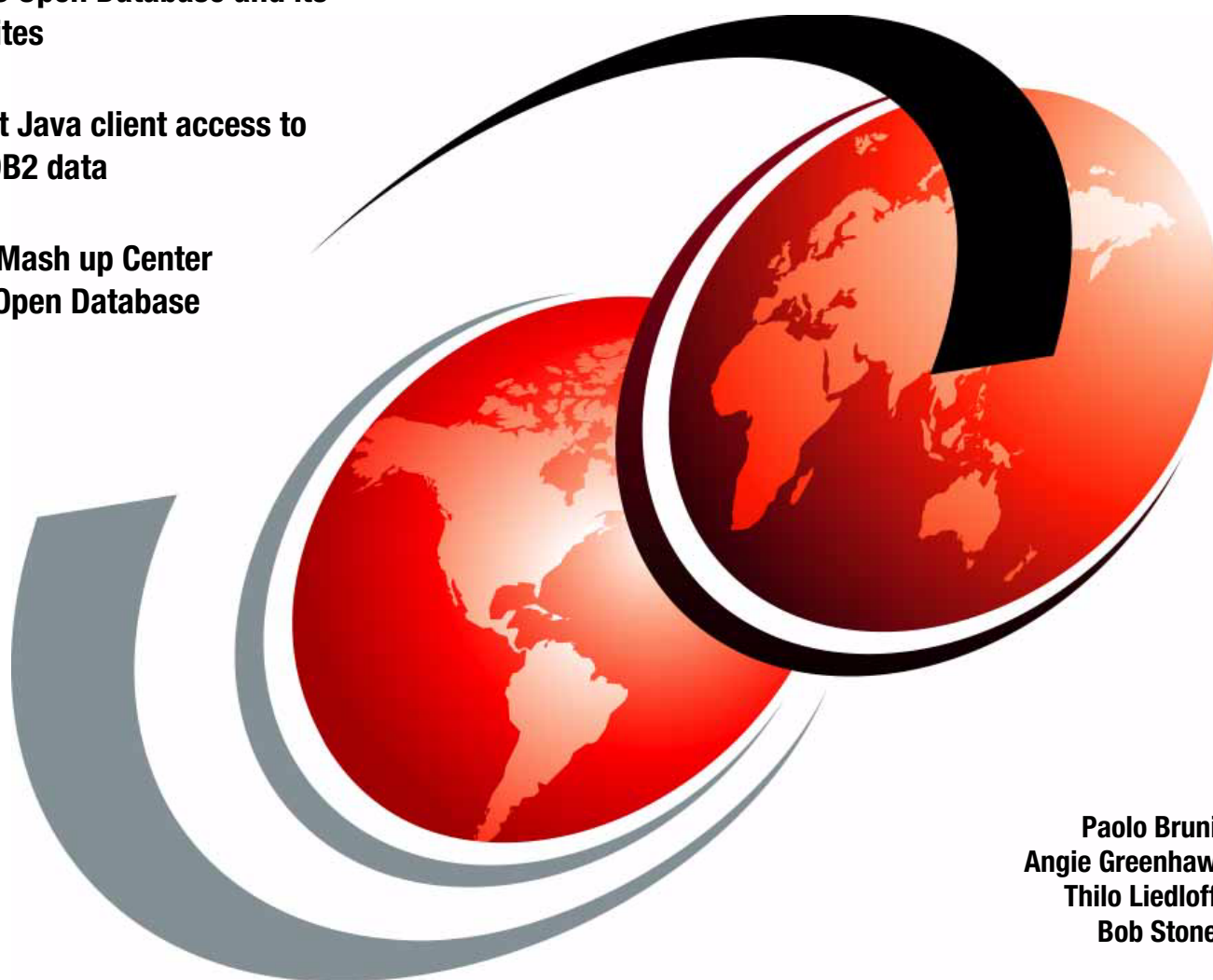


# IMS 11 Open Database

**Install IMS Open Database and its prerequisites**

**Implement Java client access to IMS and DB2 data**

**Integrate Mash up Center with IMS Open Database**



**Paolo Bruni  
Angie Greenhaw  
Thilo Liedloff  
Bob Stone**

**Redbooks**





International Technical Support Organization

**IMS 11: The Open Database**

May 2010

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

### **First Edition (May 2010)**

This edition applies to Version 11 of Information Management System (IMS) Transaction and Database Servers (program number 5635-A02) and IMS Enterprise Suite for z/OS, Version 1.1 (program numbers 5655-T60 and 5655-T61).

This document created or updated on May 4, 2010.

© Copyright International Business Machines Corporation 2010. All rights reserved.

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

# Contents

<b>Figures</b> .....	vii
<b>Tables</b> .....	xi
<b>Examples</b> .....	xiii
<b>Notices</b> .....	xv
Trademarks .....	xvi
<b>Preface</b> .....	xvii
The team who wrote this book .....	xvii
Acknowledgments .....	xviii
Now you can become a published author, too! .....	xix
Comments welcome .....	xix
<b>Chapter 1. Introduction</b> .....	1
1.1 IMS is open .....	2
1.2 IMS 10 and SOA .....	2
1.2.1 The value of including existing IMS assets into SOA .....	2
1.2.2 IMS Connect and IMS Connect Extensions .....	5
1.2.3 The IMS SOA Integration Suite .....	6
1.2.4 DataPower and IMS .....	9
1.3 IMS 11 and Open Database .....	10
1.4 The IMS Enterprise Suite .....	14
<b>Chapter 2. Open Database architecture</b> .....	17
2.1 Why IMS Open Database .....	18
2.2 Accessing IMS DB in Versions 9 and 10 .....	19
2.3 Evolution in IMS 11 .....	20
2.4 IMS 11 Architecture .....	23
2.5 Open Database functions .....	24
2.5.1 IMS Open Database uses DRDA .....	24
2.5.2 Open DataBase Manager .....	25
2.5.3 IMS Connect .....	28
2.5.4 Distributed sync pointing .....	33
2.5.5 Distributed Data Management .....	34
2.6 IMS 11 Universal Drivers .....	35
<b>Chapter 3. System environment</b> .....	41
3.1 Required environment setup for IMS Open Database .....	42
3.2 Common Service Layer components .....	42
3.2.1 Base Primitive Environment configuration .....	43
3.2.2 Structured Call Interface .....	45
3.2.3 Operations Manager .....	46
3.2.4 Open Database Manager .....	48
3.3 IMS Connect .....	59
3.3.1 First-time implementation: setup and configuration .....	60
3.3.2 Modifying existing IMS Connect definitions for IMS Open DB support .....	63
3.4 Using IMS applications to help set up CSL and IMS Connect .....	63
3.4.1 Installation Verification Program .....	63

3.4.2 IMS Syntax Checker .....	67
3.5 Security considerations .....	71

## **Chapter 4. Generating IMS metadata class with IMS Enterprise Suite DLIModel Utility**

75

4.1 Introduction .....	76
4.2 Overview of IMS Enterprise Suite DLIModel utility .....	76
4.2.1 Requirements .....	78
4.2.2 Restrictions .....	79
4.2.3 History .....	79
4.3 Download and installation .....	80
4.3.1 Installing the IBM Installation Manager .....	81
4.3.2 Installing the IMS Enterprise Suite DLIModel utility .....	83
4.4 Setup for sample scenarios included in this book .....	86
4.4.1 Downloading the Car Dealer IVP database source .....	86
4.5 Using the IMS Enterprise Suite DLIModel utility .....	86
4.5.1 Generating metadata for Car Dealer database .....	87
4.5.2 Editing the AUTPSB11 Project .....	90
4.5.3 Export the metadata as Jar file .....	92
4.6 Additional considerations for the IMS Enterprise Suite DLI Model Utility .....	93
4.6.1 Ensuring consistency between generated class files and other JRE files .....	93
4.6.2 Track changes of IMS database reorganizations .....	94
4.6.3 Integrate the DLIModel Utility with other Eclipse products .....	94
4.6.4 Data type conversion table .....	94

## **Chapter 5. IMS Open Database for application developers**

5.1 Overview of IMS Open Database on the application side .....	96
5.1.1 IMS Universal DB drivers .....	96
5.1.2 IMS database metadata .....	97
5.1.3 Java version requirements .....	98
5.2 Architectural considerations .....	98
5.2.1 Transactional support .....	98
5.2.2 Access types .....	99
5.2.3 Programming approach .....	102
5.2.4 Comparison of the IMS Universal drivers .....	103
5.3 IMS Universal Database resource adapter .....	104
5.3.1 JCA/Common Client Interface approach .....	105
5.3.2 JCA/JDBC approach .....	107
5.4 IMS Universal JDBC driver (Stand Alone) .....	109
5.4.1 Connecting to an IMS database using the JDBC DataSource interface .....	109
5.4.2 Connecting to an IMS database using the JDBC DriverManager interface .....	111
5.5 IMS Universal DL/I driver .....	112
5.5.1 Basic steps in writing a IMS Universal DL/I driver application .....	112
5.5.2 Example code using IMS Universal DL/I driver .....	113
5.6 SQL syntax for the IMS Universal drivers .....	113
5.6.1 SQL keywords .....	114
5.6.2 Primary key and virtual foreign key handling .....	115
5.6.3 Usage of SELECT statement .....	117
5.6.4 Usage of INSERT statement .....	118
5.6.5 Usage of UPDATE statement .....	118
5.6.6 Usage of DELETE statement .....	118
5.6.7 Usage of the WHERE statement .....	119
5.6.8 Usage of AGGREGATE functions .....	120
5.7 Data transformation support .....	121

