareable.application.class.path=
/pathprefix/usr/lpp/ims/imsjava91/samples.jar
```

   If you need to debug your application, you can also add the JVM debug options in this file.

2. In the DFHJVMPR member of the DFHJVM data set, add:

   ```
   JVMPROPS=path/dfjjvmpr.props
   STDOUT=path
   STDERR=path
   ```

3. Start IMS DB and CICS.
4. Turn off the uppercase translation feature of CICS by entering `CEOT NOUCTRAN`.

   By default, everything you type on the CICS terminal is converted to uppercase. However, the samples.jar file and path contain lowercase letters that must remain in lowercase.
5. Define a program that can run the CICSIVP application (JVM class):
   a. From the CICS terminal, enter: `CEDA DEFINE PROGRAM`
   b. In the list of program attributes, type the following:

      ```
      PROGram      ==> cicsivp
      Group        ==> ivp
      COncurrency  ==> Threadsafe
      JVM          ==> Yes
      JVMClass     ==> samples.ivp.cics.CICSIVP
      JVMProfile   ==>DFSJVMPR
      ```

   c. Press F3 to return to the main CICS terminal.
6. Define a transaction that can run the program:
   a. From the CICS terminal, enter: `CEDA DEFINE TRANSACTION`
   b. In the list of transaction attributes, type the following:

      ```
      TRANSaction  ==> civp
      Group        ==> ivp
      PROGram      ==> cicsivp
      ```

   c. Press F3 to return to the main CICS terminal.
7. Install the program that you defined in step 5:
   a. From the CICS terminal, enter: `CEDA INSTALL`
   b. In the list of program attributes, type the following:

      ```
      PROGram      => cicsivp
      Group        => ivp
      ```

   c. Press F3 to return to the main CICS terminal.
8. Install the transaction that you defined in step 6:
   a. From the CICS terminal, enter: `CEDA INSTALL`
   b. In the list of transaction attributes, type the following:

      ```
      TRANSaction  => civp
      Group        => ivp
      ```

   c. Press F3 to return to the main CICS terminal.
9. Run the transaction by entering: `civp`

   If the IVP was successful, it displays IVP PASSED.

   If the IVP was not successful, it displays IVP FAILED or IVP INCOMPLETE.

   See the STDOUT data set for the results of the individual tests that are performed by the IVP.

Running the IMS Java Sample Application on CICS

IMS Java provides the dealership sample application to run on CICS. The dealership sample files for CICS are located in *pathprefix*/usr/lpp/ims/imsjava91/samples/dealership/cics.

Prerequisites:

- "Running the IMS Java IVP on CICS" on page 120
- "Preparing to Run the Dealership Samples" on page 165

To run the IMS Java dealership sample on CICS:

1. Modify the HFS dfjjvmpr.props file to set the ibm.jvm.shareable.application.class.path= parameter to the path of the application. The location of the dfjjvmpr.props file is specified by the JVMPROPS variable in the CICS JVM profile:

   ```
   ibm.jvm.shareable.application.class.path=
   /pathprefix/usr/lpp/ims/imsjava91/samples/samples.jar
   ```

2. Start IMS DB and CICS.
3. Turn off the uppercase translation feature of CICS by entering: `CEOT NOUCTRAN`
4. Define a program that can run the IMS Java sample application (JVM class):
   a. From the CICS terminal, enter: `CEDA DEFINE PROGRAM`
   b. In the list of program attributes, type the following:

      ```
      PROGram      ==> cicsauto
      Group        ==> imsj
      COncurrency  ==> Threadsafe
      JVM          ==> Yes
      JVMClass     ==> samples.ivp.cics.CICSAuto
      JVMProfile   ==> DFSJVMPR
      ```

   c. Press F3 to return to the main CICS terminal.
5. Define a transaction that can run the program:
   a. From the CICS terminal, enter: `CEDA DEFINE TRANSACTION`
   b. In the list of transaction attributes, type the following:

      ```
      TRANSaction  ==> samp
      Group        ==> imsj
      PROGram      ==> cicsauto
      ```

   c. Press F3 to return to the main CICS terminal.
6. Install the program that you defined in step 4:
   a. From the CICS terminal, enter: `CEDA INSTALL`
   b. In the list of program attributes, type the following:

      ```
      PROGram      => cicsauto
      Group        => imsj
      ```

   c. Press F3 to return to the main CICS terminal.
7. Install the transaction that you defined in step 5:
   a. From the CICS terminal, enter: `CEDA INSTALL`
   b. In the list of transaction attributes, type the following:

      ```
      TRANSaction  => samp
      Group        => imsj
      ```

   c. Press F3 to return to the main CICS terminal.
8. Run the transaction by entering: `samp`
   The sample application displays information about models of cars.

Running Your Applications on CICS

Prerequisite: "Running the IMS Java IVP on CICS" on page 120

To run your Java application that accesses IMS DB from CICS:

1. Modify the HFS dfjjvmpr.props file to set the ibm.jvm.shareable.application.class.path= parameter to the path of the application. The location of the dfjjvmpr.props file is specified by the JVMPROPS variable in the CICS JVM profile.
2. Start IMS DB and CICS.
3. Turn off the uppercase translation feature of CICS by entering: `CEOT NOUCTRAN`
4. Define a program that can run the Java application (JVM class).
5. Define a transaction that can run the program.
6. Install the program that you defined in step 4.
7. Install the transaction that you defined in step 5.

Developing CICS Applications that Access IMS DB

The following programming model outlines the supported structure for JCICS applications that use IMS Java. The model is not complete, but it shows the normal flow of the application for both the JDBC and SSA access methods.

In a CICS environment, only one PSB can be allocated at a time. Therefore, an application can have only one active JDBC connection at a time. The application must close the JDBC connection before it opens another JDBC connection.

```
public static void main(CommAreaHolder cah) {  //Receives control

    conn = DriverManager.getConnection(...);   //Establish DB connection

    repeat {
        results = statement.executeQuery(...); //Perform DB processing
        ...
        //send output to terminal
    }

    conn.close();                              //Close DB connection
    return;
}
```

Chapter 7. JDBC Access to IMS Data

JDBC is the SQL-based standard interface for data access in the Java 2 SDK Standard Edition and Enterprise Edition. IMS Java's implementation of JDBC supports a selected subset of the full facilities of the JDBC 2.1 API.

IMS Java supports a subset of SQL keywords. Some keywords have specific IMS usage requirements. For this usage information, see "Supported SQL Keywords" on page 128.

Recommendation: Use JDBC to access IMS data instead of the IMS Java hierarchical database interface.

This chapter uses the sample dealership applications that are shipped with IMS Java to describe how to use JDBC to access an IMS database.

The following topics provide additional information:

- "Comparison of Hierarchical and Relational Databases"
- "Supported SQL Keywords" on page 128
- "Supported SQL Aggregate Functions" on page 136
- "SQL Extensions for XML Storage and Retrieval" on page 137
- "Supported JDBC Interfaces" on page 140
- "JDBC Prepared Statements for SQL" on page 142
- "Supported JDBC Data Types" on page 142
- "General Mappings from COBOL Copybook Types to IMS Java and Java Data Types" on page 144
- "JDBC Recommendations for IMS Databases" on page 145
- "Java Metadata Classes for IMS Databases" on page 146
- "Sample Application that Uses JDBC" on page 148

Comparison of Hierarchical and Relational Databases

A database segment definition defines the fields for a set of segment instances similar to the way a relational table defines columns for a set of rows in a table. In this way, segments relate to relational tables, and fields in a segment relate to columns in a relational table.

The name of an IMS segment becomes the table name in an SQL query, and the name of a field becomes the column name in the SQL query.

A fundamental difference between segments in a hierarchical database and tables in a relational database is that, in a hierarchical database, segments are implicitly joined with each other. In a relational database, you explicitly join two tables. A segment instance in a hierarchical database is already joined with its parent segment and its child segments, which are all along the same hierarchical path. In a relational database, this relationship between tables is captured by foreign and primary keys.

This section compares the dealership sample database, which is shipped with IMS Java, to a relational representation of the database.

The dealership sample database contains five segment types, which are shown in Figure 18. The root segment is the Dealer segment. Under the Dealer segment is its child segment, the Model segment. Under the Model segment are its children: the segments Order, Sales, and Stock. See Figure 34 on page 146 for the database description (DBD) of the dealership sample database.

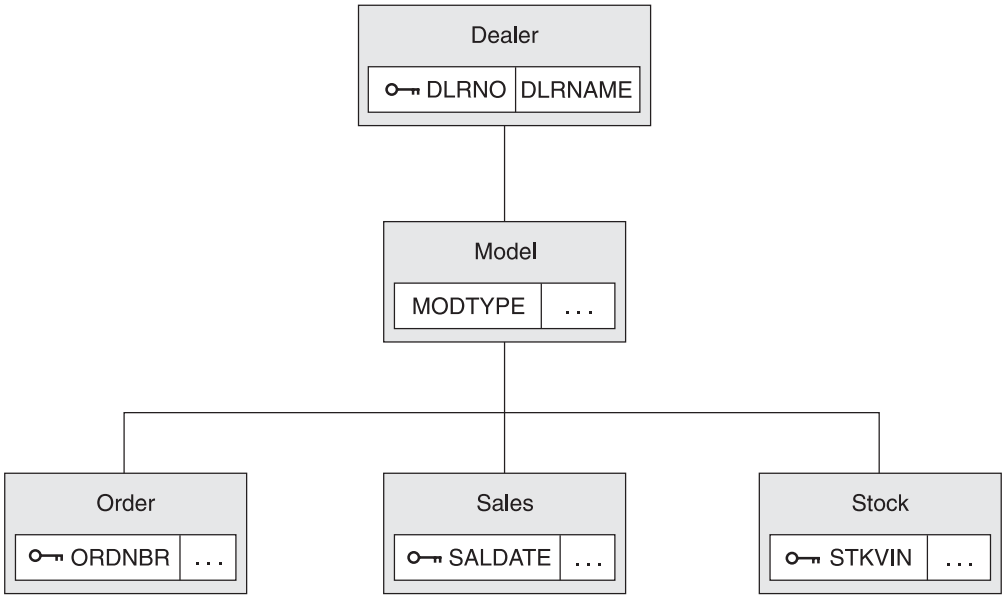


Figure 18. Sample Dealership Database

The Dealer segment identifies a dealer selling cars, and the segment contains a dealer name and a unique dealer number in the fields DLRNAME and DLRNO.

Dealers carry car types, each of which has a corresponding Model segment. A Model segment contains a type code in the field MODTYPE.

There is an Order segment for each car that is ordered for the dealership. A Stock segment is created for each car that is available for sale in the dealer's inventory. When the car is sold, a Sales segment is created.

Figure 19 on page 127 shows a relational representation of the IMS database record shown in Figure 18.

Important: This figure is only to help you understand how to use JDBC calls in a hierarchical environment. IMS Java does not change the structure of IMS data in any way.

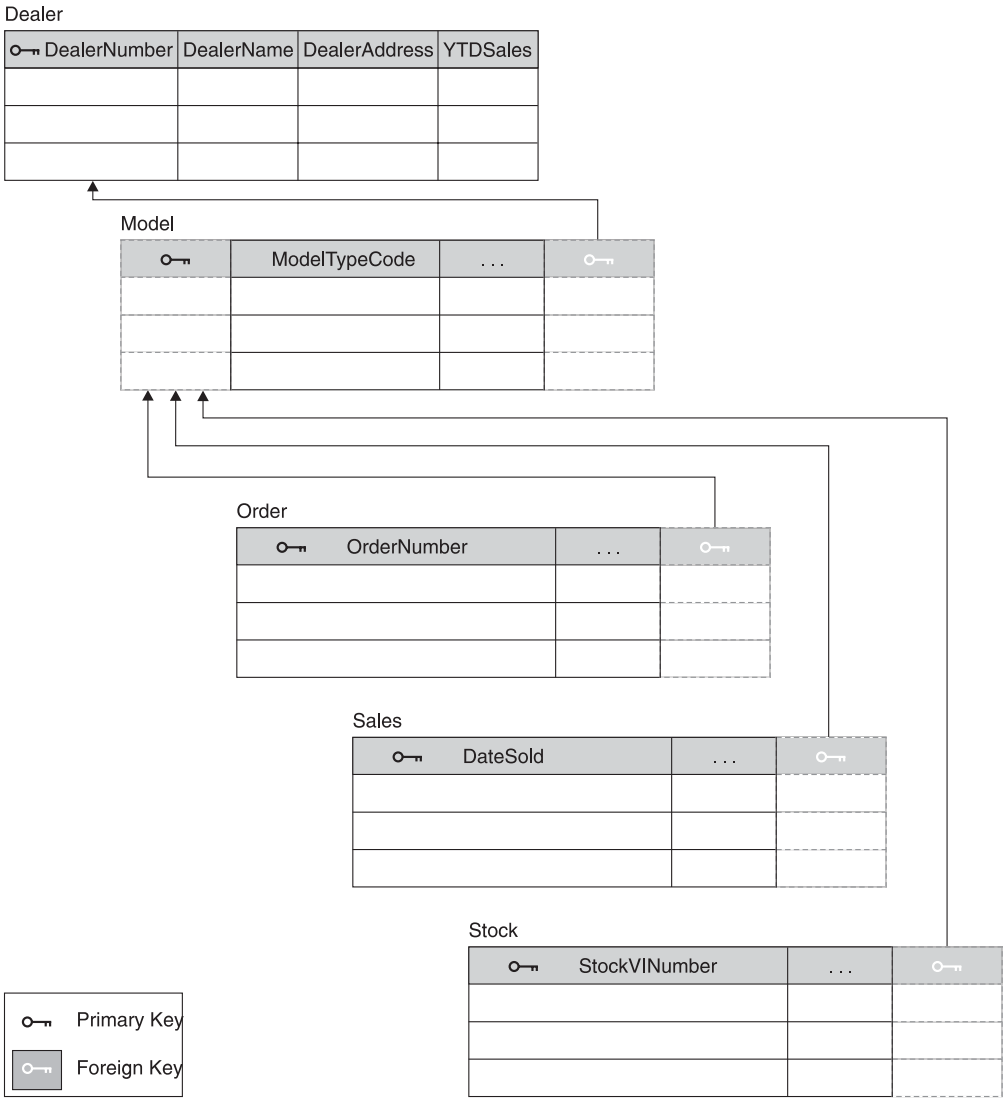


Figure 19. Relational Representation of the Dealership Database

If a segment does not have a unique key, which is similar to a primary key in relational databases, view the corresponding relational table as having a generated primary key added to its column (field) list. An example of a generated primary key is in the Model table (segment) of Figure 19. Similar to referential integrity in relational databases, you cannot insert, for example, an Order (child) segment to the database without it being a child of a specific Model (parent) segment.

Also note that the field (column) names have been renamed. You can rename segments and fields to more meaningful names using the DLIModel utility.

An occurrence of a segment in a hierarchical database corresponds to a row (or tuple) of a table in a relational database. Figure 20 on page 128 shows three dealership database records. The Dealer segment occurrences have dependent Model segment occurrences. The relational representation of these segment occurrences is shown in Figure 21 on page 128.

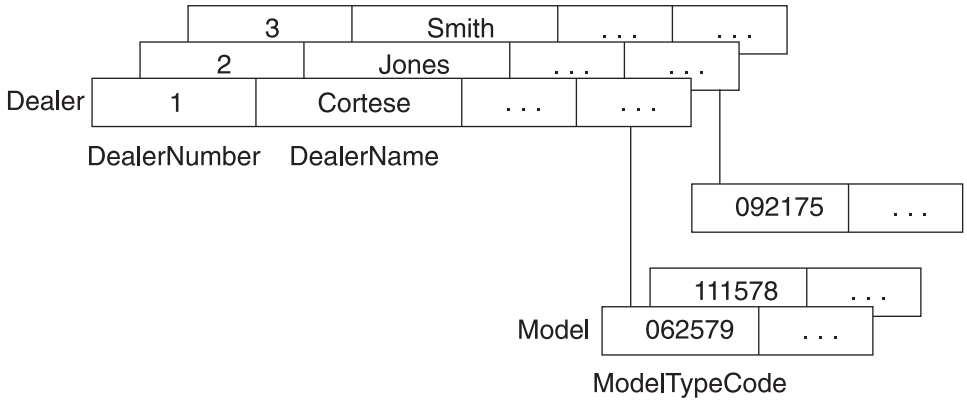


Figure 20. Segment Occurrences in the Dealership Database

Dealer

DealerNumber	DealerName	. . .	. . .
1	Cortese		
2	Jones		
3	Smith		

Model

ModelTypeCode	. . .	🔑
062579		1
111578		1
092175		2

Figure 21. Relational Representation of Segment Occurrences in the Dealership Database

The following example shows the SELECT statement of an SQL call. Model is a segment name that is used as a table name in the query:

```
SELECT * FROM Model
```

In the following example, ModelTypeCode is the name of a field contained in the Model segment and it is used in the SQL query as a column name:

```
SELECT * FROM Model WHERE ModelTypeCode = '062579'
```

In both of the preceding examples, Model and ModelTypeCode are alias names that you assign by using the DLIModel utility. These names will likely not be the same 8-character names used in the database description (DBD) for IMS. Alias names act as references to the 8-character names that are described in the DBD.

Supported SQL Keywords

The following portable SQL keywords are currently supported by IMS Java. IMS-specific usage for frequently-used keywords is described in this section. None of the keywords is case-sensitive. These keywords are a subset of all SQL keywords, which are listed in “SQL Keywords” on page 169.

ALL
AND
AS
ASC
AVG
COUNT
DELETE
DESC
DISTINCT
FROM
GROUP BY
INSERT
INTO
MAX
MIN
OR
ORDER BY
SELECT
SET
SUM
UPDATE
VALUES
WHERE

Important: Because the IMS Java SQL parser supports portable SQL, you cannot use any SQL keywords as Java aliases for PCBs, fields, or segments. When you define Java aliases, do not use an SQL keyword. If a PCB, segment, or field has the same name as an SQL keyword, you must explicitly define a different Java alias for it. If you use an SQL keyword as an alias for a PCB, segment, or field, your application will receive an error when it attempts an SQL query. For a complete list of SQL keywords, see "SQL Keywords" on page 169.

The following topics provide additional usage information about SQL keywords:

- "SELECT Statement Usage"
- "INSERT Statement Usage" on page 133
- "DELETE Statement Usage" on page 133
- "UPDATE Statement Usage" on page 134
- "FROM Clause Usage" on page 134
- "WHERE Clause Usage" on page 135

SELECT Statement Usage

A SELECT statement is a query used as a top-level SQL statement. A SELECT statement can be executed against a `Statement` or `PreparedStatement` object, which returns the results as a `ResultSet` object.

Figure 22 on page 130 shows sample code that uses the results of a SELECT query to update the `modelOutput` object with the model information. This example requires an `inputMessage` object with the ModelTypeCode field information.

```
public boolean getModelDetails(InputMessage inputMessage,
                       ModelOutput modelOutput) throws IMSException {

 // Parse the input message for ModelTypeCode
    String queryString = "SELECT * FROM DealershipDB.Model WHERE ModelTypeCode = "
       + "'" + inputMessage.getString("ModelTypeCode").trim() + "'";
 // Create a statement and execute it to get a ResultSet
    try {
        Statement statement = connection.createStatement();
        ResultSet results = statement.executeQuery(queryString);
        // Send back the result of the query
        // Note: because "ModelTypeCode" is unique - only 1 row
        // is returned
        if (results.next()) {
           modelOutput.setString("ModelTypeCode",
                                 results.getString("Type").trim());
           modelOutput.setString("Make",
                                 results.getString("CarMake").trim());
           modelOutput.setString("Model",
                                 results.getString("CarModel").trim());
           modelOutput.setString("Year",
                                 results.getString("CarYear"));
           modelOutput.setString("CityMiles",
                                 results.getString("EPACityMileage").trim());
           modelOutput.setString("HighwayMiles",results.getString
                                 ("EPAHighwayMileage").trim());
           modelOutput.setString("Price",
                                 results.getString("Price").trim());
           modelOutput.setString("Horsepower",
                                 results.getString("Horsepower").trim());
           return true;
        }
        else {
           reply("Unknown Type");
           return false;
        }
    }
    catch (SQLException e) {
       reply("Query Failed:"+ e.toString());
       return false;
    }
}
```

Figure 22. Example of SELECT Statement Query Results

Notice that the PCB reference name, DealershipDB, qualifies the Model segment name in the query string. You qualify the segment name with the PCB name because a PSB can contain multiple PCBs, and the PCBs can have segments with the same name. When you use the PCB name to indicate the exact segment to access, you avoid the ambiguity checking and improve the performance of your application.

Note: The method `trim()` is used because IMS character fields are padded with blanks if they are not long enough. The method trims off the extra blanks.

Figure 22 illustrates the use of a `Statement` object to execute an SQL query. You can also use a `PreparedStatement` object to execute an SQL query. A `PreparedStatement` object has two advantages over a `Statement` object:

- The SQL can be parsed one time for many executions of the query.
- You can build the query and use substitute values with each execution.

Selecting Multiple Segments

By using IMS Java to write IMS applications, you can avoid the long process of coding segment search arguments (SSAs) for every segment in the path that leads to the segment being queried. Instead, you can use the IMS Java JDBC driver for SQL queries to retrieve results from any segment in the path that leads to the segment being queried.

The primary difference between SQL queries to relational databases and SQL queries to IMS using IMS Java is that the hierarchical structure of an IMS database eliminates the need for the join that is required for tables in relational databases.

For example, Figure 23 is a query to a relational database for the address of a dealership that sells a particular car model (AnyCarModel):

```
SELECT Dealer.Address
FROM DealershipDB.Dealer,DealershipDB.Model
WHERE Model.CarMake = 'AnyCarModel'
                        AND Dealer.DealerName = Model.CarrierName
```

Figure 23. Sample Relational Database Query

In a relational database query, you must query two independent tables (Dealer and Model) and indicate how they are joined using a WHERE clause. This query is not valid against an IMS database.

In an IMS Java application, you can write the query in Figure 24 to access the same data in a hierarchical database using a WHERE clause:

```
SELECT Dealer.Address
FROM DealershipDB.Model
WHERE Model.CarMake = 'AnyCarModel'
```

Figure 24. Sample Hierarchical Database Query

In a hierarchical database, all data in segments along the hierarchical path from the root segment to the target segment are implicitly included in the query results, and therefore they do not need to be explicitly stated. In Figure 24, the information about the Dealer segment is included in the result set because it is along the hierarchical path to the Model segment.

Requirement: This implicit inclusion of segments is called a *path call*. For a path call to be made, the PROCOPT parameter in the PCB or SENSEG statement of the PSB source must include 'P'. If P is not included in the PROCOPT parameter and you issue a query that requires a path call to be made, an `SQLException` object is generated.

Selecting All Fields in a Segment

You can select all fields in a segment by using the asterisk (*) operator in the SELECT statement. In the following sample query, all of the fields from the Model segment are retrieved.

```
SELECT *
FROM DealershipDB.Model
```

If you want all of the fields in more than one segment, use the asterisk operator with the segments that you want to retrieve all the fields from. The SELECT statement in Figure 25 on page 132 shows an example where all of the fields from

both the Dealer and Model segments are retrieved. Figure 26 shows an equivalent query without using the asterisk operator.

```
SELECT Dealer.*,Model.*
FROM DealershipDB.Model
```

Figure 25. Simple Way to Select All Fields in a Segment

```
SELECT Dealer.DealerNo, Dealer.DealerCity, Dealer.Zip, Dealer.Phone,
       Model.ModelType, Model.Make, Model.Model, Model.Year, Model.MSRP, Model.Count
FROM DealershipDB.Model
```

Figure 26. Long Way to Select All Fields in a Segment

Segment-Qualified Fields

SQL dictates that whenever a field is common between two tables in an SQL query, the desired field must be table-qualified to resolve the ambiguity. Similarly, whenever a field name is common in any two segments along a hierarchical path, the field must be segment-qualified. For example, if a PCB has two segments, segment ROOT and segment CHILD, and both possess a field named `id`, any query that references the `id` field must be segment-qualified.

The following example is incorrect because the `id` field is not segment-qualified:

```
SELECT id
FROM PCBName.CHILD
WHERE id='10'
```

The following example is correct because the `id` field is segment-qualified:

```
SELECT CHILD.id
FROM PCBName.CHILD
WHERE ROOT.id='10'
```

Recommendations:

- For performance reasons, always qualify fields by prefixing the field names with a segment. This improves performance because IMS Java does not need to search through all the segments to locate the field and check for ambiguity.
- Although you do not need to provide the PCB reference name on the query unless the query is ambiguous without it, you should always provide the PCB reference name to remove ambiguity and to eliminate the need for checking.

Retrieving XML Using the SELECT Statement

You can retrieve XML from an IMS database using the `retrieveXML` user-defined function (UDF) in the SELECT statement. You can retrieve an intact XML document or compose an XML document from standard IMS segments.

For example, the following SELECT statement returns the Model fields in XML:

```
SELECT retrieveXML(Model)
FROM DealershipDB.Model
```

Related Reading: For more information about the `retrieveXML` UDF, see “SQL Extensions for XML Storage and Retrieval” on page 137.

Summary of SELECT Statement Usage

When using the SELECT statement in SQL calls to IMS databases:

- Qualify fields by prefixing them with segment names.
- Retrieve or create XML using the retrieveXML UDF.

- Select fields that are in any segment from the root segment down to the segment in the FROM clause.

INSERT Statement Usage

An INSERT statement inserts a segment instance with the specified data under any number of parent segments that match the criteria specified in the WHERE clause. All field names must be specified in the statement, unless you set a default value in the IMS Java metadata class with the DLIModel utility control statements. For more information about the DLIModel control statements, see the *IMS Version 9: Utilities Reference: System*.

Figure 27 shows an example of an INSERT statement that inserts a segment occurrence in the database using the DealershipDB PCB:

```
INSERT INTO DealershipDB.Sales (DateSold, PurchaserLastName,
    PurchaserFirstName, PurchaserAddress, SoldBy, StockVINNumber)
VALUES ('07032000', 'Beier', 'Otto', '101 W. 1st Street',
   'Springfield, OH', 'S123', '1ABCD23E4G5678901234')
WHERE Dealer.DealerNumber = 'A123'
AND Model.ModelTypeCode = 'K1'
```

Figure 27. Sample INSERT Statement

You can set a default value for any field in a segment by using the FIELD control statement when running the DLIModel utility. For more information, see the description of the Default parameter of the DLIModel utility in *IMS Version 9: Utilities Reference: System*.

One difference between JDBC queries to relational databases and to IMS is that standard SQL does not have a WHERE clause in an INSERT statement because tuples are being inserted into the table that is specified by the INTO keyword. In an IMS database, you are actually inserting a new instance of the specified segment, so you need to know where in the database this segment occurrence should be placed. With an INSERT statement, the WHERE clause is always necessary, unless you are inserting a root segment. With a prepared statement, the list of values can include a question mark (?) as the value that can be substituted before the statement is executed. For example:

```
INSERT INTO DealershipDB.Model(ModelTypeCode, CarMake, CarModel, CarYear, Price,
                               EPACityMileage, EPAHighwayMileage, Horsepower)
VALUES (?,?,?,?,?,?,?,?)
WHERE  Dealer.DealerNumber=?
```

DELETE Statement Usage

A DELETE statement can delete any number of segment occurrences that match the criteria specified in the WHERE clause. A DELETE statement with a WHERE clause also deletes the child segments of the matching segments. If no WHERE clause is specified, all of the segment occurrences of that type are deleted as are all of their child segment occurrences. Figure 28 shows an example of a DELETE statement:

```
DELETE FROM DealershipDB.Order
WHERE Dealer.DealerNumber = '123' AND OrderNumber = '345'
```

Figure 28. Sample DELETE Statement

UPDATE Statement Usage

An UPDATE statement modifies the value of the fields in any number of segment occurrences.

An UPDATE statement applies its SET operation to each instance of a specified segment with matching criteria in the WHERE clause. If the UPDATE statement does not have a WHERE clause, the SET operation is applied to all instances of the specified segment.

A SET clause contains at least one assignment. In each assignment, the values to the right of the equal sign are computed and assigned to columns to the left of the equal sign. For example, the UPDATE statement in Figure 29 is called to accept an order. When a customer accepts an order, the Order segment's SerialNo and DeliverDate fields are updated.

```
UPDATE DealershipDB.Order
SET SerialNo = '93234', DeliverDate = '12/11/2004'
WHERE OrderNumber = '123'
```

Figure 29. Sample UPDATE Statement

FROM Clause Usage

A FROM clause in IMS Java differs from standard SQL in that explicit joins are not required or allowed. Instead, the lowest-level segment in the query (in the SELECT statement and WHERE clause) must be the only segment that is listed in the FROM clause. The lowest-level segment in the FROM clause is equivalent to a join of all the segments, starting with the one that is listed in the FROM clause up the hierarchy to the root segment. For example, the FROM clause `FROM DealershipDB.Order` is equivalent to the following FROM clause in a relational query:

```
FROM DealershipDB.Order,DealershipDB.Model,DealershipDB.Dealer
```

PCB-Qualified SQL Queries

In IMS Java, connections are made to PSBs. Because there are multiple database PCBs in a PSB, there must be a way to specify which PCB (using its alias) in a PSB to use when executing an SQL query on the `java.sql.Connection` object. To specify which PCB to use, always qualify segments that are referenced in the FROM clause of an SQL statement by prefixing the segment name with the PCB name. You can omit the PCB name only if the PSB contains only one PCB.

Figure 30 shows a PCB-qualified SQL query.

```
SELECT *
    FROM DealershipDB.Model
```

Figure 30. PCB-Qualified SQL Query Example

Recommendation: For clarity and performance reasons, always qualify segments in the FROM clause by using the PCB alias.

Summary of FROM Clause Usage

When using the FROM clause in SQL calls to IMS databases:

- Do not join segments in the FROM clause.
- List only one segment in the FROM clause.

- List the lowest-level segment that is used in the SELECT list and WHERE clause.
- Qualify the segment in the FROM clause by using the PCB alias.

WHERE Clause Usage

IMS Java converts the WHERE clause in an SQL query to an SSA list when querying a database. SSA rules restrict the type of conditions you can specify in the WHERE clause. This section describes how you must form your WHERE clause so that it can be converted into SSA lists.

The WHERE clause can contain fields only from the segment in the FROM clause or segments that are higher in the hierarchy. The fields in the WHERE clause can be DBD-defined fields. These fields that are in the DBD are marked in the DLIModel IMS Java Report as being either primary key fields or search fields.

Fields in the WHERE clause also can be can be fields that are defined by a COBOL copybook or by the DLIModel utility when these non-DBD-defined fields are sub-fields of a DBD-defined field. See “Non-DBD-Defined Fields in the WHERE Clause” on page 136.

You cannot use parentheses in the WHERE clause because SSAs do not support parentheses.

Fields in the WHERE clause can be compared only to values, not to other fields. You can use the following operators between field names and values in the individual qualification statements:

```
<
<=
=
=<
<
!=
```

For example, the following WHERE clause will fail because it is trying to compare two fields:

```
WHERE Sales.SoldBy=Sales.PurchaserFirstName
```

The following example is valid because the WHERE clause is comparing a field to a value:

```
WHERE Sales.SoldBy='Lauren'
```

When using prepared statements, you can use the question mark (?) character, which is later filled in with a value. For example, the following WHERE clause is valid:

```
WHERE Sales.Soldby= ?
```

You can combine multiple qualification statements with AND and OR operators, but you must follow special rules. Because separate SSAs are created for each segment, list all qualification statements for a segment together and combine qualification statements for different segments with an AND operator.

Qualification statements that are combined with an AND operator make up a *qualification set*. For a qualification set to be satisfied (true), all qualification statements in the set must be satisfied. For the WHERE clause (and, therefore, the SSA qualification) to be satisfied, at least one qualification set must be satisfied.

The OR operator can be used only between qualification statements that contain fields from the same segment. Because of the way SSA lists are created, you cannot use the OR operator across segments. For example, the following WHERE clause will fail because the Soldby field and DealerName fields are in different segments:

```
WHERE Sales.SoldBy='Kiran' OR Dealer.DealerName='Bach'
```

However, the following WHERE clause is valid because the OR operator is between two qualification statements for the same segment:

```
WHERE Sales.SoldBy='Kyle' OR Sales.PurchaserFirstName='Chris'
```

Non-DBD-Defined Fields in the WHERE Clause

In addition to using DBD-defined (search) fields in your WHERE clause, you can use fields that are defined by a COBOL copybook or the DLIModel utility, as long as the fields are a subset of a field defined in a DBD. This function is useful when you have broken a large field that is defined in the DBD into smaller sub-fields. IMS supports all type conversions for the individual sub-fields.

The following rules apply when you use sub-fields in an SQL WHERE clause:

- The set of sub-fields that comprise a DBD-defined field must account for all of the bytes in the DBD-defined field.
- All sub-fields in a set that comprises a DBD-defined field must be listed together (similarly to how all fields in a segment must be listed together), but these sub-fields can be listed in any order.
- The only comparison operator allowed for sub-fields is ″=″.
- The sub-fields in a set that comprises a DBD-defined field must be separated by the AND operator. OR operators are not allowed to connect sub-fields in a set together. OR operators can be used to connect two sets of sub-fields.

Summary of WHERE Clause Usage

When using the WHERE clause in SQL calls to IMS databases:

- Use fields that are in any segment from the root segment down to the segment in the FROM clause.
- Qualify fields with segment names.
- Compare fields and sub-fields to values, not other fields.
- Do not use parentheses.
- List all qualification statements for a segment together.
- Combine qualification statements for different segments with an AND operator.
- Do not use the OR operator across segments.

Supported SQL Aggregate Functions

IMS Java supports the following SQL aggregate functions and related keywords:

AS
ASC
AVG
COUNT
DESC
GROUP BY
MAX
MIN
ORDER BY
SUM

Important: The field names that are specified in a GROUP BY or ORDER BY clause must match exactly the field name that is specified in the SELECT statement.

The supported SQL aggregate functions accept only a single field name in a segment as the argument (the DISTINCT keyword is not allowed). Table 3 shows the data types of the fields that are accepted by the aggregate functions, along with the resulting data type in the result set.

Table 3. Supported SQL Aggregate Functions and Their Supported Data Types

Function	Argument Type	Result Type
SUM and AVG	Byte	Long
	Short	Long
	Integer	Long
	Long	Long
	Single-precision floating point	Double-precision floating point
	Double-precision floating point	Double-precision floating point
MIN and MAX	Any type except BIT, BLOB, or BINARY	Same as argument type
COUNT	Any type	Long

The result set column name from an aggregate function is a combination of the aggregate function name and the field name separated by an underscore character (_). For example, the statement `SELECT MAX(age)` results in a column name MAX_age. Use this column name in all subsequent references—for example, `resultSet.getInt("MAX_age")`.

If the aggregate function argument field is segment-qualified, the result-set column name is the combination of the aggregate function name, the segment name, and the field name, separated by underscore characters (_). For example, `SELECT MAX(Employee.age)` results in a column name MAX_Employee_age.

You can use the AS keyword to rename the aggregate function column in the result set or any other field in the SELECT statement. You cannot use the AS keyword to rename a segment in the FROM clause. When you use the AS keyword to rename the field, you must use this new name to refer to the field. For example, if you specify `SELECT MAX(age) AS oldest`, a subsequent reference to the aggregate function column is `resultSet.getInt("oldest")`.

The result set type for aggregate functions and ORDER BY and GROUP BY clauses is always TYPE_SCROLL_INSENSITIVE, even if they are defined explicitly as TYPE_FORWARD_ONLY. A TYPE_SCROLL_INSENSITIVE result set is not sensitive to any changes in the database when the result set is open.

SQL Extensions for XML Storage and Retrieval

IMS Java has two SQL99 extensions for user-defined functions (UDFs): `retrieveXML` and `storeXML`. These UDFs are used during JDBC calls to store and retrieve XML from IMS databases. This interface is independent of the physical storage of the data.

In this topic:

- "retrieveXML UDF"
- "storeXML UDF" on page 139

retrieveXML UDF

The `retrieveXML` UDF creates an XML document from an IMS database and returns an object that implements the `java.sql.Clob` interface. It does not matter to the application whether the data is decomposed into standard IMS segments or the data is in intact XML documents in the IMS database.

The `Clob` JDBC type stores a Character Large Object as a column value in a row of the result set. The `getClob` method retrieves the XML document from the result set. Figure 31 shows the relationship between the `retrieveXML` UDF and the `getClob` method.

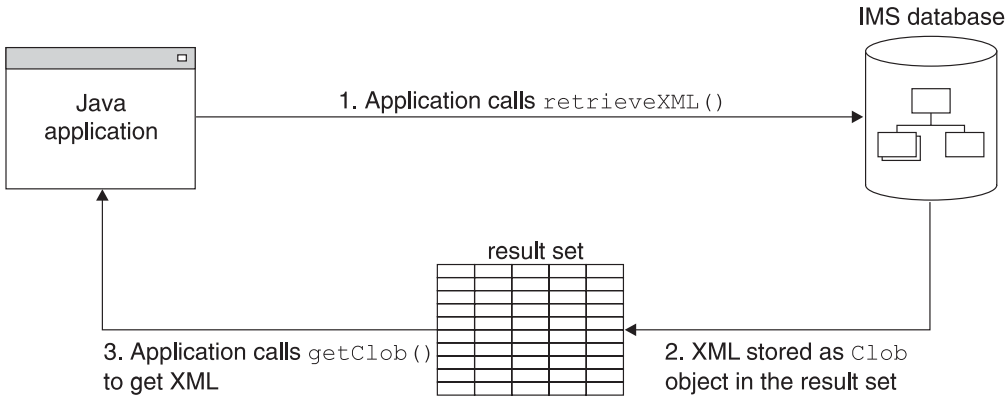


Figure 31. Creating XML Using the `retrieveXML` UDF and the `getClob` Method

To create an XML document, use a `retrieveXML` UDF in the SELECT statement of your JDBC call. Pass in the name of the segment that will be the root element of the XML document (for example, `retrieveXML(Model)`). The dependent segments of the segment that you pass in will be in the generated XML document if they match the criteria listed in the WHERE clause.

The segment that you specify to be the root element of the XML document does not have to be the root segment of the IMS record. The dependent segments are mapped to the XML document based on the generated XML schema.

Within a single application program, you can issue SELECT calls that contain `retrieveXML` UDFs against multiple PCBs in an application's PSB. You can also issue multiple `retrieveXML` UDFs that pass in various segments along the requested hierarchical path from a single SELECT call. From a single SELECT call, you can also retrieve other types of data in addition to the XML document (for example, `SELECT retrieveXML(Model), Dealer.DealerNo`).

The following example creates an XML document that has the root element of Model:

```
SELECT retrieveXML(Model)
FROM DealershipDB.Model
WHERE Model.CarYear = '2004'
```

The XML document that is created has the root element of the Model segment that has the CarYear field of 2004.

The XML document that is retrieved is stored in the result set. For each row in the result set, the UDF creates an implementation of the JDBC `java.sql.Clob` interface, and places it in the corresponding result set column. This `Clob` object encapsulates the XML document created from the database.

The storage requirements for the XML document `Clob` objects in a result set depend on whether the result set is forward-only or scroll-insensitive.

If the `Clob` object is returned to a forward-only `ResultSet` object, data is retrieved from the database and composed into XML only when the application requests the data. For example, if the application invokes the `getAsciiStream` or `getCharacterStream` method, the application receives a `Stream` object. As the application reads the XML stream, the segments are retrieved from the database and composed into XML. At the end of the stream, the entire XML document has been returned to the application having never been fully materialized in the `Clob` object.

If the `Clob` object is returned to a scroll-insensitive `ResultSet` object, the whole document is materialized in the `Clob`. This option requires more memory than forward-only result sets, especially for large XML documents and result sets with a lot of rows.

To retrieve the XML document from the result set, use the `getClob` method.

The following example retrieves an XML document, encapsulated by the `Clob` object, from the result set:

```
Clob xmlDoc = resultSet.getClob(1);
```

Using the `getClob` interface, you can, for example, retrieve all or part of document content as a `String` object, or request a `Stream` or `Reader` object for the document. With the `Stream` or `Reader` object, you can send the document to an output queue or as a response to an HTTP or SOAP request, or save it in a local HFS file. You can also selectively retrieve elements using a selected subset of XPath expressions, or transform the document using XSLT.

storeXML UDF

The `storeXML` UDF inserts an XML document into an IMS database at the position in the database that the WHERE clause indicates. IMS, not the application, uses the XML schema and the Java metadata class to determine the physical storage of the data into the database. It does not matter to the application whether the XML is stored intact or decomposed into standard IMS segments.

An XML document must be valid before in can be stored into a database. The `storeXML` UDF validates the XML document against the XML schema before storing it. If you know that the XML document is valid and you do not want IMS to revalidate it, use the `storeXML(false)` UDF.

To store an XML document, use the `storeXML` UDF in the INSERT INTO clause of a JDBC prepared statement. Within a single application program, you can issue INSERT calls that contain `storeXML` UDFs against multiple PCBs in an application's PSB.

The SQL query must have the following syntax:

```
INSERT INTO PCB.Segment (storeXML())
VALUES ( ? )
WHERE Segment.Field = value
```

Because an XML document is not a valid argument in the VALUES clause of the INSERT statement, you must use a prepared statement.

The following example stores the XML document named myDoc.xml from the file system into an IMS database using the Dealership PCB. A new Model segment, which is the root of the XML document, is inserted under the Dealer segment that has the number A123. The rest of the XML document is stored as dependent segments of Model as specified by the XML Schema.

```
InputStreamReader myXMLDoc =
                        new InputStreamReader(new FileInputStream("myDoc.xml"));

String query = "INSERT INTO Dealership.Model (storeXML())" +
               " VALUES ( ? )" +
               " WHERE Dealer.DealerNumber = 'A123' ";

PreparedStatement pstmt = conn.preparedStatement(query);

pstmt.setCharacterStream(1, myXMLDoc, -1);
```

Supported JDBC Interfaces

The following list describes the required interfaces by JDBC 2.1that are implemented in the database package, and it describes the limitations in the IMS Java implementation of these interfaces.

java.sql.Connection

`java.sql.Connection` is an object that represents the connection to the database. A `Connection` reference is retrieved from the `DriverManager` object that is implemented in the `java.sql` package. The `DriverManager` object obtains a `Connection` reference by querying its list of registered `Driver` instances until it finds one that supports the universal resource locator (URL) that is passed to the `DriverManager.getConnection` method.

Restriction: IMS does not support the local, connection-based commit scope that is defined in the JDBC model. Therefore, the IMS Java implementation of the methods `Connection.commit`, `Connection.rollback`, and `Connection.setAutoCommit` result in an SQL exception when these methods are called.

Figure 32 shows the sample dealership application code that establishes a connection to the sample database:

```
connection = DriverManager.getConnection
("jdbc:dli:dealership.application.DealerDatabaseView");
```

Figure 32. Establishing a Connection to the Dealership Database

java.sql.DatabaseMetaData

The `DatabaseMetaData` interface defines a set of methods for querying information about the database, including capabilities the database might or might not support. The class is provided for tool developers and is normally not used in client programs. Much of the functionality is specific to relational databases and is not implemented for DL/I databases.

java.sql.Driver

The `Driver` interface itself is not usually used in client applications, although

an application must dynamically load a particular Driver implementation by name. One of the first lines in an IMS JDBC program for IMS access must be:

```
Class.forName("com.ibm.ims.db.DLIDriver");
```

This code loads the IMS Java driver and causes the Driver implementation to register itself with the DriverManager object so that the driver can later be found by DriverManager.getConnection. The Driver implementation creates and returns a Connection object to the DriverManager object. The IMS Java implementation of JDBC is not fully JDBC-compliant and the Driver object method jdbcCompliant returns a value of false.

java.sql.Statement

A Statement interface is returned from the Connection.createStatement method. The Statement class and its subclass, PreparedStatement, define the interfaces that accept SQL statements and return tables as ResultSet objects. The code to create a Statement object is as follows:

```
Statement statement = connection.createStatement();
```

Restriction: The IMS Java implementation of the Statement interface does not support:

- Named cursors. Therefore, the method Statement.setCursorName throws an SQL exception.
- Aborting a DL/I operation. Therefore, the method Statement.cancel throws an SQL exception.
- Setting a time-out for DL/I operations. Therefore, the methods Statement.setQueryTimeout and Statement.getQueryTimeout throw SQL exceptions.

java.sql.ResultSet

The ResultSet interface defines an iteration mechanism to retrieve the data in the rows of a table, and to convert the data from the type defined in the database to the type required in the application. For example, ResultSet.getString converts an integer or decimal data type to an instance of a Java String. The code to return ResultSet object is as follows:

```
ResultSet results = statement.executeQuery(queryString);
```

Rather than building a complete set of results after a query is run, the IMS Java implementation of ResultSet interface retrieves a new segment occurrence each time the method ResultSet.next is called.

Restriction: The IMS Java implementation of ResultSet does not support:

- Returning data as an ASCII stream. Therefore the method ResultSet.getAsciiStream throws an SQL exception.
- Named cursors. Therefore the method ResultSet.getCursorName throws an SQL exception.
- The method ResultSet.getUnicodeStream, which is deprecated in JDBC 2.1.

java.sql.ResultSetMetaData

The java.sql.ResultSetMetaData interface defines methods to provide information about the types and properties in a ResultSet object. It includes methods such as getColumnCount, isSigned, getPrecision, and getColumnName.

java.sql.PreparedStatement

The `PreparedStatement` interface extends the `Statement` interface, adding support for pre-compiling an SQL statement (the SQL statement is provided at construction instead of execution), and for substituting values in the SQL statement (for example, `UPDATE Suppliers SET Status = ? WHERE City = ?`).

JDBC Prepared Statements for SQL

To improve performance of your IMS Java application, use JDBC prepared statements for the SQL. The `PreparedStatement` class completes the initial steps in preparing queries only once so that you need to provide the parameters only before each repeated database call.

The `PreparedStatement` object performs the following actions only once before repeated database calls are made:

1. Parses the SQL.
2. Cross-references the SQL with the IMS Java `DLIDatabaseView` object.
3. Builds SQL into SSAs before a database call is made.

Important: You must use a prepared statement when you store XML into a database. For more information, see “storeXML UDF” on page 139.

Supported JDBC Data Types

IMS Java supports the JDBC data types that are listed in Table 4. The DLIModel IMS Java Report indicates the JDBC type that is assigned to each field in the `DLIDatabaseView` subclass. Table 4 also lists the supported Java data types for each JDBC type.

Table 4. Supported JDBC Data Types

JDBC Data Type	Java Data Type
CHAR	`String`
CLOB	`Clob` (supported only for storage and retrieval of XML)
VARCHAR	`String`
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
FLOAT	float
DOUBLE	double
BINARY	byte[]
PACKEDDECIMAL	`java.math.BigDecimal`
ZONEDDECIMAL	`java.math.BigDecimal`
DATE	`java.sql.Date`
TIME	`java.sql.Time`
TIMESTAMP	`java.sql.Timestamp`

Table 5 shows the get methods that are available for accessing different types of JDBC data.

The methods that are marked with “X” are methods that are designed for accessing the given data type. No truncation or data loss occurs when you use those methods. The methods that are marked with “O” are all other legal calls. Data integrity is not be ensured when you use these methods. If the box is does not contain an “X” or an “O”, using that get method for that data type results in an exception.

Table 5. ResultSet.getxxx Methods to Retrieve JDBC Types

ResultSet.getx Methods	JDBC Types															
	TINYINT	SMALLINT	INTEGER	BIGINT	FLOAT	DOUBLE	BIT	CHAR	VARCHAR	PACKEDDECIMAL[1]	ZONEDECIMAL[1]	CLOB[2]	BINARY	DATE	TIME	TIMESTAMP
getByte	X	O	O	O	O	O	O	O	O	O	O					
getShort	O	X	O	O	O	O	O	O	O	O	O					
getInt	O	O	X	O	O	O	O	O	O	O	O					
getLong	O	O	O	X	O	O	O	O	O	O	O					
getFloat	O	O	O	O	X	O	O	O	O	O	O					
getDouble	O	O	O	O	O	X	O	O	O	O	O					
getBoolean	O	O	O	O	O	O	X	O	O	O	O					
getString	O	O	O	O	O	O	O	X	X	O	O		O	O	O	O
getBigDecimal	O	O	O	O	O	O	O	O	O	X	X					
getClob												X				
getBytes													X			
getDate								O	O					X		O
getTime								O	O						X	O
getTimestamp								O	O					O	O	X

Notes:

1. PACKEDDECIMAL and ZONEDDECIMAL are IMS Java JDBC types. All other types are standard SQL types defined in SQL92. **Restriction:** PACKEDDECIMAL and ZONEDDECIMAL data types do not support the Sign Leading or Sign Separate modes. For these two data types, sign information is always stored with the Sign Trailing method.
2. CLOB is supported only for the storage and retrieval of XML.

If the field type is either PACKEDDECIMAL or ZONEDDECIMAL, the type qualifier is the PICTURE string that represents the layout of the field. All COBOL PICTURE strings that contain valid combinations of 9s, Ps, Vs, and Ss are supported. Expansion of PICTURE strings is handled automatically. For example, ’9(5)’ is a valid PICTURE string. For zoned decimal numbers, the decimal point can also be used in the PICTURE string.

If the field contains DATE, TIME, or TIMESTAMP data, the type qualifier specifies the format of the data. For example, a type qualifier of ddMMyyyy indicates that the data is formatted as follows:

```
11122004 is December 11, 2004
```

For DATE and TIME types, all formatting options in the `java.text.SimpleDateFormat` class are supported.

For the TIMESTAMP type, the formatting option 'f' is available for nanoseconds. TIMESTAMP can contain up to nine 'f's and replaces the 'S' options for milliseconds. Instead, 'fff' indicates milliseconds of precision. An example TIMESTAMP format is as follows:

```
yyyy-mm-dd hh:mm:ss.fffffffff
```

General Mappings from COBOL Copybook Types to IMS Java and Java Data Types

Table 6 describes how COBOL copybook types are mapped to `DLITypeInfo` constants and Java data types.

Table 6. Mapping from COBOL Formats to DLITypeInfo Constants and Java Data Types

Copybook Format	`DLITypeInfo` Constant	Java Data Type
PIC X	CHAR	`java.lang.String`
PIC 9 BINARY[1]	See Table 7.[2]	See Table 7.[2]
COMP-1	FLOAT	float
COMP-2	DOUBLE	double
PIC 9 COMP-3[3]	PACKEDDECIMAL	`java.math.BigDecimal`
PIC 9 DISPLAY[4]	ZONEDDECIMAL	`java.math.BigDecimal`

Notes:

1. Synonyms for BINARY data items are COMP and COMP-4.
2. For BINARY data items, the `DLITypeInfo` constant and Java type depend on the number of digits in the PICTURE clause. Table 7 describes the type based on PICTURE clause length.
3. PACKED-DECIMAL is a synonym for COMP-3.
4. If the USAGE clause is not specified at either the group or elementary level, it is assumed to be DISPLAY.

Table 7 shows the `DLITypeInfo` constants and the Java data types based on the PICTURE clause.

Table 7. DLITypeInfo Constants and Java Data Types Based on the PICTURE Clause

Digits in PICTURE Clause	Storage Occupied	`DLITypeInfo` Constant	Java Data Type
1 through 4	2 bytes	SMALLINT	short
5 through 9	4 bytes	INTEGER	int
10 through 18	8 bytes	BIGINT	long

Table 8 on page 145 shows examples of specific copybook formats mapped to `DLITypeInfo` constants.

Table 8. Copybook Formats Mapped to DLITypeInfo Constants

Copybook Format	DLITypeInfo Constant
PIC X(25)	CHAR
PIC S9(04) COMP	SMALLINT
PIC S9(06) COMP-4	INTEGER
PIC S9(12) BINARY	BIGINT
COMP-1	FLOAT
COMP-2	DOUBLE
PIC S9(06)V99	ZONEDDECIMAL
PIC 9(06).99	ZONEDDECIMAL
PIC S9(06)V99 COMP-3	PACKEDDECIMAL

JDBC Recommendations for IMS Databases

Although the JDBC interface to an IMS database closely follows the relational database paradigm, the segments are physically stored in a hierarchical database, which affects the semantics of your JDBC calls to some extent. To avoid unexpected results or potential performance problems, follow these recommendations:

- When you code a SELECT list, generally try to supply predicates in the WHERE clause for all levels down the hierarchy to your target segment.

  If you supply a predicate in the WHERE clause for a target segment somewhere down the hierarchy and omit predicates for its parents, IMS must scan all candidate segments at the parent levels in an attempt to match the predicate that you supplied. For example, if you are retrieving a second-level segment and you supply a predicate for that second-level segment, but do not supply one for the root segment, IMS might perform a full database scan, testing every second-level segment under every root against the predicate. This has performance implications, particularly at the root level, and also might result in unexpected segments being retrieved. A similar consideration applies to locating segments for UPDATE clauses.
- When you insert a new segment, generally try to supply predicates in the WHERE clause for all levels down the hierarchy to your target new segment.

  If you omit a predicate for any level down to the insert target segment, IMS chooses the first occurrence of a segment at that level that allows it to satisfy remaining predicates, and performs the insert in that path. This might not be what you intended. For example, in a three-level database, if you insert a third-level segment, and supply a predicate for the root but none at the second-level, your new segment will always be inserted under the first second-level segment under the specified root.
- If you delete a segment that is not a bottom-level (leaf) segment in its hierarchy, you also delete the remaining segments in that hierarchical subtree. The entire family of segments of all types that are located hierarchically below your target deleted segment are also usually deleted.
- When you provide predicates to identify a segment, the search is generally faster if the predicate is qualified on a primary or secondary index key field, rather than simply on a search field. Primary and secondary key fields are identified for each segment in the DLIModel IMS Java Report.

Java Metadata Classes for IMS Databases

To access a set of IMS databases using JDBC, you must describe to IMS Java the application's view of the databases. The application view information is in the program specification block (PSB), but you must first convert this information into a form that you can use in your Java application: a subclass of `com.ibm.ims.db.DLIDatabaseView`. This subclass is called the IMS Java metadata class. When you establish the JDBC database connection, you pass the name of this class to IMS Java.

Create the metadata class for a PSB by providing the application PSB source and related DBD source files to the DLIModel utility so that the utility can generate the IMS Java metadata class. The DLIModel utility is described in *IMS Version 9: Utilities Reference: System*.

The examples used throughout this chapter are based on the sample application. The PSB for the sample dealership application is shown in Figure 33.

```
DLR_PCB1 PCB TYPE=DB,DBDNAME=DEALERDB,PROCOPT=GO,KEYLEN=42
         SENSEG NAME=DEALER,PARENT=0
         SENSEG NAME=MODEL,PARENT=DEALER
         SENSEG NAME=ORDER,PARENT=MODEL
         SENSEG NAME=SALES,PARENT=MODEL
         SENSEG NAME=STOCK,PARENT=MODEL
         PSBGEN PSBNAME=DLR_PSB,MAXQ=200,LANG=JAVA
         END
```

Figure 33. Sample PSB for the Sample Dealership Application

The physical DBD that is referenced by the PSB in Figure 33 is shown in Figure 34.

```
DBD NAME=DEALERDB,ACCESS=(HDAM,OSAM),RMNAME=(DFSHDC40.1.10)
SEGM NAME=DEALER,PARENT=0,BYTES=94,
FIELD NAME=(DLRNO,SEQ,U),BYTES=4,START=1,TYPE=C
FIELD NAME=DLRNAME,BYTES=30,START=5,TYPE=C
SEGM NAME=MODEL,PARENT=DEALER,BYTES=43
FIELD NAME=(MODTYPE,SEQ,U),BYTES=2,START=1,TYPE=C
FIELD NAME=MAKE,BYTES=10,START=3,TYPE=C
FIELD NAME=MODEL,BYTES=10,START=13,TYPE=C
FIELD NAME=YEAR,BYTES=4,START=23,TYPE=C
FIELD NAME=MSRP,BYTES=5,START=27,TYPE=P
SEGM NAME=ORDER,PARENT=MODEL,BYTES=127
FIELD NAME=(ORDNBR,SEQ,U),BYTES=6,START=1,TYPE=C
FIELD NAME=LASTNME,BYTES=25,START=50,TYPE=C
FIELD NAME=FIRSTNME,BYTES=25,START=75,TYPE=C
SEGM NAME=SALES,PARENT=MODEL,BYTES=113
FIELD NAME=(SALDATE,SEQ,U),BYTES=8,START=1,TYPE=C
FIELD NAME=LASTNME,BYTES=25,START=9,TYPE=C
FIELD NAME=FIRSTNME,BYTES=25,START=34,TYPE=C
FIELD NAME=STKVIN,BYTES=20,START=94,TYPE=C
SEGM NAME=STOCK,PARENT=MODEL,BYTES=62
FIELD NAME=(STKVIN,SEQ,U),BYTES=20,START=1,TYPE=C
FIELD NAME=COLOR,BYTES=10,START=37,TYPE=C
FIELD NAME=PRICE,BYTES=5,START=47,TYPE=C
FIELD NAME=LOT,BYTES=10,START=52,TYPE=C
DBDGEN
FINISH
END
```

Figure 34. DBD for the Sample Dealership Database

The DLIModel utility generates a subclass of `DLIDatabaseView` from the PSB and DBD. It also produces a report, called the DLIModel IMS Java Report, that provides information about the metadata class. Figure 35 shows an example of a DLIModel IMS Java Report.

The report supplements the information in the generated metadata class and the original PSB and DBD source files. Use this information when you write JDBC calls to IMS databases.

```
DLIModel IMS Java Report
========================
Class: DealerDatabaseView in package: com.ibm.ims.tooling generated for PSB: AUTPSB11

================================================
PCB: DealershipDB
================================================
Segment: Dealer
Field: DealerNumber Type=CHAR Length=4         ++ Primary Key Field ++
Field: DealerName Type=CHAR Length=30
Field: DealerAddress Type=CHAR Length=50
Field: YTDSales Type=PACKEDDECIMAL Type Qualifier=S9(18)
================================================
    Segment: Model
    Field: ModelTypeCode Type=CHAR Length=2    ++ Primary Key Field ++
    Field: CarMake Type=CHAR Length=10            (Search Field)
    Field: CarModel Type=CHAR Length=10          (Search Field)
    Field: CarYear Type=CHAR Length=4            (Search Field)
    Field: Price Type=CHAR Length=5              (Search Field)
    Field: EPACityMilage Type=CHAR Length=4
    Field: EPAHighwayMilage Type=CHAR Length=4
    Field: Horsepower Type=CHAR Length=4
    ================================================
        Segment: Order
        Field: OrderNumber Type=CHAR Length=6   ++ Primary Key Field ++
        Field: PurchaserLastName Type=CHAR Length=25   (Search Field)
        Field: PurchaserFirstName Type=CHAR Length=25  (Search Field)
        Field: Options Type=CHAR Length=30
        Field: Price Type=ZONEDDECIMAL Type Qualifier=99999
        Field: OrderDate Type=CHAR Length=8
        Field: SerialNo Type=CHAR Length=8
        Field: DeliverDate Type=CHAR Length=8
        ================================================
        Segment: Sales
        Field: DateSold Type=CHAR Length=8     ++ Primary Key Field ++
        Field: PurchaserLastName Type=CHAR Length=25    (Search Field)
        Field: PurchasetFirstName Type=CHAR Length=25  (Search Field)
        Field: StockVINumber Type=CHAR Length=20       (Search Field)
        Field: PurchaserAddress Type=CHAR Length=25
        Field: SoldBy Type=CHAR Start=84 Length=10
        ================================================
        Segment: Stock
        Field: StockVINumber Type=CHAR Length=20   ++ Primary Key Field ++
        Field: Color Type=CHAR Length=10             (Search Field)
        Field: Price Type=ZONEDDECIMAL Type Qualifier=99999
        Field: Lot Type=CHAR Length=10               (Search Field)
        Field: DateIn Type=CHAR Length=8
        Field: DateOut Type=CHAR Length=8
```

Figure 35. Sample DLIModel IMS Java Report for the Dealership Sample Database

The DLIModel IMS Java Report provides you with the following information:

- The name of the metadata class (`DealerDatabaseView` in this example) to use when you establish a connection to the database.

- The hierarchy of segments for each PCB.
- The fields within each segment, which are specified by the DBD, by any COBOL copybooks, or by control statements. For example, the fields DealerAddress and YTDSales in the Dealer segment are added fields.
- The names of PCBs, segments, and fields to use in your JDBC calls. These names may be alias names that are assigned to the IMS entities. Alias names are intended to be more representative and intuitive identifiers for your Java application to use rather than the 8-character names in the PSB and DBDs. In the example, the name `DealershipDB` replaces the PCB name `DLR_PCB1` from the PSB. A comparison of the names of the segments and the fields in the report with their names in the DBD shows that they have all been assigned more meaningful names.
- The data types of fields. The data types of the fields are based on the simple TYPE property of fields in the DBD and the DLIModel utility control statements. For example, the field YTDSales in the Dealer segment is type PACKEDDECIMAL with a type qualifier (format descriptor) of S9(18).
- The fields in each segment, which are identified as primary or secondary index fields, search fields, or other fields.

Sample Application that Uses JDBC

Because IMS is a hierarchical database, IMS Java does not fully implement the JDBC API. This section describes the IMS Java implementation of JDBC with a sample application.

To use JDBC to read, update, insert, and delete segment instances, an application must:

1. Obtain a connection to the database. Load the `DLIDriver` and retrieve a `Connection` object from the `DriverManager`.
2. Retrieve a `Statement` or `PreparedStatement` object from the `Connection` object and execute it. An example of this step is in Figure 36 on page 149.
3. Iterate the `ResultSet` object returned from the `Statement` or `PreparedStatement` object to retrieve specific field results. An example of this step is in Figure 36 on page 149.

Figure 36 on page 149, which is part of a sample method `showModelDetails`, obtains a `Connection` object, retrieves a `PreparedStatement` object, makes SQL calls to the database, and then iterates the `ResultSet` object that is returned from the `PreparedStatement` object.

```
public ModelDetailsOutput showModelDetails(ModelDetailsInput input)
throws NamingException, SQLException, IMSException {

    // Extract the key from the input
    String modelKey = input.getModelKey();
    ModelDetailsOutput output = new ModelDetailsOutput();

    // Validate the key
    if (modelKey != null && !modelKey.trim().equals("")) {

        // Build the SQL query.
        String query = "SELECT * FROM Dealer.ModelSegment WHERE "
                + "ModelSegment.ModelKey = '" + input.getModelKey() + "'";

        // Execute the query
        Statement statement = connection.createStatement();
        ResultSet results = statement.executeQuery(query);

        // Store the results in the output object and send it
        // back to the caller of this method.
        if (results.next()) {
            output.setMake(results.getString("Make"));
            output.setModelType(results.getString("ModelType"));
            output.setModel(results.getString("Model"));
            output.setYear(results.getString("Year"));
            output.setPrice(results.getString("MSRP"));
            output.setCount(results.getString("Counter"));
        }
    }
    return output;
}
```

Figure 36. Example JDBC Application

Imported Packages for JDBC Access to IMS Databases

To use unqualified class names instead of fully-qualified names in your program, include import statements at the top of the Java file.

Use the following import statement to make IMS database access classes available by their unqualified class names:

```
import com.ibm.ims.db.*;
```

Use the following import statement to make JDBC classes available by their unqualified class names:

```
import java.sql.*;
```

Connections to IMS Databases

Provide the name of the `DLIDatabaseView` subclass when retrieving a JDBC `Connection` object.

When the following code is executed, `DLIDriver`, a class in `com.ibm.ims.db`, registers itself with the JDBC `DriverManager` object:

```
Class.forName("com.ibm.ims.db.DLIDriver");
```

When the following code is executed, the JDBC `DriverManager` object determines which of the registered drivers supports the supplied string:

```
connection = DriverManager.getConnection
("jdbc:dli:dealership.application.DealerDatabaseView");
```

Because the supplied string begins with `jdbc:dli:`, the JDBC `DriverManager` object locates the `DLIDriver` instance and requests that it create a connection.

Chapter 8. XML Storage in IMS Databases

Because XML and IMS databases are both hierarchical, IMS is a natural database management system for managing XML documents. IMS allows you to easily receive and store incoming XML documents as well as compose XML documents from existing, legacy information stored that is in IMS databases. For example, you can:

- Compose XML documents from all types of existing IMS databases, to support, for example, business-to-business on demand transactions and intra-organizational sharing of data.
- Receive incoming XML documents and store them in IMS databases. These databases can be legacy databases or new databases. XML documents are stored decomposed: the document is parsed and element data and attributes are stored in fields in segments as normal IMS data. This is appropriate for data-centric documents.

You can store XML documents decomposed, intact, or in a combination of decomposed and intact. In decomposed storage mode, the incoming document is parsed and element data and attributes are stored in fields as normal IMS data. Decomposed storage is appropriate for data-centric documents. In intact storage, the incoming document, including its tags, is stored directly in the database without IMS being aware of its structure. Intact storage is appropriate for document-centric documents.

To store XML in an IMS database or to retrieve XML from IMS, you must first generate an XML schema and the IMS Java metadata class using the DLIModel utility. The metadata and schema are used during the storage and retrieval of XML. Your application uses the IMS Java JDBC user-defined functions `storeXML` and `retrieveXML` to store XML in IMS databases, create XML from IMS data, or to retrieve XML documents from IMS databases.

Figure 37 on page 152 shows the overall process for storing and retrieving XML in IMS.

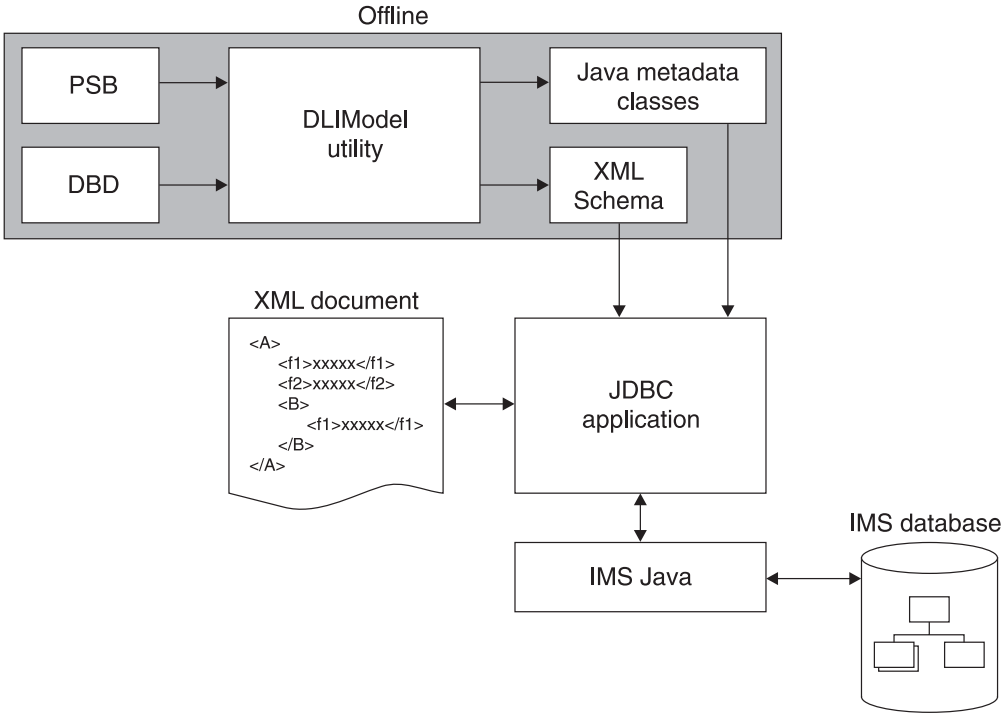


Figure 37. Overview of XML Storage in IMS

The following topics provide additional information:

- "Decomposed Storage Mode for XML"
- "Intact Storage Mode for XML" on page 154
- "XML Schema" on page 157
- "XML Type Representation" on page 157
- "JDBC Interface for Storing and Retrieving XML" on page 158

Decomposed Storage Mode for XML

In decomposed storage mode, all elements and attributes are stored as regular fields in optionally repeating DL/I segments. During parsing, all tags and other XML syntactic information is checked for validity and then discarded. The parsed data is physically stored in the database as standard IMS data, meaning that each defined field in the segment is of an IMS standard type. Because all XML data is composed of string types (typically Unicode) with type information existing in the validating XML schema, each parsed data element and attribute can be converted to the corresponding IMS standard field value and stored into the target database.

Inversely, during XML retrieval, DL/I segments are retrieved, fields are converted to the destination XML encoding, tags and XML syntactic information (stored in the XML schema) are added, and the XML document is composed.

Figure 38 on page 153 shows how XML elements are decomposed and stored into IMS segments.

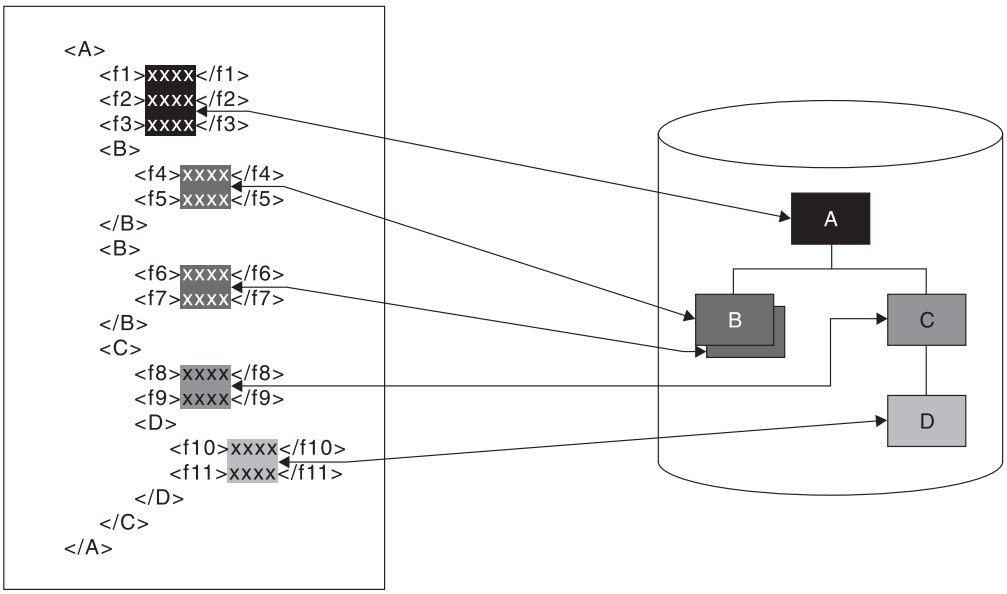


Figure 38. How XML is Decomposed XML and Stored in IMS Segments

Decomposed storage mode is suitable for data-centric XML documents, where the elements and attributes from the document typically are either character or numeric items of known short or medium length that lend themselves to mapping to fields in segments. Lengths are typically, though not always, fixed.

The XML document data can start at any segment in the hierarchy, which is the root element in the XML document. The segments in the subtree below this segment are also included in the XML document. Elements and attributes of the XML document are stored in the dependent segments of the root element segment. Any other segments in the hierarchy that are not dependent segments of that root element segment are not part of the XML document and, therefore, are not described in the describing XML schema.

When an XML document is stored in the database, the value of all segment fields is extracted directly from the XML document. Therefore, any unique key fields in any of the XML segments must exist in the XML document as an attribute or simple element.

The XML hierarchy is defined by a PCB hierarchy that is based on either a physical or a logical database. Logical relationships are supported for retrieval and composition of XML documents, but not for inserting documents.

For a legacy database, either the whole database hierarchy, or any subtree of the hierarchy can be considered as a decomposed data-centric XML document. The segments and fields that comprise the decomposed XML data are determined only by the definition of a mapping (the XML schema) between those segments and fields and a document.

One XML schema is generated for each database PCB. Therefore, multiple documents may be derived from a physical database hierarchy through different XML Schemas. There are no restrictions on how these multiple documents overlap and share common segments or fields.

A new database can be designed specifically to store a particular type of data-centric XML documents in decomposed form.

Intact Storage Mode for XML

In intact storage mode, all or part of an XML document is stored intact in a field. The XML tags are not removed and IMS does not parse the document. XML documents can be large, so the documents can span the primary intact field, which contains the XML root element, and fields in overflow segments. The segments that contain the intact XML documents are standard IMS segments and can be processed like any other IMS segments. The fields, because they contain unparsed XML data, cannot be processed like standard IMS fields. However, intact storage of documents has the following advantages over decomposed storage mode:

- IMS does not need to compose or decompose the XML during storage and retrieval. Therefore, you can process intact XML documents faster than decomposed XML documents.
- You do not need to match the XML document content with IMS field data types or lengths. Therefore, you can store XML documents with different structure, content, and length within the same IMS database.

Intact XML storage requires a new IMS database or an extension of an existing database because the XML document must be stored in segments and fields that are specifically tailored for storing intact XML.

To store all or part of an XML document intact in an IMS database, the database must define a base segment, which contains the root element of the intact XML sub-tree. The rest of the intact XML sub-tree is stored in overflow segments, which are child segments of the the base segment.

The base segment contains the root element of the intact XML sub-tree and any decomposed or non-XML fields. Table 9 shows the format of the primary intact field. This format is defined in the DBD, which is described in “DBDs for Intact XML Storage” on page 155.

Table 9. Primary Intact Field Format

Byte	Content
1	0x01
2	Reserved
3–4	Bit 1 indicates whether there are overflow segments Bit 2–16 indicate the length of the XML data in this field
rest of field	XML data

The overflow segment contains only the only the overflow XML data field. Table 10 shows the format of the overflow XML data field. This format is defined in the DBD, which is described in “DBDs for Intact XML Storage” on page 155.

Table 10. Overflow XML Data Field Format

Byte	Content
1–2	Key field sequence number
2–4	Bit 1 indicates whether there are more overflow segments after this segment Bit 2–16 indicate the length of the XML data in this field

Table 10. Overflow XML Data Field Format (continued)

Byte	Content
rest of field	Continuation of XML data

Side Segments for Secondary Indexing

IMS cannot search intact XML documents for specific elements within the document. However, you can create a side segment that contains specific XML element data. IMS stores the XML document intact, and also decomposed a specific piece XML data into a standard IMS segment. This segment can then be searched with a secondary index.

Figure 39 shows a base segment, an overflow segment, and the side segment for secondary indexing.

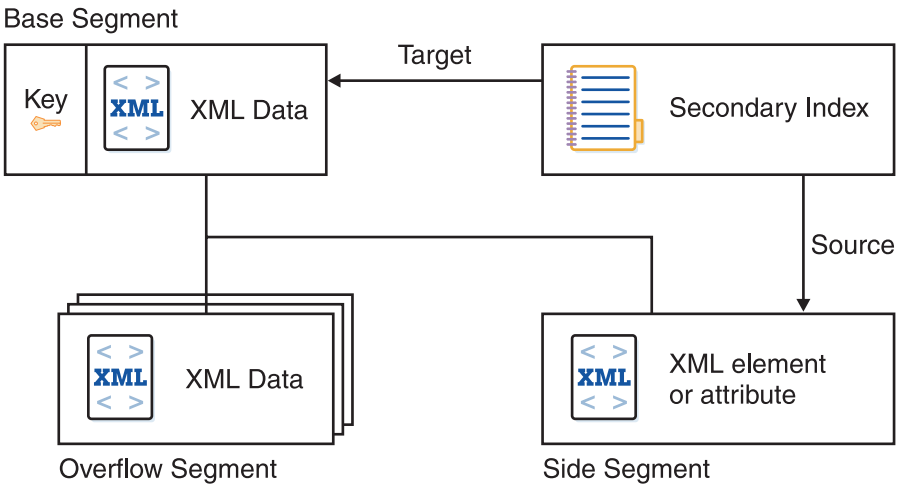


Figure 39. Intact Storage of XML with a Secondary Index

For information about the DBDs for side segments see Figure 41 on page 156 in “DBDs for Intact XML Storage.”

DBDs for Intact XML Storage

The DBD shown in Figure 40 on page 156 defines a base segment and an overflow segment. The XML intact field in the base segment contains a 4–byte header, so you must define the field to be greater than 4 bytes. The XML intact field in the overflow segment contains a 2–byte header for the length, so you must define the field to be greater than 2 bytes.

```
DBD     NAME=dbdname,ACCESS=(PHDAM,VSAM),RMNAME=(DFSHDC40,1,5,bytes)
*Base segment
SEGM    NAME=segname1,PARENT=0,BYTES=seglen1
* XML intact field, which contains a 4-byte header
FIELD   NAME=INTDATA,BYTES=length,START=startpos,TYPE=C
* Additional non-intact fields can be specified in segment
*
* Overflow Segment
SEGM    NAME=segname2,PARENT=segname1,BYTES=seglen2
FIELD   NAME=(SEQNO,SEQ,U),BYTES=2,START=1,TYPE=C
* XML intact field, which contains a 2-byte header for length
FIELD   NAME=INTDATA,BYTES=1,START=3,TYPE=C
DBDGEN
FINISH
END
```

Figure 40. DBD for Intact XML Storage and No Secondary Indexes

The DBD shown in Figure 41 defines a base segment, and overflow segment, and a side segment that is used by two secondary indexes.

```
DBD     NAME=dbdname,ACCESS=(PHDAM,VSAM),RMNAME=(DFSHDC40,1,5,200)
* Base segment
SEGM    NAME=segname1,PARENT=0,BYTES=seglen1
* XML intact field, which contains a 4-byte header
FIELD   NAME=INTDATA,BYTES=length,START=startpos,TYPE=C
*
LCHILD  NAME=(issegname1,isdbd1),POINTER=INDX
XDFLD   NAME=issrch1,SRCH=iskey1,SEGMENT=ssegname1
LCHILD  NAME=(issegname2,isdbd2),POINTER=INDX
XDFLD   NAME=issrch2,SRCH=iskey2,SEGMENT=ssegname2
* Overflow segment
SEGM  NAME=segname2,PARENT=segname1,BYTES=seglen2
FIELD   NAME=(SEQNO,SEQ,U),BYTES=2,START=1,TYPE=C
* XML intact field, which contains a 2-byte header for length
FIELD   NAME=INTDATA,BYTES=1,START=3,TYPE=C
*
* Index side segment 1
SEGM    NAME=ssegname1,PARENT=segname1,BYTES=iseglen1
FIELD   NAME=(iskey1,SEQ,U),BYTES=islen1,START=1,TYPE=C
*
* Index side segment 2
SEGM    NAME=ssegname2,PARENT=segname1,BYTES=iseglen2
FIELD   NAME=(iskey2,SEQ,U),BYTES=islen2,START=1,TYPE=C
*
DBDGEN
FINISH
END
```

Figure 41. DBD for Intact XML Storage and Two Secondary Indexes

Figure 42 shows a secondary index for the DBD shown in Figure 41.