5.7.1 JDBC data types to Java data types mapping .....	121
5.7.2 Compatible data transformation functions. ....	122
<b>Chapter 6. Scenario 1 - JDBC data access through tooling .....</b>	<b>125</b>
6.1 IBM Data Perspective in Data Studio and Rational products .....	126
6.1.1 Download and Install IBM Data Studio or Rational products .....	126
6.1.2 Configuring IBM Data Studio for use with the IMS Universal JDBC Driver. ....	127
6.1.3 Using the Data Perspective with the IMS Universal Drivers .....	131
6.2 Accessing IMS Data in Cognos. ....	137
6.2.1 IBM Cognos 8 Virtual View Manager .....	138
6.2.2 Configuring Virtual View Manager for IMS Data access. ....	141
6.3 Accessing IMS Data Using the IBM Mashup Center. ....	146
<b>Chapter 7. Scenario 2 - Developing JDBC applications .....</b>	<b>153</b>
7.1 Developing a stand alone Java application using the IMS Universal JDBC Driver. . .	154
7.1.1 Prerequisites .....	154
7.1.2 Creating and configuring a new Java Project .....	154
7.1.3 Writing the application. ....	157
7.2 Developing a managed Java application using the IMS Universal Database Resource Adapter (XA) and DB2 Data Server Drivers (XA). ....	160
7.2.1 Prerequisites .....	160
7.2.2 Installing the products .....	161
7.2.3 Creating the Projects in Rational Application Developer. ....	161
7.2.4 Sample code for a managed environment .....	163
7.2.5 Exporting the application. ....	168
7.2.6 Setting up the IMS Universal DB Resource Adapters in WebSphere Application Server 7.0. ....	168
7.2.7 Setting up DB2 Data Server Drivers in WebSphere Application Server 7.0 ....	171
7.2.8 Installing and starting the application .....	172
7.2.9 Running the application .....	173
7.3 Developing an IMS Java Transaction using the IMS Universal JDBC driver .....	174
<b>Chapter 8. Scenario 3 - Writing DL/I and mixed applications .....</b>	<b>179</b>
8.1 Writing applications with the IMS Universal DL/I Driver .....	180
8.1.1 Accessing IMS data with the IMS Universal DL/I driver .....	180
8.1.2 Retrieving Data Using the IMS Universal DL/I drivers .....	181
8.1.3 Inserting data using the IMS Universal DL/I driver .....	186
8.1.4 Updating data with the IMS Universal DL/I driver .....	187
8.1.5 Deleting data with the IMS Universal DL/I driver. ....	187
8.1.6 Using the Batch Methods with the IMS Universal DL/I driver. ....	195
8.2 Writing application with the IMS Universal DB Resource Adapter and the CCI programming approach .....	199
8.2.1 Writing the application step by step .....	200
8.2.2 Complete Code Example of CCI mixed application .....	203
<b>Chapter 9. Operational considerations .....</b>	<b>207</b>
9.1 Architectural suggestions .....	208
9.1.1 Application middle layer .....	208
9.1.2 Sysplex considerations .....	208
9.1.3 Performance considerations .....	209
9.2 Enhancing existing applications .....	210
9.2.1 ODBA access through ODBM. ....	211
9.2.2 Enabling unsupported Java environments .....	212
9.3 Tracing in problem cases .....	212

9.3.1	IMS Universal driver tracing . . . . .	212
9.3.2	ODBM tracing . . . . .	214
9.3.3	IMS Tracing . . . . .	214
9.4	Using Tools with IMS Open database . . . . .	214
9.4.1	IMS Connect Extensions . . . . .	215
9.4.2	IMS Problem Investigator . . . . .	216
9.4.3	Identifying and resolving problems . . . . .	221
9.5	Additional sample programs . . . . .	225
<b>Appendix A. IBM DB2 Data Server Drivers and Clients . . . . .</b>		<b>227</b>
A.1	IBM Data Server Drivers and Clients . . . . .	228
A.1.1	IBM Data Server Driver for JDBC and SQLJ . . . . .	228
A.1.2	IBM Data Server Driver for ODBC and CLI (CLI driver) . . . . .	230
A.1.3	IBM Data Server Driver Package . . . . .	230
A.1.4	IBM Data Server Runtime Client . . . . .	230
A.1.5	IBM Data Server Client . . . . .	231
A.1.6	Driver and Client comparison . . . . .	231
A.2	Support for JDBC and SQLJ . . . . .	231
A.3	Using the IBM Data Server Driver for JDBC and SQLJ . . . . .	232
<b>Appendix B. Car Dealer IVP Database . . . . .</b>		<b>235</b>
B.1	Car Dealer database overview . . . . .	236
B.1.1	AUTOLPCB overview diagram . . . . .	236
B.1.2	EMPLPCB overview diagram . . . . .	236
B.1.3	Metadata description . . . . .	237
B.2	Car Dealer database source files . . . . .	239
B.2.1	AUTPSB11.psb . . . . .	239
B.2.2	AUTODB.dbd . . . . .	240
B.2.3	EMPDB2.dbd . . . . .	241
B.2.4	SINDEX11.dbd . . . . .	241
B.2.5	SINDEX22.dbd . . . . .	242
B.2.6	AUTOLDB.dbd . . . . .	242
B.2.7	EMPLDB2.dbd . . . . .	243
<b>Appendix C. The environment for our scenarios . . . . .</b>		<b>245</b>
C.1	Used system configuration . . . . .	246
C.2	Used application versions . . . . .	246
C.3	Required APAR numbers . . . . .	247
<b>Appendix D. Additional material . . . . .</b>		<b>249</b>
	Locating the Web material . . . . .	249
	Using the Web material . . . . .	249
	System requirements for downloading the Web material . . . . .	250
	How to use the Web material . . . . .	250
<b>Abbreviations and acronyms . . . . .</b>		<b>253</b>
<b>Related publications . . . . .</b>		<b>257</b>
	IBM Redbooks . . . . .	257
	Other publications . . . . .	257
	Online resources . . . . .	258
	How to get Redbooks . . . . .	258
	Help from IBM . . . . .	259
<b>Index . . . . .</b>		<b>261</b>



# Figures

1-1 The IMSDC and IMSDB components of IMS .....	3
1-2 IMS and SOA: The big picture .....	4
1-3 IMS SOAP Gateway development and runtime environment.....	7
1-4 DLIModel Utility.....	9
1-5 DataPower components .....	10
1-6 Open DB environment .....	12
1-7 IMS and Java - The options .....	13
1-8 The IMS Enterprise Suite Connect API for Java simplifies client interactions with IMS	16
2-1 IMS 11 Open Database Overview .....	19
2-2 Methods of accessing online IMS DB in V9 and V10 .....	20
2-3 Architecture prior to IMS 11 .....	21
2-4 Effect of leveraging the SCI .....	22
2-5 The new CSL address space (IMS Connect) .....	23
2-6 The final architecture .....	24
2-7 Overview of an IMS configuration that includes ODBM .....	26
2-8 Overview of IMS Connect support for IMS DB systems .....	31
2-9 Open database cross LPAR transaction management.....	33
2-10 IMS Universal drivers .....	35
3-1 Sample output from QUERY ODBM TYPE(ALIAS) SHOW(ALL) command .....	54
3-2 Sample output from QUERY ODBM TYPE(CONFIG) SHOW(ALL) command.....	55
3-3 Sample output from QUERY ODBM TYPE(DATASTORE) SHOW(ALL) command. . .	55
3-4 Sample output from QUERY ODBM TYPE(SCIMEMBER) SHOW(ALL) command. . .	56
3-5 Output from QUERY ODBM TYPE(THREAD) SHOW(PSB SCIMEMBER) command.	57
3-6 Sample output from QUERY ODBM TYPE(TRACE) SHOW(ALL) command .....	57
3-7 Sample response for UPDATE ODBM STOP(CONNECTION) DATASTORE(IMSB) .	58
3-8 Sample response for UPDATE ODBM START(CONNECTION) DATASTORE(IMSB)	59
3-9 The IMS Application Menu .....	64
3-10 Selecting the sub-options of IMS Connect and the Open Database Sample in the IVP.	65
3-11 The IV3E302J and IV3E303J jobs show examples adding required PROCLIB members. ....	66
3-12 The IV3T series of the IVP contains jobs that start IMS Open Database components	67
3-13 Specifying the data set name containing our member definitions to be validated . . .	68
3-14 Selecting the IMS Connect configuration member for review within Syntax Checker.	68
3-15 The Syntax Checker requesting input regarding the member type. ....	69
3-16 Syntax Checker prompting for IMS version level .....	69
3-17 Syntax Checker view of our IMS Connect configuration member, HWSCFODB . . .	70
3-18 Help panel for the PORTTMOT parameter of the ODACCESS statement.....	71
4-1 Input and output associated with the IMS Enterprise Suite DLIModel utility.....	77
4-2 IMS family main panel.....	78
4-3 Starting point for downloading the IMS Enterprise Suite DLIModel utility plug-in . . . .	80
4-4 Download page for the IBM Installation Manager and IMS Enterprise Suite DLIModel utility plug-in .....	81
4-5 The install.exe file initiates the IBM Installation Manager installation process .....	82
4-6 The confirmation panel after the IBM Installation Manager has been installed. ....	82
4-7 The main menu displayed when the IBM Installation Manager is launched .....	83
4-8 Adding the DLIModel utility repository to the IBM Installation Manager .....	84
4-9 Selecting the IMS Enterprise Suite DLIModel Utility Plug-in package for installation . .	84

4-10	Option for the DLIModel utility to shell share with other Eclipse products . . . . .	85
4-11	New DLIModel Utility Project - Step 1 . . . . .	88
4-12	New DLIModel Utility Project - Step 2 . . . . .	89
4-13	New DLIModel Utility Project - Step 3 . . . . .	89
4-14	AUTPSB11 Overview diagram . . . . .	90
4-15	Import Copybook Fields . . . . .	91
4-16	Export of AUTPSB11.jar . . . . .	92
4-17	Compiler compliance level . . . . .	93
5-1	Distributed access - Type 4 connectivity . . . . .	100
5-2	Extract from overview diagram from Car Dealer IVP Example . . . . .	116
5-3	Query result from MODEL table . . . . .	116
6-1	Data Studio Installation - Operating System Choice . . . . .	127
6-2	Data Studio - Workspace selection . . . . .	128
6-3	Data Studio - Welcome panel . . . . .	128
6-4	Data Studio - Configuring IMS Universal Drivers Step 1 . . . . .	129
6-5	Data Studio - Configuring IMS Universal Drivers Step 2 . . . . .	130
6-6	Data Studio - Configuring IMS Universal Drivers Step 3 . . . . .	130
6-7	Data Studio - Configuring IMS Universal Drivers Step 4 . . . . .	131
6-8	Data Studio - Creating a new Connection Step 1 . . . . .	131
6-9	Data Studio - Creating a new Connection Step 2 . . . . .	132
6-10	Data Studio - AUTPSB11 expanded view . . . . .	133
6-11	Data Studio - Return all Rows . . . . .	133
6-12	Data Studio - Return all Rows - Results . . . . .	134
6-13	Data Studio - Data Edit . . . . .	134
6-14	Data Studio - Extract Data . . . . .	134
6-15	Data Studio - Add to Overview Diagram . . . . .	135
6-16	Data Studio - Overview Diagram . . . . .	135
6-17	Data Studio - New SQL Script . . . . .	136
6-18	Data Studio - New SQL Script Step 2 . . . . .	136
6-19	Data Studio - SQL Script Step 3 . . . . .	137
6-20	Virtual View Manager in the IBM Cognos 8 architecture . . . . .	139
6-21	IBM Cognos Virtual View Manager architecture . . . . .	140
6-22	Virtual View Manager Studio interface . . . . .	141
6-23	VVM - Login panel . . . . .	142
6-24	VVM - New Data Source - Step 1 . . . . .	142
6-25	VVM - New Data Source - Step 2 . . . . .	143
6-26	New Adapter values . . . . .	144
6-27	VVM - New Data Source Step 3 . . . . .	144
6-28	VVM - New Data Source Step 4 . . . . .	145
6-29	JDBC Connection Properties . . . . .	145
6-30	VVM - Show Contents of a table . . . . .	146
6-31	Install RAR . . . . .	147
6-32	Selection of J2C connection factories . . . . .	148
6-33	Creating a new Feed . . . . .	149
6-34	Database connection panel . . . . .	149
6-35	The SQL query builder panel . . . . .	150
6-36	The final feed panel . . . . .	151
7-1	Switch to Java perspective . . . . .	155
7-2	Configure Build Path . . . . .	155
7-3	Java Build Path Properties . . . . .	156
7-4	New Java Package . . . . .	156
7-5	New Java Class . . . . .	156
7-6	New Java Class Properties . . . . .	157

7-7 New EAR Project .....	162
7-8 New EJB Project .....	162
7-9 New EJB 3.0 Sessionbean .....	163
7-10 WAS - Console .....	169
7-11 WAS Console - Specify Class path .....	170
7-12 WAS Console - New J2C connection factory .....	170
7-13 WAS Console - New JDBC Provider .....	172
7-14 WAS Console - JNDI Mapping .....	173
7-15 Managed Application Web Site Results .....	174
9-1 A two member IMSplex sample environment .....	209
9-2 IMS Connect Extensions event collection .....	216
9-3 Tracking a sequence of Open Database requests .....	217
9-4 Application records filtering .....	218
9-5 Tracing a request initiation .....	219
9-6 Viewing segment search arguments .....	219
9-7 ODBM response tracking .....	220
9-8 Zooming on specific fields .....	220
9-9 Filtering by A047 records .....	221
9-10 Displaying the message area .....	222
9-11 Tracking the complete flow .....	223
9-12 DRDA and DL/I flow .....	224
9-13 Displaying detailed information with F11t .....	224
9-14 Displaying the I/O area .....	225
A-1 IBM Data Server Driver for JDBC and SQLJ connecting directly to DB2 for z/OS ...	229
B-1 AUTOLPCB Overview diagram .....	236
B-2 EMPLPCB Overview diagram .....	237
C-1 The system configuration used for this book .....	246



# Tables

2-1 Options for defining RAS security for applications that use ODBA . . . . .	28
2-2 z/OS runtime environment support for IMS Universal drivers with type-2 connectivity .	38
2-3 Comparison of programming approaches . . . . .	39
4-1 Conversion table - Copybook format to data types . . . . .	94
5-1 z/OS runtime environment support for IMS Universal drivers with type-2 connectivity	100
5-2 IMS Universal drivers settings . . . . .	101
5-3 Comparison of IMS Universal driver approaches . . . . .	103
5-4 Comparison of JCA Models - CCI and JDBC . . . . .	105
5-5 Mapping between IMS terms and relational terms . . . . .	113
5-6 SQL keywords . . . . .	114
5-7 Restricted AQL keywords . . . . .	115
5-8 Aggregate functions examples . . . . .	120
5-9 Aggregate Functions and result types . . . . .	121
5-10 JDBC data types to Java data types mapping . . . . .	122
5-11 Available get methods for data types . . . . .	122
6-1 New Adapter Values . . . . .	143
6-2 JDBC Connection Properties . . . . .	145
8-1 Methods for DL/I retrieve from the PBC interface . . . . .	184
8-2 The get methods . . . . .	185
A-1 IBM Data Server Drivers and Clients comparison . . . . .	231
C-1 Products and versions . . . . .	246
C-2 Products and APARs . . . . .	247



# Examples

3-1 Sample BPE configuration member for SCI and OM address spaces . . . . .	43
3-2 Sample BPE configuration member for ODBM address space. . . . .	44
3-3 Sample BPE configuration member for IMS Connect address space. . . . .	44
3-4 Sample configuration of the SCI initialization member . . . . .	45
3-5 Sample SCI startup procedure JCL . . . . .	46
3-6 Sample configuration of the OM initialization member . . . . .	47
3-7 Sample OM startup procedure JCL . . . . .	48
3-8 Sample ODBM initialization member . . . . .	49
3-9 Sample ODBM configuration member . . . . .	50
3-10 Sample ODBM startup procedure JCL . . . . .	52
3-11 Sample IMS Connect configuration member . . . . .	60
3-12 Sample startup procedure for the IMS Connect address space . . . . .	62
3-13 Defining the HWSAUTH0 user exit within the BPE exit list PROCLIB member . . . . .	72
3-14 Adding a user exit to the BPE configuration member with the EXITMBR statement . . . . .	72
3-15 Sample RACF definitions for authorizing a user to access a protected APSB . . . . .	73
5-1 Extract of the Java metadata class from the Car Dealer IVP Example. . . . .	97
5-2 CCI with SQL calls . . . . .	105
5-3 CCI with DL/I calls. . . . .	106
5-4 JCA/JDBC with SQL calls. . . . .	108
5-5 JDBC DataSource Connection with Application Managed approach . . . . .	110
5-6 JDBC DataSource Connection with JNDI Managed approach. . . . .	110
5-7 Connecting with the JDBC DriverManager Interface . . . . .	111
6-1 Contents of Dealer.csv . . . . .	135
7-1 Code of IMSJDBCStandalone Application . . . . .	157
7-2 XASessionBeanLocal.java . . . . .	163
7-3 XASessionBean.java . . . . .	164
7-4 XAServlet.java . . . . .	166
7-5 XATest.jsp. . . . .	167
7-6 InputMessage.java . . . . .	175
7-7 OutputMessage.java . . . . .	175
7-8 CarDealerTrans . . . . .	175
7-9 CarDealerDBInteraction . . . . .	176
8-1 IMSConnectionSpec properties . . . . .	181
8-2 Creating a PSB instance. . . . .	181
8-3 Obtaining a PCB handle and specifying SSAs using the SSAList interface . . . . .	182
8-4 Qualified SSAList with initial qualification . . . . .	183
8-5 Qualified SSAList with multiple qualifications for one SSA . . . . .	184
8-6 Qualified SSA using a command code . . . . .	184
8-7 Create and Insert method . . . . .	186
8-8 The replace method . . . . .	187
8-9 Deleting all segments in the path . . . . .	188
8-10 Deleting segments with an unqualified ssalist . . . . .	188
8-11 dlitest1 - A Complete DL/I application. . . . .	188
8-12 Output from the application . . . . .	194
8-13 dlitest2 - Batch methods example. . . . .	196
8-14 Import Statements of CCI Application. . . . .	200
8-15 Create MCF . . . . .	200
8-16 Lookup MCF . . . . .	200

8-17 Transaction calls . . . . .	201
8-18 Create Interaction and InteractionSpecs. . . . .	201
8-19 Insert segments with SQL. . . . .	201
8-20 Retrieve information with DL/I. . . . .	201
8-21 Retrieve information with SQL . . . . .	202
8-22 Update information with DL/I. . . . .	202
8-23 Output of MODKEY retrieve . . . . .	202
8-24 Delete data with SQL . . . . .	203
8-25 CCISandaloneDLIandSQL. . . . .	203
8-26 Output of CCISandaloneSQLandDLI Java application . . . . .	205
9-1 Sample DRA for ODBA access through ODBM . . . . .	211
9-2 logging.properties entries for tracing. . . . .	213
9-3 Application enabled tracing. . . . .	213
9-4 DFSERA10 options. . . . .	214
A-1 Determining the driver version . . . . .	232
A-2 Using the getConnection() . . . . .	233
A-3 Connecting to DB2 for z/OS via the DataSource interface. . . . .	233
B-1 DLIModel IMS Java Report . . . . .	237
B-2 AUTPSB11 PSB source . . . . .	239
B-3 AUTODB DBD source . . . . .	240
B-4 EMPDB2 DBD source . . . . .	241
B-5 SINDEXT11 DBD source . . . . .	242
B-6 SINDEXT22 DBD source . . . . .	242
B-7 AUTOLDB DBD source . . . . .	242
B-8 EMPLDB DBD source . . . . .	243



# Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.*

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

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

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

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

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

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


## COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate 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.

## Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at <http://www.ibm.com/legal/copytrade.shtml>

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

AIX®	DRDA®	pureXML®
CICS®	Enterprise Workload Manager™	RACF®
Cognos®	IBM®	Rational®
DataPower®	IMS™	Redbooks®
DB2 Connect™	Informix®	Redpaper™
DB2 Universal Database™	InfoSphere™	Redbooks (logo)  ®
DB2®	iSeries®	System z®
Distributed Relational Database Architecture™	OMEGAMON®	WebSphere®
	Optim™	z/OS®

The following terms are trademarks of other companies:

Cognos, and the Cognos logo are trademarks or registered trademarks of Cognos Incorporated, an IBM Company, in the United States and/or other countries.

Hibernate, Interchange, Red Hat, and the Shadowman logo are trademarks or registered trademarks of Red Hat, Inc. in the U.S. and other countries.

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

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

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

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

# Preface

IMS™ Version 11 continue to provide the leadership in performance, reliability and security expected from the product of choice for critical online operational applications. IMS 11 also offers new functions to help you keep pace with the evolving IT industry.

The introduction of the new IMS Enterprise Suite allows application developers with minimal knowledge of IMS Connect to start developing client applications to communicate with IMS.

With Open Database, IMS 11 also provides direct SQL access to IMS data from programs running on any distributed platform unlocking DL/I data to the world of SQL application programmers.

In this IBM® Redbooks® publicatio, the system programmers will see the steps for installing the new IMS components and the application programmer will follow scenarios of how client applications can take advantage of SQL to access IMS data.

We describe the installation of prerequisites such as IMS Connect and the Structured Call Interface component of Common Service Layer address space and document the set up of the three new IMS drivers:

- ▶ Universal DB resource adapter
- ▶ Universal JDBC driver
- ▶ Universal DLI driver.

Our scenarios use the JDBC driver for type 4 access from Windows® to a remote DL/I database as well as DB2® tables and extend it to use IBM Mashup Center to provide an effective Web interface and integrate with Open Database.

## The team who wrote this book

This book was produced by a team of specialists from around the world working for the International Technical Support Organization at the Silicon Valley Lab, San Jose.



**Paolo Bruni** is an Information Management software Project Leader with the ITSO since 1998. He is based in Silicon Valley Lab, San Jose. Paolo has authored many IBM Redbooks publications about IMS, DB2 for z/OS®, and related tools and has conducted workshops worldwide.



**Angie Greenhaw** joined IBM in 2000 after graduating from Arizona State University with a Bachelors degree in Computer Information Systems. She is currently an IT Specialist in the IMS Advanced Technical Skills group, where she is a primary resource in the areas of IMS security, Dynamic Resource Definition, Common Service Layer and Online Change. Angie enjoys educating customers at conferences such as IOD and SHARE, as well as IMS Regional User Group meetings on these topics among others. Prior to this role Angie worked in IMS development, specializing in the Online Change function, contributing to new IMS functionality and devising solutions as a

Level 3 Service Representative in this same area. She also spent three years as the IMS Development Representative at SHARE. Angie has also contributed to intellectual capital creation, having written a white paper on the topic of Global Online Change implementation, and having co-authored two other IBM Redbooks: *IBM IMS Version 10 Implementation Guide*, SG24-7526 and *IMS Version 11 Technical Overview*, SG24-7807.