```
DBD    NAME=isdbd1,ACCESS=(PSINDEX,VSAM)

SEGM   NAME=issegname1,PARENT=0,BYTES=iseglen
FIELD  NAME=(isfld1,SEQ,U),BYTES=islen1,START=1,TYPE=C
LCHILD NAME=(ssegname1,dbdname),INDEX=issrch1
DBDGEN
FINISH
END
```

Figure 42. Secondary Index DBD for Intact XML Storage

XML Schema

The generated XML schema is an XML document that describes an IMS database based on a PCB. In order to retrieve or store XML in IMS, an XML schema is required . IMS uses the XML schema to validate an XML document that is either being stored into IMS or being retrieved from IMS. The XML schema, not the application program, determines the structural layout of the XML in the database. The `DLIDatabaseView` subclass determines how the data is physically stored in the database.

The DLIModel utility generates a schema that is based on a PCB. For information about generating an XML schema, see *IMS Version 9: Utilities Reference: System*.

The generated XML schema must be made available at run time to provide the XML structure of the data retrieved from the database or of an incoming XML document being stored into IMS.

By default, a schema is loaded from the HFS root directory based on the PSB and PCB names that are used in the `retrieveXML` or `storeXML` query.

To override the default location, which is the root file system, define the environment variable ″http://www.ibm.com/ims/schema-resolver/file/path″ with the value of the XML schema locations. For example, if the XML schemas are located in the directory /u/schemas, define an environment variable to the SDK as follows:

```
-Dhttp://www.ibm.com/ims/schema-resolver/file/path=/u/schema/
```

You can also specify the XML schema in the application program by setting the system property. For example:

```
System.setProperty("http://www.ibm.com/ims/schema-resolver/file/path", "/u/schema");
```

XML Type Representation

IMS has no inherent type information and stores all of its segments as a simple array of bytes. Therefore, it is up to all application programs that access an IMS segment to agree on three pieces of information:

- A list of fields that are represented within each segment
- What data type each field stores
- How each data type is represented as bytes, including field redefinitions

In order for IMS to correctly produce XML documents from the database and to breakdown and store XML documents into the database, it also needs to satisfy these conditions.

In addition to the type of the field, each XML schema document lists every field as one of the allowed 42 XML types. This information instructs any user of a valid XML document how to interpret the information within it, and informs IMS in how to generate an outgoing, or decompose an incoming, XML document. The `retrieveXML` and `storeXML` UDFs validate XML documents, according to the generated XML schema, and use the XML schema with the IMS Java metadata to determine how to extract element and attribute values to populate fields and segments.

Table 11 lists the IMS Java-supported XML schema data types.

Table 11. IMS Java-Supported XML Schema Data Types

JDBC Data Type	XML Schema Data Type
CHAR	xsd:string
VARCHAR	xsd:string
BIT	xsd:boolean
TINYINT	xsd:byte
SMALLINT	xsd:short
INTEGER	xsd:int
BIGINT	xsd:long
FLOAT	xsd:float
DOUBLE	xsd:double
BINARY	Not supported
PACKEDDECIMAL	xsd:decimal
ZONEDECIMAL	xsd:decimal
DATE	xsd:gYear (for yyyy-MM) xsd:date (for yyyy) xsd:gYearMonth (for yyyy-MM-dd)
TIME	xsd:time
TIMESTAMP	xsd:dateTime

JDBC Interface for Storing and Retrieving XML

A Java application program that is running in any of the following environments can store XML in IMS and retrieve XML from IMS:

- IMS dependent region (JMP or JBP)
- WebSphere Application Server for z/OS
- WebSphere Application Server on a non-z/OS platform
- DB2 UDB for z/OS stored procedure
- CICS JCICS region

The IMS Java JDBC interface has been extended to support storage and retrieval of XML. For more information, see “SQL Extensions for XML Storage and Retrieval” on page 137.

Chapter 9. Problem Determination

This chapter describes how to debug your Java applications that use IMS Java and determine the source of problems within your applications.

The following topics provide additional information:

- "Exceptions"
- "XML Tracing for IMS Java" on page 160
- "Debugging an Unresettable JVM in a JMP or JBP Region" on page 163

Exceptions

Exceptions are thrown as a result of non-blank status codes and non-zero return codes (in cases when there were no PCBs to deliver status codes) from IMS DL/I calls. Even though an exception is thrown by the `JavaToDLI` class for every non-blank status code, some of these exceptions are caught by the application or database packages and converted to return values.

How Exceptions Map to DL/I Status Codes

The `com.ibm.ims.base.IMSException` class extends the `java.lang.Exception` class.

The `DLIException` class extends the `IMSException` class. The `DLIException` class includes all errors that occur within the IMS Java library that are not a result of any call to IMS.

You can use the following methods to get information from an `IMSException` object:

getAIB
: Returns the IMS application interface block (AIB) from the DL/I call that caused the exception. The IMS AIB is null for the `DLIException` object. The methods on the AIB can be called to return other information at the time of the failure, including the resource or PCB name and the PCB itself.

getStatusCode
: Returns the IMS status code from the DL/I call that caused the exception. This method works with the `JavaToDLI` set of constants. The status code is zero (0) for a `DLIException` object.

getFunction
: Returns the IMS function from the DL/I call that caused the exception. The function is zero (0) for a `DLIException` object.

The following database access methods of the `DLIConnection` class return false if they receive a GB status code (no more such segments or segment not found) or a GE status code (no such segment or end of database):

- `DLIConnection.getUniqueSegment`
- `DLIConnection.getNextSegment`
- `DLIConnection.getUniqueRecord`
- `DLIConnection.getNextRecord`
- `DLIConnection.getNextSegmentInParent`

The IMSMessageQueue.getUniqueMessage method returns false if it receives a QC (no more messages) status code. The IMSMessageQueue.getNextMessage method returns false if it receives a QD status code, which means that there are no more segments for multi-segment messages.

The example in Figure 43 extracts information from an IMSException object:

```
try {
   DealerDatabaseView dealerView = new DealerDatabaseView();
   DLIConnection connection = DLIConnection.createInstance(dealerView);
   connection.getUniqueSegment(dealerSegment, dealerSSAList);
} catch (IMSException e ) {
   short statusCode = e.getStatusCode();
   String failingFunction = e.getFunction();
}
```

Figure 43. IMSException Class Example

Related Reading: For more information about DL/I status codes, see *IMS Version 9: Application Programming: Database Manager* and *IMS Version 9: Application Programming: Transaction Manager*.

SQLException Objects

An SQLException object is thrown to indicate that an error has occurred either in the Java address space or during database processing.

Each SQLException provides the following information:

- A string that describes the error.
  - This string is available through the use of the getMessage() method.
- An “SQLstate” string that follows XOPEN SQLstate conventions.
  - The values of the SQLstate string are described in the XOPEN SQL specification.
- A link to the next SQL exception if more than one was generated.
  - The next exception is used as a source of additional error information.

XML Tracing for IMS Java

Using the com.ibm.ims.base.XMLTrace class for z/OS applications or com.ibm.ims.rds.XMLTrace for distributed applications, you can debug your Java applications by tracing, or documenting, the flow of control throughout your application. By setting up trace points throughout your application for output, you can isolate problem areas and, therefore, know where to make adjustments to produce the results you expect. In addition, because the XMLTrace class supports writing input parameters and results, and the IMS Java library methods use this feature, you can verify that correct results occur across method boundaries.

The XMLTrace class is different from the DLIModel utility trace. For information about how to enable tracing for the DLIModel utility, see the OPTIONS statement of the DLIModel utility in *IMS Version 9: Utilities Reference: System*.

Note: The XMLTrace class replaces the IMSTrace class. However, applications that use the IMSTrace class will still function properly.

WebSphere Application Server Security Requirements for XML Tracing

Before you can trace your application that runs on WebSphere Application Server V5.0 for z/OS or WebSphere Application Server V5.0 on a non-z/OS platform, you must add permissions to the WebSphere Application Server for z/OS server.policy file and create a was.policy for the application EAR file.

To add permissions to the WebSphere Application Server for z/OS server.policy file:

1. Open the WebSphere Application Server for z/OS server.policy file, which is in the properties directory of the WebSphere Application Server installation directory, and find the following code, which was added when you installed the custom service (if this code is not in the file, add it):

```
grant codeBase "file:/imsjava/-" {
      permission java.util.PropertyPermission "*", "read, write";
      permission java.lang.RuntimePermission "loadLibrary.JavTDLI";
      permission java.io.FilePermission "/tmp/*", "read, write";
};
```

2. Below `permission java.io.FilePermission "/tmp/*", "read, write";`, add the following permission, replacing `traceOutputDir` with the directory name for the trace output file:

```
permission java.io.FilePermission "/traceOutputDir/*", "read, write";
```

To create the was.policy file:

1. Create a new file named was.policy that contains the following code, replacing `traceOutputDir` with the directory name for the trace output file:

```
grant codeBase "file:${application}" {
  permission java.io.FilePermission "/traceOutputDir/*", "read, write";
};
```

2. Put the was.policy file in the META-INF directory of your application's EAR file.

Enabling XML Tracing

To debug with `XMLTrace`, you must first turn on the tracing function by calling one of the `XMLTrace.enable` methods. Because tracing does not occur until this variable is set, it is best to do so within a static block of your main application class. Then, you must decide how closely you want to trace the IMS Java library's flow of control and how much tracing you want to add to your application code.

You can determine the amount of tracing in the IMS Java library by providing the trace level in the `XMLTrace.enable` method. By default, this value is set to `XMLTrace.TRACE_EXCEPTIONS`, which traces the construction of IMS Java-provided exceptions. `XMLTrace` also defines constants for three types of additional tracing. These constants provide successively more tracing from `IMSTrace.TRACE_CTOR1` (level-one tracing of constructions) to `IMSTrace.TRACE_DATA3` (level-three tracing of data).

`XMLTrace` has the following trace levels:

Trace level	Description
`TRACE_EXCEPTIONS`	Traces exceptions
`TRACE_CTOR1`	Traces level-1 constructors
`TRACE_METHOD1`	Traces level-1 parameters, return values, methods, and constructors

TRACE_DATA1	Traces level-1 parameters, return values, methods, and constructors
TRACE_CTOR2	Traces level-2 constructors
TRACE_METHOD2	Traces level-2 parameters, return values, methods, and constructors
TRACE_DATA2	Traces level-2 parameters, return values, methods, and constructors
TRACE_CTOR3	Traces level-3 constructors
TRACE_METHOD3	Traces level-3 parameters, return values, methods, and constructors
TRACE_DATA3	Traces level-3 parameters, return values, methods, and constructors

Tracing the IMS Java Library Methods

To enable the tracing that is shipped with IMS Java library methods:

1. Call the `XMLTrace.enable` method and specify the root element name and the trace level. For example:

   ```
   XMLTrace.enable("MyTrace", XMLTrace.TRACE_METHOD1);
   ```

2. Set an output stream (a print stream or a character output writer) as the current trace stream. For example:
   a. Set the system error stream as the current trace stream:

      ```
      XMLTrace.setOutputStream(System.err);
      ```

   b. Set a `StringWriter` object (or any other type of writer) as the current trace stream:

      ```
      StringWriter stringWriter = new StringWriter();
      XMLTrace.setOutputWriter(stringWriter);
      ```

3. Close the XML trace:

   ```
   XMLTrace.close();
   ```

Steps 1 and 2 are best implemented within a static block of your main application class, as shown in Figure 44.

```
public static void main(String args[]){
    static {
        XMLTrace.enable("MyTrace", XMLTrace.TRACE_METHOD1);
        XMLTrace.setOutputStream(System.err);
    }
}
```

Figure 44. Setting a Trace within a Static Method

Tracing Your Application

You can add trace statements to your application, similar to those provided by the IMS Java library, by defining an integer variable that you test prior to writing trace statements. Using a variable other than `XMLTrace.libTraceLevel` enables you to control the level of tracing in your application independently of the tracing in the IMS Java library. For example, you can turn off the tracing of IMS Java Library routines by setting `XMLTrace.libTraceLevel` to zero, but still trace your application code.

To enable tracing for your application:

1. Define an integer variable to contain the trace level for application-provided code:

   ```
   public int applicationTraceLevel = XMLTrace.TRACE_CTOR3;
   ```

2. Set up the `XMLTrace` method to trace methods, parameters, and return values as necessary.

Debugging an Unresettable JVM in a JMP or JBP Region

If you need to debug reset trace events for the persistent reusable JVM in a JBP or JMP region, you need to run the JVM in debug mode. The following messages in your job log indicate that you should run in debug mode to determine the problem:

```
DFSJVM00: ResetJavaVM() RC=-1
DFSJVM00: DestroyJavaVM() RC=-1
```

To run the JVM in debug mode, add `DEBUG=Y` to the DFSJVMEV sample member, or the member that is specified by the DFSJMP or DFSJBP ENVIRON= parameter.

Related Reading:

- For more information about running the JVM in debug mode, see *IBM Developer Kit for OS/390, Java 2 Technology Edition: New IBM Technology featuring Persistent Reusable Java Virtual Machines*.
- For more information about the DFSJVMEV member and the ENVIRON= parameter, see *IMS Version 9: Installation Volume 2: System Definition and Tailoring*.

Preparing to Run the Dealership Samples

To run the dealership sample, you must prepare IMS by modifying the stage 1 input statements and loading the databases.

An IMS Java metadata class, which is a Java class that describes the IMS databases, is required for all applications. The IMS Java metadata class for the dealership sample applications, `AUTPSB11DatabaseView`, is provided compiled and is included with the application files. You do not have to do anything further to prepare this file.

The following topics provide additional information:

- "Modifying IMS Stage 1 Input Statements"
- "Loading the Dealership Sample Databases"

Modifying IMS Stage 1 Input Statements

Before you can access the sample dealership databases with the sample applications, you must modify the IMS system definition stage 1 input statements.

To modify the stage 1 input statements:

1. Add the following DATABASE macro statements to the IMS stage 1 input statements:

   ```
   DATABASE DBD=AUTODB,ACCESS=UP
   DATABASE DBD=EMPDB2,ACCESS=UP
   DATABASE DBD=SINDEX11,ACCESS=UP
   DATABASE DBD=SINDEX22,ACCESS=UP
   ```

2. Add a APPLCTN macro statement to the IMS stage 1 input statements for the sample application's program resource requirements. The sample applications use AUTPSB11 as the PSB. All sample applications require an APPLCTN statement for the AUTPSB11 PSB. For example:

   ```
   APPLCTN PSB=AUTPSB11,PGMTYPE=TP,SCHEDTYP=PARALLEL
   ```

3. If you are running the JMP version of the dealership sample application, add the TRANSACT macro statement following the APPLCTN macro statement. The JMP dealership sample application is nonconversational. For example:

   ```
   TRANSACT CODE=AUTRAN11,PRTY=(7,10,2),INQUIRY=NO,MODE=SNGL,   X
            MSGTYPE=(SNGLSEG,NONRESPONSE,1)
   ```

Loading the Dealership Sample Databases

To run the sample dealership applications, you must first load the databases that the applications access. The files that are needed to load these databases are in the samples directory: *pathprefix*usr/lpp/ims/imsjava91/samples/dealership/databases. To use these files, however, you must move them from HFS files to PDS members. The following steps provide sample jobs that move the files. If you use these sample jobs, replace *path* with *pathprefix*`usr/lpp/ims/imsjava91/samples/dealership/databases`.

To load the dealership sample databases:

1. Move the following DBD source files (HFS) to your DBD source library (PDS members):
   - AUTODB (physical DBD of the auto database)
   - EMPDB2 (physical DBD of the employee database)

- SINDEX11 (first secondary index)
- SINDEX22 (second secondary index)
- AUTOLDB (logical DBD of the auto database)
- EMPLDB2 (logical DBD of the employee database)

The following sample job moves these DBDs to PDS members:

```
//name     JOB parameters
//MV2PDS1  EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=*
//SYSTSNT  DD SYSOUT=*
//O1       DD DISP=SHR,DSN=hlq.dbdsrc(AUTODB)
//I1       DD DISP=SHR,PATH='path/AUTODB'
//O2       DD DISP=SHR,DSN=hlq.dbdsrc(EMPDB2)
//I2       DD DISP=SHR,PATH='path/EMPDB2'
//O3       DD DISP=SHR,DSN=hlq.dbdsrc(SINDEX11)
//I3       DD DISP=SHR,PATH='path/SINDEX11'
//O4       DD DISP=SHR,DSN=hlq.dbdsrc(SINDEX22)
//I4       DD DISP=SHR,PATH='path/SINDEX22'
//O5       DD DISP=SHR,DSN=hlq.dbdsrc(AUTOLDB)
//I5       DD DISP=SHR,PATH='path/AUTOLDB'
//O6       DD DISP=SHR,DSN=hlq.dbdsrc(EMPLDB2)
//I6       DD DISP=SHR,PATH='path/EMPLDB2'
//SYSTIN  DD*
OCOPY INDD(I1) OUTDD(01)
OCOPY INDD(I2) OUTDD(02)
OCOPY INDD(I3) OUTDD(03)
OCOPY INDD(I4) OUTDD(04)
OCOPY INDD(I5) OUTDD(05)
OCOPY INDD(I6) OUTDD(06)
/*
```

2. Generate the DBDs using the DBDGEN utility:
   a. Move the JCL file named AUTDBD to a partitioned data set from which it can be run.
   b. Edit the JCL statements as necessary.
   c. Run the job, which executes the DBDGEN procedure.
3. Move the following PSB source files (HFS) to your PSB source library (PDS members):
   - AUTPSBAL (for loading the auto database)
   - AUTPSBEL (for loading the employee database)
   - AUTPSB11 (for processing the databases)

   The following example moves these PSBs to PDS members:

```
//name     JOB parameters
//MV2PDS2  EXEC PGM=IKJEFT01
//SYSPRINT DD SYSOUT=*
//SYSTSNT  DD SYSOUT=*
//O1       DD DISP=SHR,DSN=hlq.psbsrc(AUTPSBAL)
//I1       DD DISP=SHR,PATH='path/AUTPSBAL'
//O2      DD DISP=SHR,PATH='path/AUTPSBEL'
//O3       DD DISP=SHR,DSN=hlq.psbsrc(AUTPSB11)
//I3       DD DISP=SHR,PATH='path/AUTPSB11'
//SYSTIN  DD*
OCOPY INDD(I1) OUTDD(01)
OCOPY INDD(I2) OUTDD(02)
OCOPY INDD(I3) OUTDD(03)
/*
```

4. Generate the PSBs by using the PSBGEN utility:
   a. Move the JCL file named AUTPSB to a partitioned data set from which it can be run.
   b. Edit the JCL statements if necessary.

c. Run the job, which executes the PSBGEN procedure.

5. Generate the ACBs for the IMS system that are used when running the sample application:
   a. Ensure that DFSACBCP is available in a partitioned data set.
   b. Move the JCL file named AUTACB to a partitioned data set from which it can be run.
   c. Edit the JCL statements if necessary.
   d. Run the job, which executes the ACBGEN procedure.
6. Initial load the databases:
   a. Move the JCL files named AUTLOAD and IV3H103A to a partitioned data set from which they can be run.

      The following sample job moves AUTLOAD and IV3H103A to PDS members:

      ```
      //name     JOB parameters
      //MV2PDS3  EXEC PGM=IKJEFT01
      //SYSPRINT DD SYSOUT=*
      //SYSTSNT  DD SYSOUT=*
      //O1       DD DISP=SHR,DSN=hlq.library(AUTLOAD)
      //I1       DD DISP=SHR,PATH='path/AUTLOAD'
      //O2       DD DISP=SHR,DSN=hlq.library(IV3H103A)
      //I2      DD DISP=SHR,PATH='path/IV3H103A'
      //SYSTIN  DD*
      OCOPY INDD(I1) OUTDD(01)
      OCOPY INDD(I2) OUTDD(02)
      /*
      ```

   b. Edit the JCL statements by adding the high-level qualifiers to the data set names and to the volume information, and my makinga any other necessary changes required by your installation.
   c. Run the AUTLOAD job, which is an IMS batch job. System data sets must be available and the control region must not be running. This job completes the following steps:
      - Scratches old database data sets.
      - Allocates new database data sets.
      - Loads the physical AUTDB and EMPDB2 databases.
      - Resolves and updates logical relationships.
      - Builds the two secondary indexes.

      Because no data exists in the databases yet, the final three steps are null operations and therefore, 0004 return codes are acceptable.
7. Add data to the initialized databases:
   a. Move the JCL file named AUTSEED to a partitioned data set from which it can be run.

      The following sample job moves AUTSEED to a PDS member:

      ```
      //name     JOB parameters
      //MV2PDS4  EXEC PGM=IKJEFT01
      //SYSPRINT DD SYSOUT=*
      //SYSTSNT  DD SYSOUT=*
      //O1       DD DISP=SHR,DSN=hlq.library(AUTSEED)
      //I1       DD DISP=SHR,PATH='path/AUTSEED'
      //SYSTIN  DD*
      OCOPY INDD(I1) OUTDD(01)
      /*
      ```

   b. Edit the JCL statements in AUTSEED if necessary.

c. Run the AUTSEED job, which executes the DFSDDLT0 procedure. This job completes the following steps:
   - Deletes the root segments, if present.
   - Adds roots and dependent segments to the database using the AUTPSB11 PSB.

   You can run this job repeatedly without re-running the AUTLOAD job.

8. Optionally, confirm that the databases loaded correctly:
   a. Move the JCL file named AUTLIST to a partitioned data set from which it can be run.

      The following sample job moves AUTLIST to a PDS member:

      ```
      name     JOB parameters
      //MV2PDS5  EXEC PGM=IKJEFT01
      //SYSPRINT DD SYSOUT=*
      //SYSTSNT  DD SYSOUT=*
      //O1       DD DISP=SHR,DSN=hlq.library(AUTLIST)
      //I1       DD DISP=SHR,PATH='path/AUTLIST'
      //SYSTIN  DD*
      OCOPY INDD(I1) OUTDD(01)
      /*
      ```

   b. Edit the JCL statements if necessary.
   c. Run the job, which executes the DFSDDLT0 procedure. This job lists segments to the database using the AUTPSB11 PSB.
9. Compile the dynamic allocation members for the databases:
   a. Move the JCL file named AUTODA to a partitioned data set from which it can be run.

      The following sample job moves AUTODA to a PDS member:

      ```
      //name     JOB parameters
      //MV2PDS6  EXEC PGM=IKJEFT01
      //SYSPRINT DD SYSOUT=*
      //SYSTSNT  DD SYSOUT=*
      //O1       DD DISP=SHR,DSN=hlq.library(AUTODA)
      //I1       DD DISP=SHR,PATH='path/AUTODA'
      //SYSTIN  DD*
      OCOPY INDD(I1) OUTDD(01)
      /*
      ```

   b. Edit the JCL statements if necessary.
   c. Run the job.

SQL Keywords

Because the IMS Java SQL parser supports portable SQL, you cannot use any SQL keywords as Java aliases for PCBs, segments, or fields. When you define Java aliases, do not use an SQL keyword. If a PCB, segment, or field has the same name as an SQL keyword, you must explicitly define a different Java alias for it. For information on defining Java aliases, see *IMS Version 9: Utilities Reference: System*.

If you use an SQL keyword as a name of a PCB, segment, or field, your application program receives an error when it attempts an SQL query.

The following SQL keywords cannot be used as PCB, segment, or field names:

ABORT	CROSS	IS	REAL
ANALYZE	CURRENT	JOIN	REFERENCES
AND	CURSOR	LAST	RESET
ALL	DECIMAL	LEADING	REVOKE
ALLOCATE	DECLARE	LEFT	RIGHT
ALTER	DEFAULT	LIKE	ROLLBACK
AND	DELETE	LISTEN	SELECT
ANY	DESC	LOAD	SET
ARE	DISTINCT	LOCAL	SETOF
AS	DO	LOCK	SHOW
ASC	DOUBLE	MAX	SMALLINT
ASSERTION	DROP	MIN	SUBSTRING
AT	END	MOVE	SUM
AVG	EXECUTE	NAMES	TABLE
BEGIN	EXISTS	NATIONAL	TO
BETWEEN	EXPLAIN	NATURAL	TRAILING
BINARY	EXTRACT	NCHAR	TRANSACTION
BIT	EXTEND	NEW	TRIM
BOOLEAN	FALSE	NO	TRUE
BOTH	FIRST	NONE	UNION
BY	FLOAT	NOT	UNIQUE
CASCADE	FOR	NOTIFY	UNLISTEN
CAST	FOREIGN	NULL	UNTIL
CHAR	FROM	NUMERIC	UPDATE
CHARACTER	FULL	ON	USER
CHECK	GRANT	OR	USING
CLOSE	GROUP	ORDER	VACUUM
CLUSTER	HAVING	OUTER	VALUES
COLLATE	IN	PARTIAL	VARCHAR
COLUMN	INNER	POSITION	VARYING
COMMIT	INSERT	PRECISION	VERBOSE
CONSTRAINT	INT	PRIMARY	VIEW
COPY	INTEGER	PRIVILEGES	WHERE
COUNT	INTERVAL	PROCEDURE	WITH
CREATE	INTO	PUBLIC	WORK

IMS Java Hierarchical Database Interface

The IMS Java hierarchical database interface is more closely related to the standard DL/I database call interface that is used with other languages, and provides a lower-level access to IMS database functions than the JDBC interface. Using IMS Java hierarchical database interface, you can build segment search arguments (SSAs) and call the functions of the `DLIConnection` object to read, insert, update, or delete segments. The application has full control to navigate the segment hierarchy.

Although you can use the IMS Java hierarchical database interface to access IMS data, it is recommended that you use JDBC. However, you can use this package if you need more controlled access than the higher-level JDBC package provides.

Related Reading: For detailed information about the classes in the IMS Java hierarchical database interface, see the IMS Java API Specification (Javadoc). Go to the IMS Web site at www.ibm.com/ims and link to the IMS Java page.

The following topics provide additional information:

- "Application Programming Using the DLIConnection Object"
- "Creating a DLIConnection Object"
- "Creating an SSAList Object" on page 172
- "Accessing IMS Data Using SSAs" on page 172

Application Programming Using the DLIConnection Object

To use a `DLIConnection` object to read, update, insert, and delete segment instances, your application must:

1. Acquire a `DLISegment` object for each segment using the `cloneSegment` method on the `DLIDatabaseView` subclass.
2. Provide a subclass of `DLIDatabaseView` that defines the segment hierarchy accessed by the application.
3. Create a `DLIConnection` object to access the database.
4. Create an `SSAList` object.
5. Invoke the database access methods of the `DLIConnection` class to read, write, or delete segments from the database.

Create the required classes by running the DLIModel utility (see *IMS Version 9: Utilities Reference: System*).

Creating a DLIConnection Object

You must create a`DLIConnection` object in one of two ways:

- By providing a `DLIDatabaseView` object
- By providing the fully-qualified name of the `DLIDatabaseView` subclass

When you code directly to a `DLIConnection` object, it is faster to create and pass the `DLIDatabaseView` object because it simplifies finding the class by its name. Figure 45 on page 172 illustrates how to create a `DLIConnection` object:

```
DealerDatabaseView dealerView = new DealerDatabaseView();
DLIConnection connection = DLIConnection.createInstance(dealerView);
```

Figure 45. Creating a DLIConnection Object

Creating an SSAList Object

SSAs identify the segment to which a DL/I call applies. Because of the hierarchical structure of IMS databases, you often have to specify several levels of SSAs to access a segment at a low level in the hierarchy. An `SSAList` object is a collection of one or more `SSA` objects. Use the `SSAList` object when you make DL/I calls. The `SSAList` object is also where you specify which database that you want to access within a `DLIDatabaseView` object by providing the PCB reference name.

Figure 46 shows how to create an `SSAList` object that will find all “Alpha” cars that were made in 2004:

```
// Create an SSAList
SSAList modelSSAList = SSAList.createInstance("DealershipDB");

// Construct an unqualified SSA for the Dealer segment
SSA dealerSSA = SSA.createInstance("Dealer");

// Construct a qualified SSA for the Model segment
SSA modelSSA = SSA.createInstance("Model", "CarMake", SSA.EQUALS, "Alpha");
// Add an additional qualification statement
modelSSA.addQualification(SSA.AND, "CarYear", SSA.EQUALS, "1989");

// Add the SSAs to the SSAList
modelSSAList.addSSA(dealerSSA);
modelSSAList.addSSA(modelSSA);
```

Figure 46. Creating an SSAList Object

Accessing IMS Data Using SSAs

After you create an `SSAList` object, you can issue database calls by invoking the access method on the `DLIConnection` object and passing in the following:

- The `SSAList` object.
- An instance of the segment, which is the intended target of the database call results.

Get the passed-in instance of the segment by calling the `cloneSegment` method on the `DLIDatabaseView` subclass.

The following example how to call and print the results using the `SSAList` object that was built in “Creating an SSAList Object”:

```
DLISegment model = dealerView.cloneSegment("Model");
    boolean recordRead = connection.getUniqueSegment(model, modelSSAList);
    while (recordRead) {
        System.out.println("Car Name: " + model.getString("ModelName"));
        recordRead = connection.getNextSegment(model, modelSSAList);
    }
```

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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs

and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
J46A/G4
555 Bailey Avenue
San Jose, CA 95141-1003
U.S.A.

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

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

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

BookManager	OS/390
DB2	QMF
CICS	RACF
IBM	Rational Rose
IMS	WebSphere
IMS/ESALibrary Reader	z/OS
MVS	

Java and all Java-based trademarks and logos are 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.

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

Bibliography

This bibliography includes all the publications cited in this book, including the publications in the IMS library.

DB2 Universal Database for OS/390 and z/OS: Application Programming Guide and Reference for Java, SC26-9932

Enterprise COBOL for z/OS and OS/390: Programming Guide, SC27-1412

IBM Developer Kit for OS/390, Java 2 Technology Edition: New IBM Technology featuring Persistent Reusable Java Virtual Machines, SC34-6034

z/OS: UNIX System Services Command Reference, SA22-7802

z/OS: UNIX System Services File System Interface Reference, SA22-7808

z/OS: UNIX System Services User's Guide, SA22-7801

CICS Transaction Server for z/OS: CICS System Definition Guide, SC34-5988

WebSphere Application Server V4.0.1 for z/OS and OS/390 : Assembling Java 2 Platform, Enterprise Edition (J2EE) Applications, SA22-7836

WebSphere Application Server V4.0.1 for z/OS and OS/390 : System Management User Interface, SA22-7838

IMS Version 9 Library

Title	Acronym	Order number
IMS Version 9: Administration Guide: Database Manager	ADB	SC18-7806
IMS Version 9: Administration Guide: System	AS	SC18-7807
IMS Version 9: Administration Guide: Transaction Manager	ATM	SC18-7808
IMS Version 9: Application Programming: Database Manager	APDB	SC18-7809
IMS Version 9: Application Programming: Design Guide	APDG	SC18-7810
IMS Version 9: Application Programming: EXEC DLI Commands for CICS and IMS	APCICS	SC18-7811
IMS Version 9: Application Programming: Transaction Manager	APTM	SC18-7812
IMS Version 9: Base Primitive Environment Guide and Reference	BPE	SC18-7813
IMS Version 9: Command Reference	CR	SC18-7814
IMS Version 9: Common Queue Server Guide and Reference	CQS	SC18-7815
IMS Version 9: Common Service Layer Guide and Reference	CSL	SC18-7816
IMS Version 9: Customization Guide	CG	SC18-7817
IMS Version 9: Database Recovery Control (DBRC) Guide and Reference	DBRC	SC18-7818
IMS Version 9: Diagnosis Guide and Reference	DGR	LY37-3203
IMS Version 9: Failure Analysis Structure Tables (FAST) for Dump Analysis	FAST	LY37-3204
IMS Version 9: IMS Connect Guide and Reference	CT	SC18-9287
IMS Version 9: IMS Java Guide and Reference	JGR	SC18-7821
IMS Version 9: Installation Volume 1: Installation Verification	IIV	GC18-7822
IMS Version 9: Installation Volume 2: System Definition and Tailoring	ISDT	GC18-7823
IMS Version 9: Master Index and Glossary	MIG	SC18-7826
IMS Version 9: Messages and Codes, Volume 1	MC1	GC18-7827
IMS Version 9: Messages and Codes, Volume 2	MC2	GC18-7828
IMS Version 9: Open Transaction Manager Access Guide and Reference	OTMA	SC18-7829
IMS Version 9: Operations Guide	OG	SC18-7830
IMS Version 9: Release Planning Guide	RPG	GC17-7831
IMS Version 9: Summary of Operator Commands	SOC	SC18-7832
IMS Version 9: Utilities Reference: Database and Transaction Manager	URDBTM	SC18-7833
IMS Version 9: Utilities Reference: System	URS	SC18-7834

Supplementary Publications

Title	Order number
IMS Connector for Java 2.2.2 and 9.1.0.1 Online Documentation for WebSphere Studio Application Developer Integration Edition 5.1.1	SC09-7869
IMS Version 9 Fact Sheet	GC18-7697
IMS Version 9: Licensed Program Specifications	GC18-7825

Publication Collections

Title	Format	Order number
IMS Version 9 Softcopy Library	CD	LK3T-7213
IMS Favorites	CD	LK3T-7144
Licensed Bill of Forms (LBOF): IMS Version 9 Hardcopy and Softcopy Library	Hardcopy and CD	LBOF-7789
Unlicensed Bill of Forms (SBOF): IMS Version 9 Unlicensed Hardcopy Library	Hardcopy	SBOF-7790
OS/390 Collection	CD	SK2T-6700
z/OS Software Products Collection	CD	SK3T-4270
z/OS and Software Products DVD Collection	DVD	SK3T-4271

Accessibility Titles Cited in This Library

Title	Order number
z/OS V1R1.0 TSO Primer	SA22-7787
z/OS V1R5.0 TSO/E User's Guide	SA22-7794
z/OS V1R5.0 ISPF User's Guide, Volume 1	SC34-4822

Index

A

aggregate functions 136
  AS 136
  ASC 136
  AVG 136
  COUNT 136
  data types 137
  DESC 136
  GROUP BY 136
  MAX 136
  MIN 136
  ORDER BY 136
  renaming 137
  result set column 137
  result set type 137
  SUM 136
AND operator
  IMS rules 135
Apache open source XML libraries
  about 3
  downloading 3
applications
  message processing, building 17
  programming 171
AS keyword 136, 137
ASC keyword 136
asterisk operator 131
AVG keyword 136

B

BIGINT data type 142
BINARY data type 142
BIT data type 142
byte data type 142

C

CHAR data type 142
CICS
  _CXX_LSYSLIB environment variable 119
  application
    sample 121
  applications
    IVP 120
    running 122
    writing 123
  CICSPSB DLL 119
  configuring for IMS Java 119
  DFHJVMPR environment member 120
  dfjjvmpr.props 120
  IMS Java overview 119
  IVP 120
  LIBPATH variable 120
  Makefile 119
  running applications 122
  sample application 121
CICS *(continued)*
  writing applications 123
CLASSPATH ENVAR keyword 112
Clob interface
  result set 139
  retrieveXML 138
COBOL
  See also Enterprise COBOL
  copybook types 144
  mapping to IMS 144
columns
  fields, compared to 125
  relational representation, in 126
com.ibm.connector2.ims.db package 5
com.ibm.ims.application package 5
com.ibm.ims.base package 5
com.ibm.ims.db package 5
com.ibm.ims.db.DLIDatabaseView class 146
com.ibm.ims.rds package 5
com.ibm.ims.rds.host package 6
com.ibm.ims.rds.util package 6
com.ibm.ims.xms package 6
Connection object 148
conversational transactions 22
copybook types 144
COUNT keyword 136
custom service, installing 41, 43

D

data types
  aggregate functions 137
  conversion 142
  mapped to COBOL 144
databases
  describing to IMS Java 146
DATE data type 142
DB2 Recoverable Resource Manager Services
 attachment facility 16
DB2 UDB for z/OS access
  application programming 34
  committing work 34
  drivers 34
  FSDB2AF DD statement 17
  IMS databases, compared to 34
  JBP region, from a
    configuring 16
    programming model 30
  JMP region, from a
    configuring 16
    programming model 21
  rolling back work 34
  RRSAF 16
DB2 UDB for z/OS stored procedures
  developing 118
  environment variables 112
  IVP 113
  JAVAENV data set 111

DB2 UDB for z/OS stored procedures *(continued)*
overview with IMS Java 111
running 116
sample application
running 115
system requirements 111
DB2_HOME ENVAR keyword 112
DB2Auto application
running 115
DB2AutoClient application
running 115
db2sqljjdbc.properties 115
DBD (database description), sample 146
DD statements
DFSDB2AF 17
DFSRESLB 113
JAVAERR 11
JAVAOUT 11
STEPLIB 11
dealership sample application
DB2 UDB for z/OS stored procedures 115
dealership samples
DBD 146
PSB 146
DEBUG=Y
for JVM debugging 163
debugging
XMLTrace 160
DELETE keyword 133
example 133
deployment descriptor
example 73
requirements for IMS Java 72
DESC keyword 136
dfjjvmpr.props file 120
DFSJMP procedure 11
DFSJVMEV member
DB2 JDBC driver 17
IVP changes 11
DFSJVMMS
DB2 JDBC driver 17
DFSJVMMS member
IVP changes 11
DFSRESLB DD statement 113
DL/I data, accessing 172
DL/I status codes
mapping to exceptions 159
DLIConnection
creating 171
DLIConnection class 159
DLIDatabaseView class 146
DLIDriver
loading 148
registering 149
DLIException class 159
DLIModel IMS Java Report 147
DLIModel utility
DLIModel IMS Java Report 147
using 146
DLITypeInfoList class 25
DOUBLE data type 142
driver
registering with DriverManager 149
DriverManager facility 140

E

EJB (Enterprise JavaBean)
deployment descriptor requirements 72
Enterprise COBOL 31
back end 31
implementing 32
CALL statement 31
compiler 32
front end 31, 32
JVM, locating 33
main method 32
object oriented syntax 31
performance 33
exceptions
description 159
IMSException object
getAIB method 159
getFunction method 159
getStatusCode method 159
mapping to DL/I status codes 159

F

fields
columns, compared to 125
default value 133
in SQL queries 128
segment qualified 132
float data type 142
FROM keyword 134
joins 134
FSDB2AF DD statement 17

G

getAIB method 159
getFunction method 159
getNextException 160
getStatusCode method 159
GROUP BY keyword 136

H

HFS (Hierarchic File System)
allocating data set for 2
mounting directory 2
Hierarchic File System (HFS)
allocating data set for 2
mounting directory 2
hierarchical database
example 126
relational database, compared to 125
High Performance Java (HPJ) 6
HPJ (High Performance Java) 6

I

IBM Developer Kit for OS/390, Java 2 Technology Edition 1
importing packages 149
IMS distributed JDBC resource adapter, installing 81, 83
IMS Java
  administering 4
  class library 5
  data type support 142
  exceptions 159
  IMS Java API Specification 6
  installing 2
    DFSJSMKD job 2
    DFSJSMKDR REXX script 2
    HFS data set 2
    HFS mount point 2
    SMP/E 2
  Javadoc 6
  JDBC application 148
  JDBC support 1
  overview 1
  packages 5
    com.ibm.connector2.ims.db 5
    com.ibm.ims.application 5
    com.ibm.ims.base 5
    com.ibm.ims.db 5
    com.ibm.ims.rds 5
    com.ibm.ims.rds.host 6
    com.ibm.ims.rds.util 6
    com.ibm.ims.xms 6
    IMS Java API Specification 6
    Javadoc 6
  problem determination 159
  Redbooks 6
  remote database services
    about 75
    components 75
  restrictions 6
  supported environments 1
  system requirements 1
IMS Java API Specification 6
IMS Java hierarchical database interface
  about 1
  using 171
IMS JDBC resource adapter, installing 40, 43
IMSException class 159
IMSFieldMessage 26
IMSFieldMessage class
  subclassing 18
IMSMessageQueue 160
input messages, defining 18
INSERT keyword 133
  example 133
  WHERE clause 133
installation verification programs (IVPs)
  CICS 120
  DB2 UDB for z/OS stored procedures 113
  JBP region 13
  JMP region 10
  WebSphere Application Server (non-z/OS) 84, 88
installation verification programs (IVPs) *(continued)*
  WebSphere Application Server for z/OS 45, 48
installing IMS Java
  DFSJSMKD job 2
  DFSJSMKDR REXX script 2
  HFS data set 2
  HFS mount point 2
  SMP/E 2
int data type 142
INTEGER data type 142
IVPs (installation verification programs)
  CICS 120
  DB2 UDB for z/OS stored procedures 113
  JBP region 13
  JMP region 10
  WebSphere Application Server (non-z/OS) 84, 88
  WebSphere Application Server for z/OS 45, 48

J

Java batch processing (JBP) regions i
  DB2 UDB for z/OS access
    application programming 34
    configuring 16
    programming model 30
  description 9
  IVP 13
  programming models 29
  restart 28
  symbolic checkpoint 28
Java Batch Processing (JBP) regions
  program switching 35
Java data types 142
Java message processing (JMP) regions i
  DB2 UDB for z/OS access
    application programming 34
    configuring 16
    programming model 21
  description 9
  IVP 10
    DFSJMP procedure 11
    DFSJVMEV member 11
    DFSJVMMS member 11
    JVM.out file 11
  programming models 20
Java Message Processing (JMP) regions
  program switching 35
JAVA_HOME ENVAR keyword 112
java.math.BigDecimal 142
java.sql.Clob
  See Clob interface
java.sql.Connection interface 140
java.sql.DatabaseMetaData interface 140
java.sql.Date 142
java.sql.Driver interface 140
java.sql.PreparedStatement interface 141
java.sql.ResultSet interface 141
java.sql.ResultSetMetaData interface 141
java.sql.Statement interface 141
java.sql.Time 142
java.sql.Timestamp 142

JAVAENV data set
  creating 111
  sample 112
JAVAERR DD statement 11
JAVAOUT DD statement 11
JBP (Java batch processing) regions i
  DB2 UDB for z/OS access
    application programming 34
    configuring 16
    programming model 30
  description 9
  IVP 13
  program switching 35
  programming models 29
  restart 28
  symbolic checkpoint 28
JDBC
  connecting to IMS database 149
  Connection object, returning 149
  data types 142
  explanation 125
  importing packages 149
  interfaces
    java.sql.Connection 140
    java.sql.DatabaseMetaData 140
    java.sql.Driver 140
    java.sql.PreparedStatement 141
    java.SQL.ResultSet 141
    java.sql.ResultSetMetaData 141
    java.sql.Statement 141
  interfaces, limitations 140
  jdbc:dli 149
  sample application 148
  SQL keywords, supported 128
  using 148
  writing an application 148
  XML, extension for 137
JDBC drivers
  DB2 JDBC/SQLJ 1.2 driver 16
  DB2 JDBC/SQLJ 2.0 driver 16
  DB2 Universal JDBC driver 16
JMP (Java message processing) regions i
  DB2 UDB for z/OS access
    application programming 34
    configuring 16
    programming model 21
  description 9
  IVP 10
    DFSJMP procedure 11
    DFSJVMEV member 11
    DFSJVMMS member 11
    JVM.out file 11
  program switching 35
  programming models 20
JMP applications
  message handling
    conversational transactions 22
    multi-segment messages 24
    multiple input messages 26
    repeating structures 25
  programming models 20
joining segments 131
JVM, debugging
  DEBUG=Y 163
  log messages 163
  reset trace events 163

L

LIBPATH ENVAR keyword 112
long data type 142

M

main() method 19
MAX keyword 136
message processing application
  building 17
messages
  input, defining 18
  multi-segment 24
  output, defining 18
  repeating structures
    defining in IMS Java 25
  SPA 22
  subsequent 24
MIN keyword 136
multi-segment messages 24

O

object
  DLIConnection, creating 171
OR operator
  IMS rules 135
ORDER BY keyword 136
output messages, defining 18

P

PACKEDDECIMAL data type 142
path call 131
Persistent Reusable Java Virtual Machine 1
prepared statements
  java.sql.PreparedStatement interface 141
PreparedStatement object 148
Problem Determination 159
program switches 35
programming models
  JBP applications
    symbolic checkpoint and restart 29
    with rollback 30
    without rollback 29
  JMP applications 20
    DB2 UDB for z/OS data access 21
    IMS data access 21
    with rollback 21
    without rollback 20
  WebSphere Application Server (non-z/OS) applications
    bean-managed EJBs 69

programming models *(continued)*
WebSphere Application Server (non-z/OS) applications *(continued)*
container-managed EJBs 71
servlets 71
WebSphere Application Server for z/OS applications
bean-managed EJBs 69
container-managed EJBs 71
servlets 71
PSB (program specification block)
sample 146

R

Recoverable Resource Manager Services attachment facility (RRSAF) 16
relational database
hierarchical database, compared to 125
remote database services
configuring 78, 81
repeating structures
accessing 25
DLITypeInfoList class 25
dotted notation 25
sample output message 25
res-sharing-scope element 72
res-type element 72
resource-ref element
example 73
requirements for IMS Java 72
ResultSet
aggregate data types 137
iterating 148
TYPE_FORWARD_ONLY 137
TYPE_SCROLL_INSENSITIVE 137
ResultSet.getAsciiStream method 141
ResultSet.getCursorName method 141
ResultSet.getUnicodeStream method 141
rows
relational representation, in 128
segment instances, compared to 125
RRSAF (Recoverable Resource Manager Services attachment facility) 16

S

sample applications
DB2 UDB for z/OS stored procedures 115
samples
message processing application 17
segments
in SQL queries 128
tables, compared to 125
segments, selecting multiple 131
SELECT keyword
asterisk operator 131
description 129
example 129
example query 128
retrieveXML 132
selecting all fields in a segment 131
SELECT keyword *(continued)*
selecting multiple segments 131
setModifiableAlternatePCB(String) method 35
short data type 142
SMALLINT data type 142
SPA 22
message, defining 22
SPA (scratch pad area) 22
conversational transactions for IMS Java 24
SQL (Structured Query Language)
aggregate functions 136
argument types 137
result types 137
AS 136, 137
ASC 136
AVG 136
COUNT 136
DELETE 133
DESC 136
example query 128
FROM 134
GROUP BY 136
IMS requirements for 128
INSERT 133
keyword list 169
MAX 136
MIN 136
ORDER BY 136
PCB-qualified query 134
prepared statements 142
recommendations
PCB-qualified query 134
segment-qualified fields 132
segment-qualified fields 132
SELECT keyword requirements 129
SUM 136
supported keywords 128
UPDATE 134
WHERE 135
SQLException 160
SQLstate 160
SSA (segment search argument) 131
WHERE clause, relation to 135
SSAList
creating an 172
DL/I data, accessing 172
Statement object
retrieving 148
status codes
mapping 159
stored procedures
See DB2 UDB for z/OS stored procedures
String data type
boolean data type 142
SUM keyword 136
syntax diagram
how to read xiv
system requirements
DB2 UDB for z/OS stored procedures 111

T

tables
  relational representation, in 126
  segments, compared to 125
TIME data type 142
TIMESTAMP data type 142
TINYINT data type 142
TMSUFFIX ENVAR keyword 112
tracing
  IMS Java library methods 162
  J2EE 63, 67, 107
  Trace statements, adding 162
  WebSphere Application Server for z/OS 63, 67, 107
  XMLTrace 160
transactions
  conversational 22
types
  data, mapped to COBOL 144
  supported 142

U

UPDATE keyword 134
  example 134

V

VARCHAR data type 142

W

WebSphere Application Server (non-z/OS)
  application, installing 86, 89
  applications
    IVP 84, 88
    sample 92, 96
  configuring
    data source, installing 78, 82
    EAR file, installing on server side 79, 83
    IMS distributed JDBC resource adapter, installing 81, 83
    prerequisites 78, 81
  data source, installing 85, 88, 101, 105
  deployment descriptor 72
  downloading IMS Java files 77
  EJB
    client side 109
    server side 109
  EJB considerations 108
  IVP 84, 88
    application, installing on the client side 86, 89
    data source, installing on the client side 85
    prerequisites 84, 88
    testing 87, 90
  running your application
    application, installing 102, 106
    data source, installing 101, 105
    prerequisites 100, 104
  sample application
    application, installing 93, 98
WebSphere Application Server (non-z/OS) *(continued)*
  sample application *(continued)*
    data source, installing 92, 96
    prerequisite 92, 96
    testing dealership sample 95, 99
    testing phonebook sample 95, 99
  tracing with XMLTrace 103
WebSphere Application Server for z/OS
  applications 37
    IVP 45, 48
    samples 52, 56
  classpath, setting 60, 65
  configuring
    access to IMS 39, 42
    custom service, installing 41, 43
    IMS JDBC resource adapter, installing 40, 43
    prerequisites 39, 42
  deployment descriptor 72
  IVP 45, 48
    application, installing 46, 50
    data source, installing 45, 49
    prerequisites 45, 48
    testing 48, 50
  overview 37
  restrictions
    container-managed signon 72
    java.sql.Connection object 72
    shared connections 72
  running your application
    application, installing 62, 67
    classpath, setting 60, 65
    data source, installing 60, 65
    prerequisites 60, 65
  sample applications
    application, installing 54, 57
    data source, installing 52, 56
    prerequisites 52, 56
    testing 55, 58
  server.policy file 38, 41, 44
  tracing 63, 67, 107
WHERE keyword 135
  fields, valid 135
  operators, valid 135
  SSAs, relation to 135

X

xalan.jar 3
Xalan–Java version 2.6.0 3
xercesImpl.jar 3
XML (Extensible Markup Language)
  composition 151
  data-centric documents 153
  decomposed storage mode 152
  IMS, and 151
  intact storage mode
    about 154
    base segment 154
    database for 154
    DBD example 155
    overflow segment 154

XML (Extensible Markup Language) *(continued)*
intact storage mode *(continued)*
side segment 155
JDBC extensions for 137
legacy databases, and 153
open source files for IMS Java 3
overview 151
retrieveXML 138
Clob interface 139
example 138
storeXML 139
example 140
SQL syntax 139
storing 151
supported environments 158
type representation 157
UDFs 137
xalan.jar 3
Xalan–Java version 2.6.0 3
xercesImpl.jar 3
xml-apis.jar 3
XML schema
data types 157
overview 157
xml-apis.jar 3
XMLTrace
application 162
enabling 161
WebSphere Application Server (non-z/OS) applications 103
XMLTrace class 160
XMLTrace.enable 161
XMLTrace.IMS Java library methods 162
XMLTrace.libTraceLevel 162

Z

ZONEDECIMAL data type 142

Program Number: 5655-J38

Printed in USA

SC18-7821-03

Spine information:

IMS IMS Java Guide and Reference Version 9