**Thilo Liedloff** is a field technical professional for IMS in IBM Germany since 2007. He holds a Bachelors degree in Business Information technology from the Baden-Wuerttemberg Cooperative State University Stuttgart, Germany. He has been working in the area of IMS and IMS Tools for three years. His specializations are in the area of modern IMS Connectivity and IMS Application Development.



**Bob Stone** is a System z® IM technical pre-sales professional in IBM UK. He had 8 years as a COBOL application developer, 18 years as an IMS DBA/Systems Programmer and 10 years as a DB2 DBA working for a large number of commercial companies. He joined IBM in 2000 and spent 8 years as a System z DB2 DBA and Systems Programmer supporting IBM internal systems before taking up his current role where he now specializes in IMS. He has co-authored the book *DB2 for z/OS Tools for Database Administration and Change Management*, SG24-6420, published in July 2003.

## Acknowledgments

The authors thank Rafael Avigad for his contribution in written content.

Rafael Avigad is a product architect and information developer with Fundi Software, in Perth, Western Australia. For the past five years, he has worked on solutions for managing TCP/IP access to IMS and is helping develop current and future IMS tools graphical user interfaces. Before working at Fundi, Rafael developed operational support and billing systems for mobile telephony and ISPs.

Special thanks to Alison Coughtrie and Kevin Hite for making the bases for some example sources available and Kyle Charlet for the support throughout the project.

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

Rich Conway  
 Bob Haimowitz  
 Emma Jacobs  
 International Technical Support Organization

John Barmettler  
 Thomas Bridges  
 John Butterweck  
 Kyle Charlet  
 David Compton  
 Kevin Hite  
 Rose Levin  
 Nisanti Mohanraj

Danny Nguyen  
Richard Tran  
IBM Silicon Valley Lab

Alison Coughtrie  
IBM UK

Denis Gaebler  
IBM Germany

## Now you can become a published author, too!

Here is an opportunity to spotlight your skills, grow your career, and become a published author - all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

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

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

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

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

[ibm.com/redbooks](http://ibm.com/redbooks)

- Send your comments in an e-mail to:

[redbooks@us.ibm.com](mailto:redbooks@us.ibm.com)

- Mail your comments to:

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





# Introduction

This chapter provides an introduction to contents of this book.

We describe how IMS continues to provide solutions to exploit the latest technologies to address customer requirements and provide background information on the IMS Open Database architecture.

This chapter contains the following:

- ▶ IMS is open
- ▶ IMS 10 and SOA
- ▶ The IMS Open Database provides distributed access to IMS database resources. It also helps to drive open standards and open technology into IMS. The open standards that are introduced into this solution include the Java EE Connector Architecture, JDBC, and DRDA®.
- ▶ The IMS Enterprise Suite

## 1.1 IMS is open

The IBM Information Management System (IMS) is the IBM premier transaction and hierarchical database management system, the product of choice for critical online operational applications and data where support for high availability, performance, capacity and integrity are key factors. IMS manages the world's mission-critical data and continues as a major player in the on demand world. IMS customers are still growing in size and in number. As we are moving into the era of on demand computing, IMS is helping to lead the way by continuing to provide solutions to exploit the latest technologies to address customers' requirements for performance and availability, but also focusing on enterprise modernization through integration and open access with an on demand service-oriented architecture.

New application development tools and the IBM service-oriented architecture capabilities for IMS can help your business improve the speed and agility of its development efforts. Both IMS and the IMS SOA Integration Suite support your on demand systems and your distributed IMS application environment.

The introduction of the new IMS Enterprise Suite Connect API for Java™ allows application developers with minimal knowledge of IMS Connect to start developing client applications to communicate with IMS. The Connect API for Java is a simple, easy-to-use, lightweight programming solution for communicating with IMS transactions through IMS Connect.

With Open Database, IMS 11 provides direct SQL access to IMS data from programs running on any distributed platform. Open Database reduces the complexity and processing associated with IMS data access unlocking DL/I data to the wide population of SQL application programmers.

## 1.2 IMS 10 and SOA

Service-oriented architecture (SOA) is an architecture style that is centered around components, or services, with standardized interfaces. It is a methodology of designing and running the software portion of an information technology infrastructure so that it supports the various individual and interrelated functions that are needed to operate a particular enterprise. SOA helps bridge the business/IT gap and helps systems remain scalable and flexible while your business is growing and changing. SOA is focused on business processes, and although many legitimate approaches exist for software architecture, SOA is intended explicitly for business applications:

- ▶ Reusing
- ▶ Packaging
- ▶ Liberating business from the constraints of technology
- ▶ Services

For a more exhaustive discussion on IMS and SOA refer to the IMS manuals and *Powering SOA Solutions with IMS*, SG24-7662.

### 1.2.1 The value of including existing IMS assets into SOA

There is great value in utilizing existing IMS assets in modernization projects:

- ▶ Existing IMS assets support core business processes and provide crucial information.
- ▶ Existing IMS assets contain billions of lines of valuable business rules.



- ▶ Using proven, time-tested IMS applications can significantly lower risk, cost, and time to market.
- ▶ The quality of the IMS system is recognized by all IMS users as fast, reliable, and mature.
- ▶ An IMS server is not a single point-of-failure. IMS has built-in recovery features, with take-over mechanisms, and supports a Sysplex environment.

The traditional IMS application development and database creation do not provide a fast market driven responsiveness model with the multi-step procedure and the involvement of several tools and techniques.

When considering IMS assets, we must distinguish two areas or functions of IMS: the Transaction Manager (IMS TM) and the Database Manager (IMSDB).

- ▶ IMSTM or data communication controller (DCCTL) is the facility to link applications that are running as transactions to networks.
- ▶ IMSDB or Data Base Controller (DBCTL) interfaces with non-IMS communication controllers and traditionally supplies database services through the facilities of Database Resource Adapter (DRA) code.

Figure 1-1 illustrates these two environments.

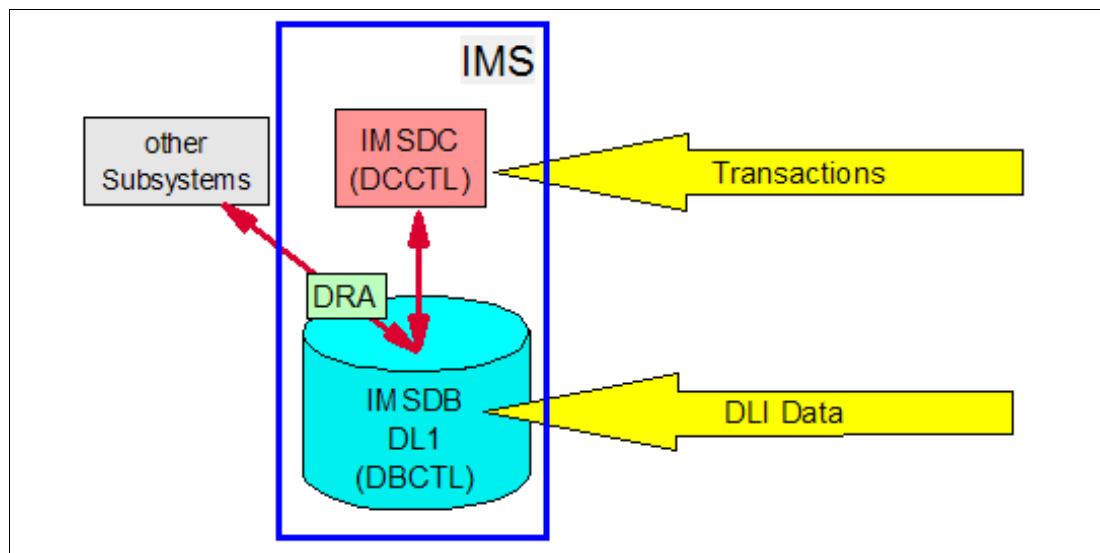


Figure 1-1 The IMSDC and IMSDB components of IMS

When considering IMS integration in SOA, we include both domains, TM and DB.

IMS provides the ability to leverage existing IMS transactions by making them available as callable Web Services. Figure 1-2 illustrates this function. You can view the solutions as allowing network access to and from the IMS host environment and also opening IMS databases for access by non-IMS service requestors.

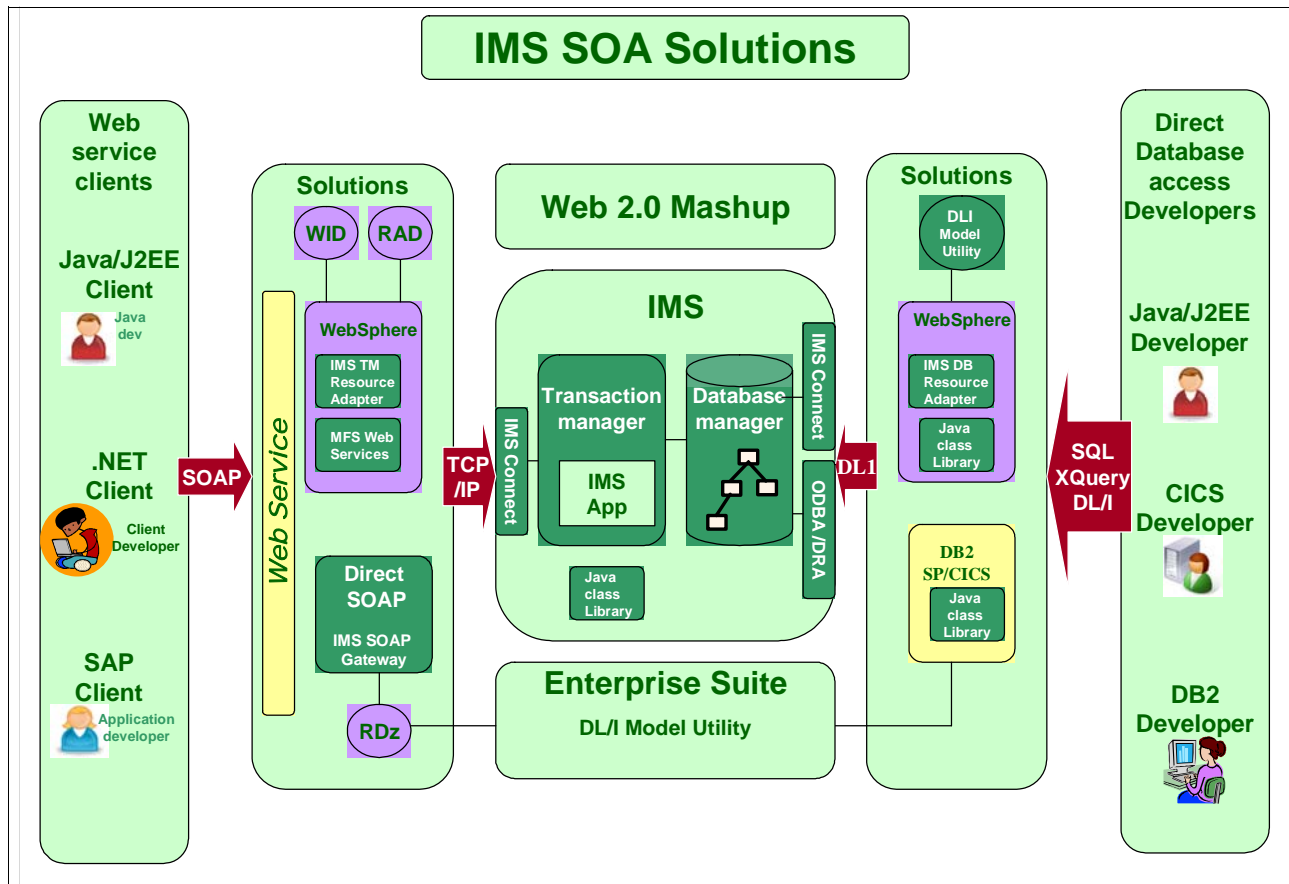


Figure 1-2 IMS and SOA: The big picture

The components in Figure 1-2 are:

- ▶ IMS Connect is the TCP/IP gateway to IMS and runs within a separate z/OS address space to the IMS control region. With IMS 11, IMS Connect is also an ODBM client and it allows distributed applications to access any database in the entire IMSplex.
- ▶ The IMS TM Resource Adapter (TMRA) is a Java EE Connector Architecture (JCA) resource adapter that offers support to access existing IMS transactions from callable Web Services, Enterprise JavaBeans (EJBs), or even from HTML pages and servlets. IMS TMRA is a WebSphere® Application Server-based solution. The tooling used in this solution is either IBM Rational® Application Developer (RAD) or WebSphere Integration Developer (WID), and they receive definitions of the input/output messages in either COBOL, C, or PL/I and generate all of the necessary artifacts and code.
- ▶ Another WebSphere Application Server-based solution is targeted at applications that are MFS based. MFS is much more complex than, for example a simple COBOL copybook, and as such IMS offers a specific solution for MFS. This solution offers tooling that consumes the MFS definition of the input/output messages and again generates the necessary artifacts. The MFS solution is an IMS TM Resource Adapter client so it leverages all of its functionality.
- ▶ Another solution offered is the IMS SOAP gateway, which offers direct SOAP access to existing IMS transactions. It is a non-WebSphere based solution and does not require a Java EE (Java Platform, Enterprise Edition) container. There is tooling that is specific to this solution in RAD for z that consumes input/output message definitions, and once again generates all of the necessary artifacts for exposing an IMS transaction as a callable SOAP service.

- ▶ Although the TM-based solutions put focus on leveraging existing IMS transactions, the DB based solutions are focused on new IMS application development. These solutions offer direct IMS database access from a variety of environments. IMS allows WebSphere, CICS®, DB2, and even application developers the ability to access IMS database assets using industry standard programming models and APIs.
- ▶ Two new IMS JMP and JBP-dependent regions were introduced. A JMP region is analogous to an MPP region and a JBP, to a non-message driven BMP region. The fundamental difference is that with the new regions, IMS now can fully house and maintain a JVM (Java Virtual Machine), which enables IMS to now effectively process Java workloads. In addition to this, IMS offers the Java class libraries, which contain a complete API for Java developers to use. The Java libraries offer a JDBC driver for IMS, which can process both SQL and XQuery expressions.

These same Java libraries can be utilized from several different runtime environments:

- WebSphere Application Server
- DB2 stored procedures on z/OS
- CICS using their JCICS API
- ▶ The IMS Open Database Access (ODBA) and DRA modules offer Java libraries the ability to access IMS databases from a non-IMS environment.
- ▶ For the WebSphere Application Server environment, IMS offers another JCA resource adapter, the IMS DB Resource Adapter. The difference with this adapter, as opposed to the IMS TM Resource Adapter, is that all of the business logic is in the EJBs themselves. With the IMS DB Resource Adapter, there is no IMS-dependent region involved at all.
- ▶ With IMS 11, the IMS Universal drivers, part of the Open Database solution, are software components that provide Java applications with access to IMS databases from z/OS and from distributed environments through TCP/IP.
- ▶ For all of these solutions, IMS also offers tooling support through the DLIModel utility, which is an Eclipse-based GUI tool now part of the Enterprise Suite offering. It offers visualization of IMS databases. It consumes PSB, DBD, and even COBOL copybook source to visualize all of the PCBs in a particular PSB. Information, such as hierarchies, segments, fields, and field types, are captured. With respect to new application development, the utility also generates database metadata definitions that are consumed at runtime by the Java libraries, for example, this enables the libraries to convert a SQL query into a native IMS DLI call.

## 1.2.2 IMS Connect and IMS Connect Extensions

With IMS Version 9, IMS Connect was delivered as an integral component of IMS. It performs the vital function of connecting from IMS to the TCP/IP world.

IMS Connect Extensions is a key tool for managing access to IMS through IMS Connect:

- ▶ Key benefits:
  - Provides event collection and instrumentation for IMS Connect
  - Streamlines operational management of IMS Connect and its clients
  - Assists in the development of TCP/IP clients and the transition to an SOA
- ▶ Principal users:
  - IMS tuning specialists, application developers, operators, and administrators

One example of using IMS Connect Extensions is in its assistance to monitor your IMS TCP/IP network flow. OMEGAMON® for IMS on z/OS, as a Real-time monitoring tool for IMS

Connect, uses the Connect Extensions Publisher API where it obtains IMS Connect event records through the API.

### 1.2.3 The IMS SOA Integration Suite

The IMS SOA Integration Suite leverages the utilization of your existing assets and running systems to integrate IMS capabilities in a SOA environment by providing these capabilities:

- ▶ Provides access to IMS transactions and data from any Web connection.
- ▶ Modernizes your IMS applications and enables them to operate with other clients, such as Microsoft®.NET or SAP® clients in a service-oriented architecture.
- ▶ Integrates business logic that is embedded in your existing IMS applications with other IT systems, both within your enterprise and in the supply chain.
- ▶ Improves development time by using Java, instead of PL/I, COBOL, or Assembler.
- ▶ Accesses your IMS data directly for use by your applications from environments, such as DB2, CICS, and WebSphere Application Server.
- ▶ Stores and retrieves your XML content directly in IMS without any intermediate steps, and exchanges data with other systems by using established schemas.

The following tools and functions support access to IMS transactions:

- ▶ IMS SOAP Gateway
- ▶ IMS TM Resource Adapter
- ▶ IMS MFS Web Solutions
- ▶ IMS Web 2.0 Solution
- ▶ DLIModel utility and IMS XML DB

The IMS SOA Integration Suite enables you to access IMS transactions and data. In the next section, we provide an overview of these technologies and their business value.

#### IMS SOAP Gateway

IMS SOAP Gateway is an XML-based connectivity solution that enables existing or new IMS applications to communicate outside of the IMS environment using SOAP message protocol to provide and request services independently of platform, environment, application language, or programming model.

Figure 1-3 illustrates the IMS SOAP Gateway deployment and runtime environment. IMS SOAP Gateway uses RDz to generate both the correlator and WSDL files that are used when deploying Web Services within IMS SOAP Gateway. An IMS SOAP Gateway Deployment utility is included for you to set up properties to deploy and maintain IMS Web services. Also, an IMS SOAP Gateway Administrative Console is available to list the deployed Web services when the server is started.

IMS SOAP Gateway interfaces with IMS Connect using TCP/IP protocols but uses SOAP or HTTP or HTTPS when communicating with SOAP clients.

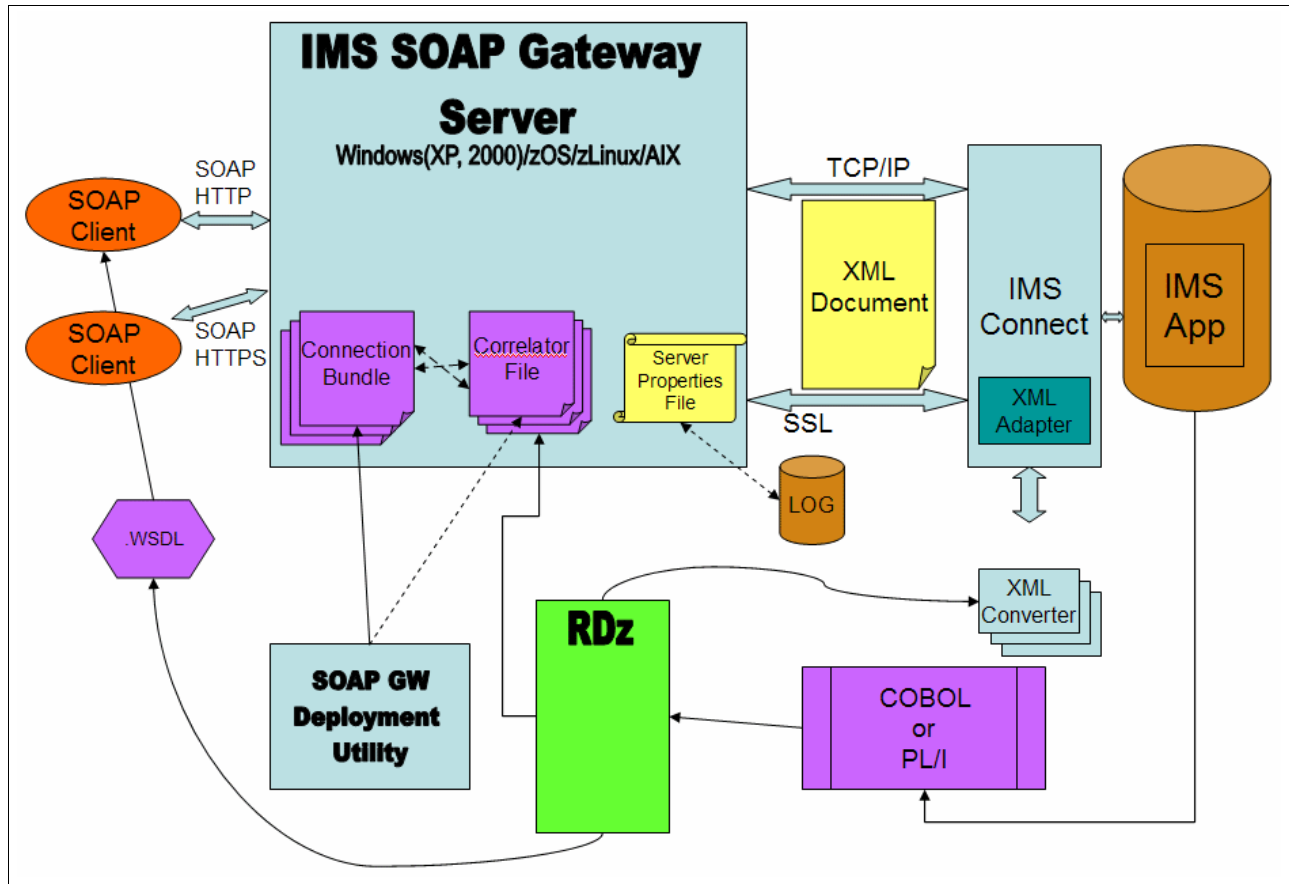


Figure 1-3 IMS SOAP Gateway development and runtime environment

### **The value of using SOAP Gateway**

IMS SOAP Gateway is a light-weight Web Services solution that enables IMS applications to interoperate in a SOA environment without needing a full-blown application server (for example, Java EE Server). One typical usage scenario of providing Web Services with the IMS SOAP Gateway is to enable Microsoft.NET client applications or intermediary servers that submit SOAP requests into IMS to drive business logic transactions.

Through SOAP, IMS SOAP Gateway provides and requests services that are independent of platform, environment, application language, or programming model:

- ▶ It enables IMS application assets as Web Services.
- ▶ It allows non-WebSphere customers to reuse existing and to create new IMS-based business logic.
- ▶ It operates with any types of client application using SOAP/HTTP protocols.

Generated IMS service definitions (that is Web Services Description Language files) can be published or exposed to an UDDI directory for businesses to publish their offerings and for users to discover their needs. You can retrieve IMS WSDL files out of the UDDI directory and fit them into a tool (such as Microsoft.Net or Apache Axis server tools) to generate SOAP messages to be sent to the host to run existing IMS applications.

### **IMS TM Resource Adapter**

The IMS TM Resource Adapter, previously known as IMS Connector for Java, is part of the IMS SOA Integration Suite of middleware functions and tools. Using the IMS TM Resource

Adapter you can quickly and easily create Java applications that access new and existing IMS transactions over the Internet. Using TMRA within a WebSphere or Rational-family development environment, you can:

- ▶ Develop components of business processes in support of SOA.
- ▶ Create Java EE applications from Java beans.

The development version of the IMS TM Resource Adapter is included in the following integrated development environments:

- ▶ Rational Application Developer for WebSphere Software
- ▶ WebSphere Integration Developer
- ▶ WebSphere Transformation Extender
- ▶ Rational Developer for System z (formerly known as WebSphere Developer for System z)
- ▶ Rational Software Architect

You can download the runtime component of the IMS TM Resource Adapter from the Web site:

<http://www.ibm.com/software/data/ims/ims/components/tm-resource-adapter.html#downloads>

### ***The value of using IMS TMRA***

The IMS TM Resource Adapter implements the Java EE Connector Architecture (J2C), which connects Enterprise Information Systems (EISs), such as IMS to the Java EE platform. The Java EE Connector Architecture provides your applications with the qualities of service that a Java EE application server can provide, such as connection, transaction, and security management, which allows for:

- ▶ You can use the IMS TM Resource Adapter with a Java EE server, such as IBM WebSphere Application Server when a Java application accesses an IMS transaction that is running on a host IMS system. The IMS TM Resource Adapter also enables an IMS application to act as a client to invoke applications in a Java EE server.
- ▶ Although the IMS TM Resource Adapter is intended for use primarily by Java applications or Web Services that submit transactions to IMS, the IMS TM Resource Adapter can also be used by services that submit IMS commands to IMS.

MRA provides tooling support for development of Java EE applications, Web Services, and business processes that access IMS transactions in various Rational and WebSphere-integrated development environments.

MRA also provides programming for deployment to the WebSphere Application Server and WebSphere Process Server (WPS) runtime environment on many platforms, which includes z/OS and Linux® on System z.

IMS TM Resource Adapter Version 9 has PID number 5655-J38, and IMS TM Resource Adapter Version 10 uses PID number program number 5635-A01.

### **The DLIModel utility**

Without the DLIModel utility, the IMS-Java user must manually create Java classes that describe the metadata (for example, structure, segment layouts, and so on) of the IMS databases that are to be processed. The DLIModel utility creates these classes from PSB and DBD source plus optionally, high-level language source that more fully describes segment field layouts. The DLIModel utility supplies a Web download version that runs as a plug-in to Eclipse, WebSphere Developer for System z, RAD for WebSphere, and RDz.

### ***The value of using the DLIModel utility***

The business goal of the DLIModel utility is to ease IMS application development. Using the DLIModel utility you can transform your IMS database information (program specification blocks, database descriptions, and COBOL copybooks) into application-independent metadata. In addition to creating metadata, using the IMS DLIModel utility you can:

- ▶ Generate XML schemas of IMS databases, which are used to retrieve XML data from or store XML data in IMS databases.
- ▶ Incorporate additional field information from COBOL copybooks.
- ▶ Incorporate additional PCB, segment, and field information or override existing information.
- ▶ Generate a DLIModel report, which is designed to assist Java application programmers in developing applications based on existing IMS database structures.
- ▶ Generate an optional DLIModel trace log.

Figure 1-4 illustrates the input and output flows from the DLIModel utility.

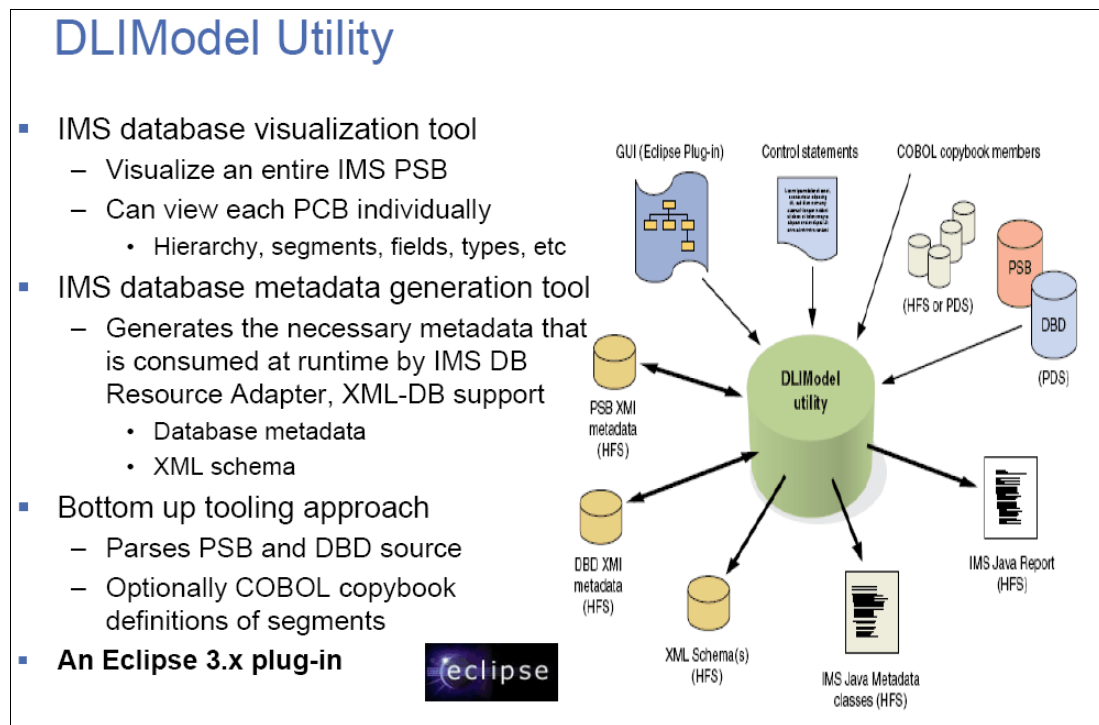


Figure 1-4 DLIModel Utility

## **1.2.4 DataPower and IMS**

DataPower® is a powerful solution for XML acceleration, XML security, monitoring, and managing SOA environments.

Included in the DataPower firmware V3.6.1 is a feature called "IMS Protocol Support" that adds support to allow Multi-Protocol Gateway services to accept IMS connections from clients and connect to IMS-based applications. This functionality provides:

- ▶ An IMS Connect proxy to IMS Connect clients. The use case is for existing IMS Connect clients who want to make in-flight modifications to headers and payloads without changing the client or IMS.



- Web Service facade to IMS Connect transactions. The use case is to make use of the strong Web Service features in DataPower to quickly enable Web Service support for IMS Connect.

Figure 1-5 presents the DataPower hardware components and their primary roles.

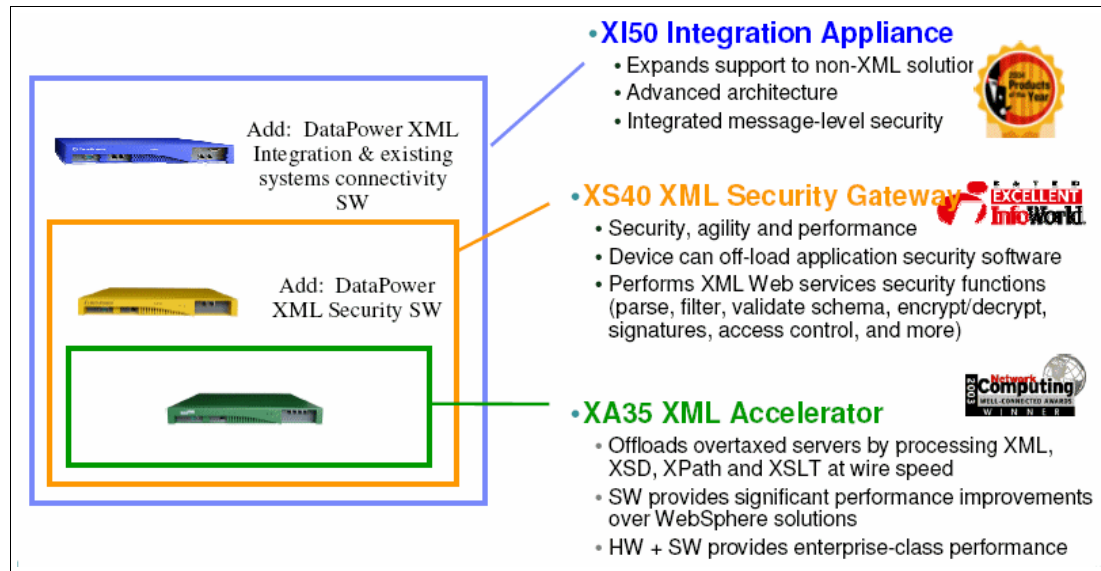


Figure 1-5 DataPower components

## 1.3 IMS 11 and Open Database

The ease with which you can install IMS 11 ensures that you can continue to rely on IMS and its hallmarks of reliability, availability, matchless scalability, and performance. You can take advantage of:

- Dynamic commands to query and change IMS Transaction Manager and IMS Database Manager resources, to create points of consistency, and to reduce the time databases are offline.
- User exit services enhancements to define multiple instances and to dynamically refresh modules.
- Enhanced syntax checking.
- System performance enhancements and improved availability through IMS Fast Path Buffer Manager, Application Control Block library, and Local System Queue Area storage reduction, which utilize 64-bit storage, freeing up valuable ECSA.

When running core applications that are at the heart of business processing, most large corporations worldwide continue to depend on IMS. Today's enterprise IT needs are more closely tied to the business than ever before. Businesses require efficiency to meet the cost challenges and responsiveness demanded by the global economy. IMS 11 provides an easier-than-ever roadmap that supports your business growth for years to come.

This capability reduces the complexity and processing associated with IMS data access. IMS Connect now provides simplified TCP/IP access to *both* IMS transactions and data.



This support for easier integration and open access can also help with industry regulations compliance and internal controls, and can enable you to more rapidly develop and deploy new applications and services.

The IMS Open Database provides distributed access to IMS database resources. It also helps to drive open standards and open technology into IMS. The open standards that are introduced into this solution include the Java EE Connector Architecture, JDBC, and DRDA®.

Historically, the IMS database was a closed architecture, and by opening it up, IMS is positioned for the future as it pertains to industry standard access APIs and the emerging SOA market.

The business challenges addressed by Open Database are:

- ▶ Data can be difficult to access outside of the IMS environment. Clients might want to participate in the emerging SOA market.
- ▶ Traditional IMS support skills are becoming increasingly rare and difficult to acquire and train.
- ▶ It requires too long a time to deploy new applications and enhance existing applications.

### **The value of using IMS Open Database**

The ability to access IMS data from outside of the IMS environment simplifies the process of developing new applications that leverage existing investment in IMS data. Also, it reduces costs by providing distributed access to IMS database resources through industry-standard interfaces.

The distributed access function offers two distinct types of IMS database resource distribution:

- ▶ The distribution of IMS database resources across LPARs in an IMSplex. IMS data can be accessed through TCP/IP from a Java EE application server that resides on a z/OS platform that is on a different logical partition (LPAR) from the IMS subsystem.
- ▶ Pure distribution, which means that IMS database resources are now directly accessible from non-mainframe platforms, which includes full distributed transaction processing and two-phase commit semantics.

All of this is accomplished by three main components:

- ▶ Client-side libraries that implement the industry-standards interfaces and protocols.
- ▶ IMS Connect, which processes the distributed requests.
- ▶ The use of the Open Database Manager (ODBM) address space.

Figure 1-6 introduces the Open Database environment available with IMS Version 11.

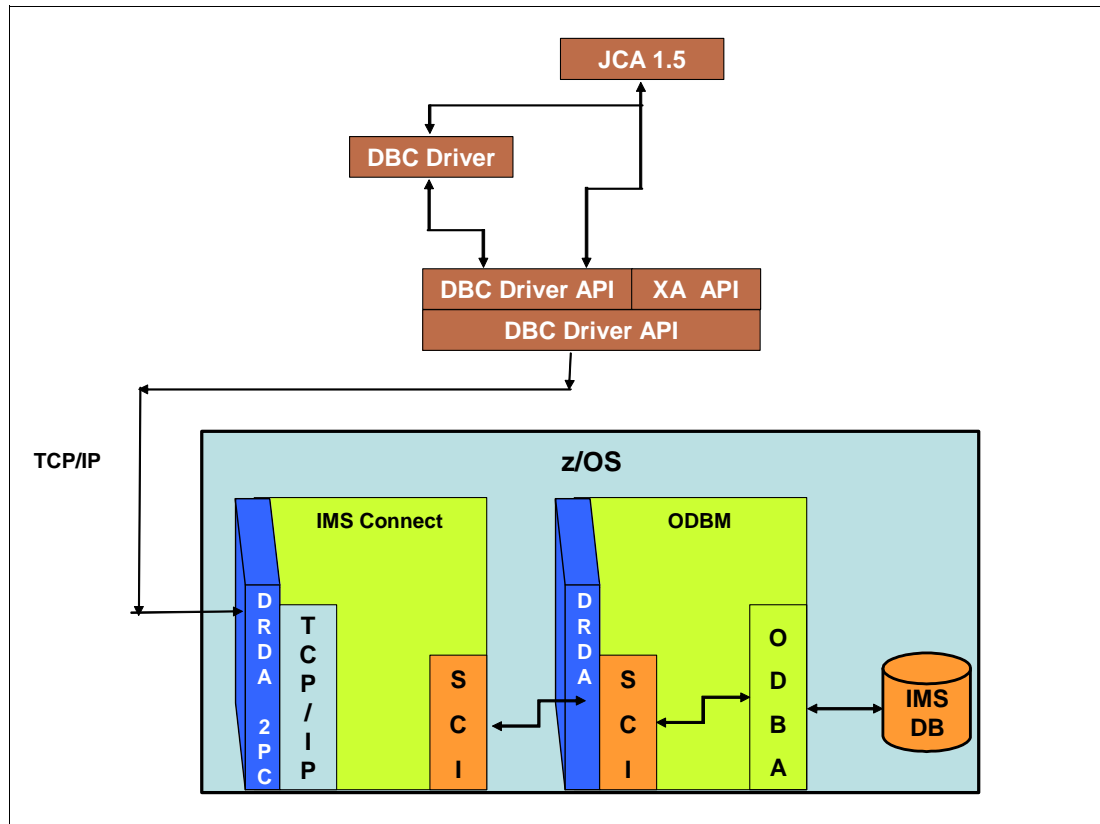


Figure 1-6 Open DB environment

For details about Open Database environment, see Chapter 2, “Open Database architecture” on page 17.

Data is one of the largest and most valuable assets of any business. An organization’s ability to share and deliver trusted information is not only essential to the success of that business, but provides a competitive advantage. However, IMS data hasn’t always been easily accessible and shareable in the way that supports the growth and agility of businesses. To meet these needs, IMS needed to overcome two obstacles: connectivity to the data and the lack of industry standards as part of the data access model for IMS. The IMS Version 11 Open Database solution addresses the challenges of modernizing and standardizing both IMS database access and application development by providing an integrated distributed data access solution. Now you can more easily integrate IMS assets with other products and platforms across your enterprise and the Internet.

### The IMS Universal drivers

The IMS Universal drivers, part of the Open Database Solution, are software components that provide Java applications with connectivity and access to IMS databases from z/OS and from distributed environments through TCP/IP. See Figure .

IMS provides three Universal drivers that support multiple standards and runtime environments:

- IMS Universal DB Resource Adapter - A JCA-compliant resource adapter that provides all the services that the JEE platform provides, including connection, transaction, and security management

- **IMS Universal JDBC Driver** - A Java Database Connectivity (JDBC) driver that supports access to IMS data by using SQL calls

**IMS Universal DL/I Driver** - An IMS-specific Java API for DL/I that can access IMS data by using Java methods that are based on IMS DL/I semantics.

The term Universal is used because the drivers share the same common framework and can be deployed in multiple platforms and runtime environments. Because the drivers have environment-detection protocols built into them, application developers do not need to be concerned with where the application will be deployed—the drivers handle all runtime environment and platform-specific needs.

The IMS Open Database solution focuses on Java programming skills. The ability to tap into this skill base opens significant opportunities for new IMS application development. The standard data access interface in Java today is JDBC. JDBC offers support for SQL, which is used almost exclusively across all major database management systems. The IMS Open Database solution provides an implementation of JDBC, coupled with an engine for SQL processing. If you want to develop an IMS application in Java you have now several options as shown in Figure 1-7.

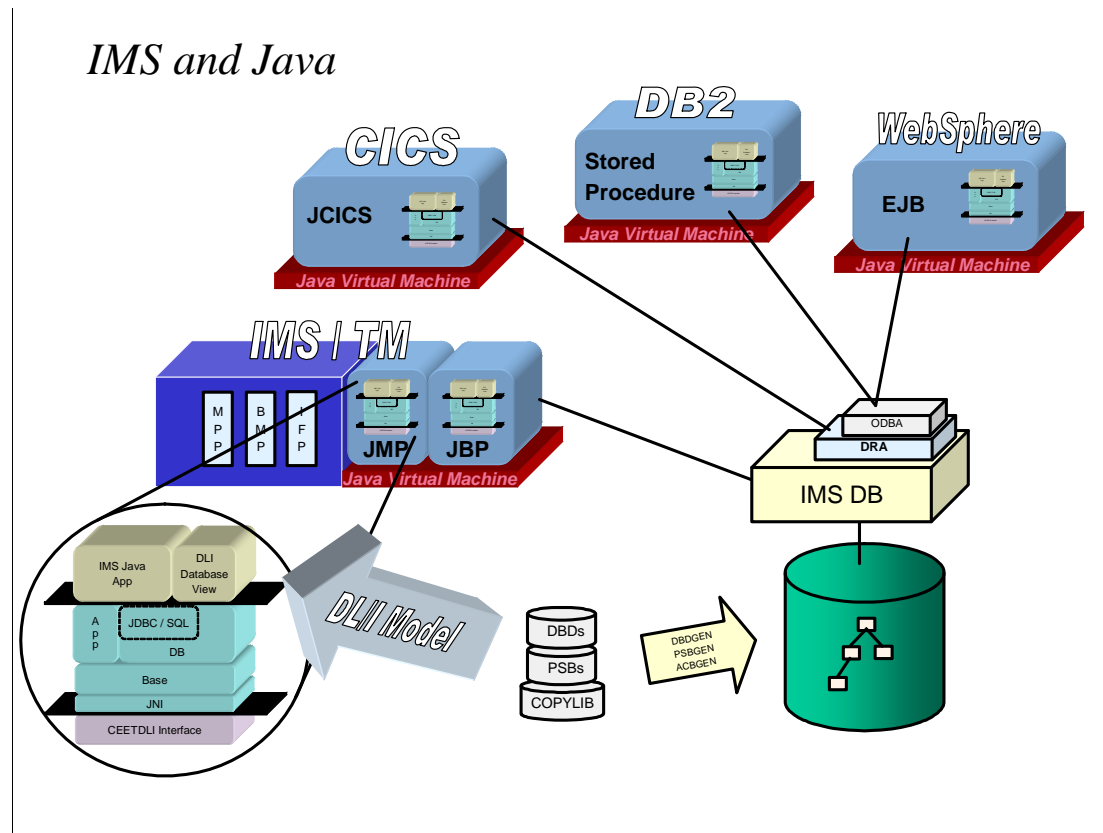


Figure 1-7 IMS and Java - The options

The main choice for many application architects is enterprise application development in tightly managed application servers. The Java EE and Java Connection Architecture standards are built around integration into these server runtime environments. WebSphere Application Server, the IBM Java EE server, adheres to these standards, as does the IMS Open Database solution. Application developers can now deploy their applications into the WebSphere Application Server runtime environment and take full advantage of its management, connection pooling, and security capabilities.

IMS Connect has been the gateway for TCP/IP access to IMS transactional resources. The IMS Open Database solution enhances IMS Connect so that it now provides access to IMS database resources via a new CSL address space: Open Database Manager (ODBM).

To provide programmatic access to IMS data, the Universal drivers can be deployed in the following runtime environments:

- ▶ IMS Java dependent regions
- ▶ WebSphere Application Server for distributed platforms
- ▶ WebSphere Application Server for z/OS
- ▶ CICS
- ▶ DB2 and DB2 for z/OS stored procedures
- ▶ Standalone Java SE

The drivers can run on both distributed and z/OS platforms. TCP/IP is used to access IMS data from distributed platforms, which include Windows, Linux, AIX®, Sun Solaris, UNIX®, and Linux for System z. Pre-existing assembler interfaces are used to access IMS data from z/OS runtime environments.

The Open Database solution allows authentication of all clients at various layers. For example, in a WebSphere Application Server (Java EE) runtime environment, you can use encrypted credentials to manage the security credentials, instead of having the application manage them. RACF® on z/OS (or an equivalent security product) is used to both authenticate the user and ensure that the user has the authorization to the PSB. The Secure Sockets Layer (SSL) protocol is supported for the type 4 Universal drivers, to ensure that your communication layer is encrypted. Rest assured that the Open Database solution offers a robust security model to fit your needs.

## 1.4 The IMS Enterprise Suite

Developing an IMS Connect client application to access IMS transactions can be challenging for application developers. To implement complex business scenarios, an IMS Connect application developer needs specific programming skills to code an IMS Connect client, including in-depth knowledge of IMS Connect protocols, TCP/IP socket programming, message and header formats, and how to set header fields.

With the introduction of the new IMS Enterprise Suite Connect API for Java, application developers need only minimal knowledge of IMS Connect to start developing client applications to communicate with IMS. The Connect API for Java is a simple, easy-to-use, lightweight programming solution for communicating with IMS transactions through IMS Connect. Developers can use a high-level programming language to write IMS Connect client applications. In addition, developers have the flexibility to run the applications standalone, without the overhead of an enterprise application server environment.

The components of the IBM IMS Enterprise Suite V1.1 fully integrate with IBM WebSphere, IBM Rational, and industry tooling. They utilize a common programming model for a service-oriented architecture (SOA) based on standards such as XML, SOAP, Java, JDBC, and other emerging standards. This release of IMS Enterprise Suite V1.1 includes:

- ▶ Connect API for Java
- ▶ Java Message Server (JMS) API
- ▶ DLI Model Utility plug-in
- ▶ SOAP Gateway
- ▶ The IBM Installation Manager and SMP/E support

The Connect APIs enable application to more easily develop client applications that communicate with IMS. Distributed platforms can now more easily connect to IMS; the design, development, and test of IMS access for client TCP/IP applications has never been simpler. The Connect APIs provide a simple, standard way to describe TCP/IP socket connections, interaction protocols, message headers, and data. Developers can use preconfigured values to create connections and interactions rather than manually setting these values.

The industry-standard JMS API allows application developers to define a common set of messaging concepts and programming strategies that are supported by all JMS technology-compliant messaging systems, increasing their overall productivity and addressing skills issues. The JMS API can be used for synchronous callout from an IMS Java application.

### ***DLI Model utility plug-in***

The DLI Model utility plug-in, now delivered as part of IMS Enterprise Suite V1.1, helps application developers transform IMS database information into metadata. Application Developers can now integrate their existing PL/I and COBOL data structures into IMS metadata. JDBC driver performance is enhanced with the addition of PROCOPT to the DLIDatabaseView metadata class. Application developers can now see Virtual Foreign Key fields and a relational view of data, enabling use of the Universal JDBC driver and Universal DB resource adapter. They can also automatically select DBDs referred to by a PSB, and merge existing metadata with modified PSB and DBD sources. Because the DLI Model utility plug-in is Eclipse-based, there is seamless shell-sharing with other Eclipse-based products from IBM with IBM Installation Manager.

IMS Enterprise Suite Connect API for Java is a lightweight programming solution in Java requiring minimal setup, easy to configure, with no additional tooling required. Developers can leverage Java support for code portability, exception handling, and logging. To accelerate development, developers can take advantage of an integrated development environment (IDE) for Java development such as IBM Rational Application Developer or IBM Rational Developer for System z.

Figure 1-8 shows how the Connect API for Java hides the complexity of communication protocols for interactions with IMS Connect behind a simple programming interface.

The client application sets connection and interaction properties by loading from a profile, or configures the property values programmatically. To perform an interaction, the client application simply issues an execute function call. The interaction types supported by the Connect API for Java include SENDRECV, RESUMETPIPE, and several SENDONLY interaction types.

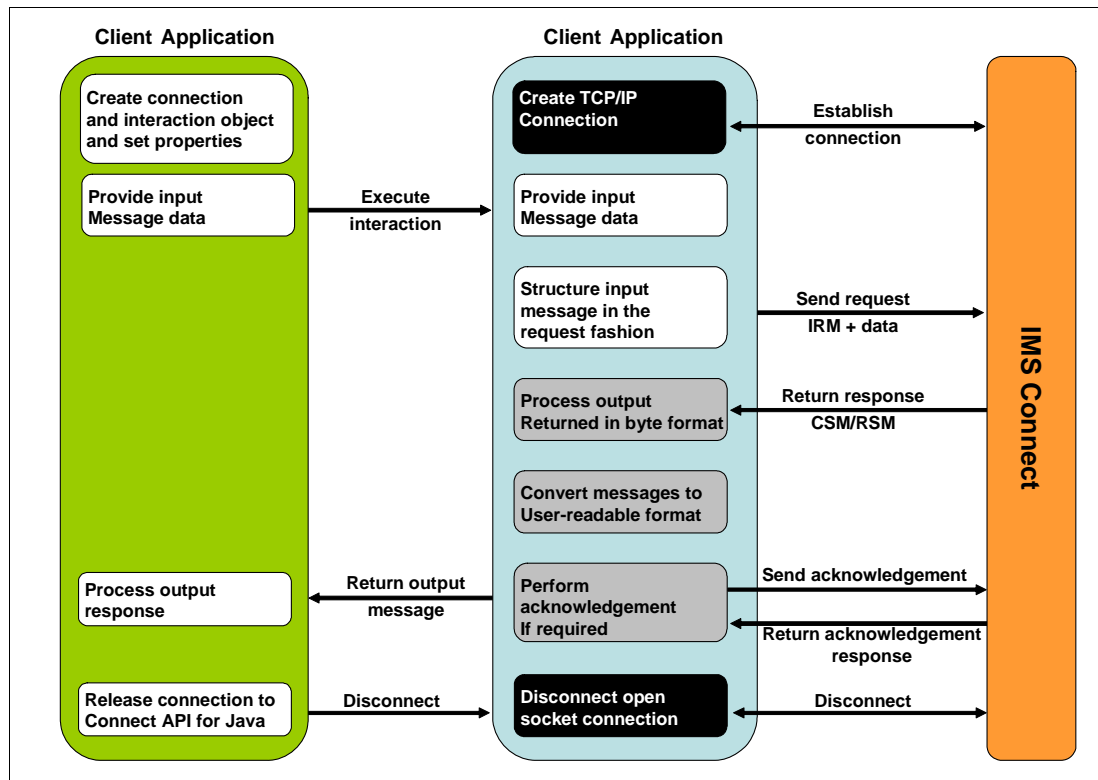


Figure 1-8 The IMS Enterprise Suite Connect API for Java simplifies client interactions with IMS

The Connect API for Java handles the work of establishing a connection, creating an IMS Connect input message, sending the input message to IMS Connect, retrieving any output message returned by IMS Connect, and storing the message in a data structure for easy client application retrieval. .

The Connect API for Java also supports profiles. Profiles are plain text files that contain property name-value pairs that can be predefined for specific connection and interaction scenarios and reused to load connection and interaction property values in different application clients. Developers can also configure the connection and interaction properties dynamically in the client by using property-setter methods.

# 2

## Open Database architecture

In this chapter we describe the architecture of an IMS Open Database environment.

The chapter contains the following:

- ▶ Why IMS Open Database
- ▶ Accessing IMS DB in Versions 9 and 10
- ▶ Evolution in IMS 11
- ▶ IMS 11 Architecture
- ▶ Open Database functions
- ▶ IMS 11 Universal Drivers

## 2.1 Why IMS Open Database

IMS Open Database is a new function in IMS 11 taking on the challenge of modernizing IMS Database access and application development. It provides an environment that manages access to online IMS databases from anywhere in the enterprise. It supports open-standards for connectivity to online IMS databases:

- ▶ Within a z/OS LPAR
- ▶ Across z/OS LPARs
- ▶ From distributed platforms

It addresses two significant bottlenecks for business growth:

- ▶ Connectivity – IMS DB has been historically grounded to the mainframe...certainly there are ways to get to it but none straightforward and simple.
- ▶ Application development – even when connectivity is not an issue – the skills are not readily available to develop new application workload. DL/I is not industry standard and skills are not plentiful.

IMS 11 rolls out a complete suite of Universal drivers in support of IMS database connectivity and programmatic access. The intent is to access IMS in a uniform way using the most relevant industry standards from any platform and from within the most strategic runtimes. A standards-based approach opens a lot of growth and expansion opportunity. The fundamental communication protocol for communicating with IMS Connect is the industry standard Distributed Relational Database Architecture™ (DRDA) protocol. Single Universal driver in support of both type-4 and type-2 connectivity in all supported runtimes – there is no need to learn another driver's semantics to toggle between environments and desired connectivity – it is all built into the framework. Distribution of resources within an IMSplex is included. The idea is to extend the reach of IMS by extending the data. IMS DB metadata is exposed by use of the standard JDBC API and therefore can be consumed and visualized by JDBC tooling. By allowing inspection of metadata, the next step is query. Query syntax uses standard query language syntax.

IMS Open Database – IMS Open Database offers direct distributed access to IMS database resources. The distributed nature is two-fold. At the IMSplex level, it allows cross-LPAR access to any IMS database in the IMSplex. At the pure distributed level, it allows non-mainframe (e.g., Windows OS) access directly to IMS database resources through industry standard interfaces. This enhancement extends IMS Connect as the gateway to IMS DB. It adds a new Common Service Layer address space which manages connections to the IMS ODBA interface. This enhancement improves application access to IMS.

IMS has seen an increased number of requests for distributed access to all database types. IMS Connect is currently the gateway to IMS TM. It also becomes the gateway to IMS DB.

Distribution of database assets comes in two flavours

- ▶ Distribution within an IMSplex. Applications on one LPAR can access an IMS database on another LPAR
- ▶ Distribution to non-System z platforms. Applications on a non-System z platform can have direct IMS DB access without needing an IMS transaction to proxy the data.

The Universal drivers (JCA, JDBC, DLI) allow both distributed as well as local (CICS, IMS, WAS z, DB2 z) access to IMS databases.

Figure 2-1 shows an IMS 11 Open Database overview.



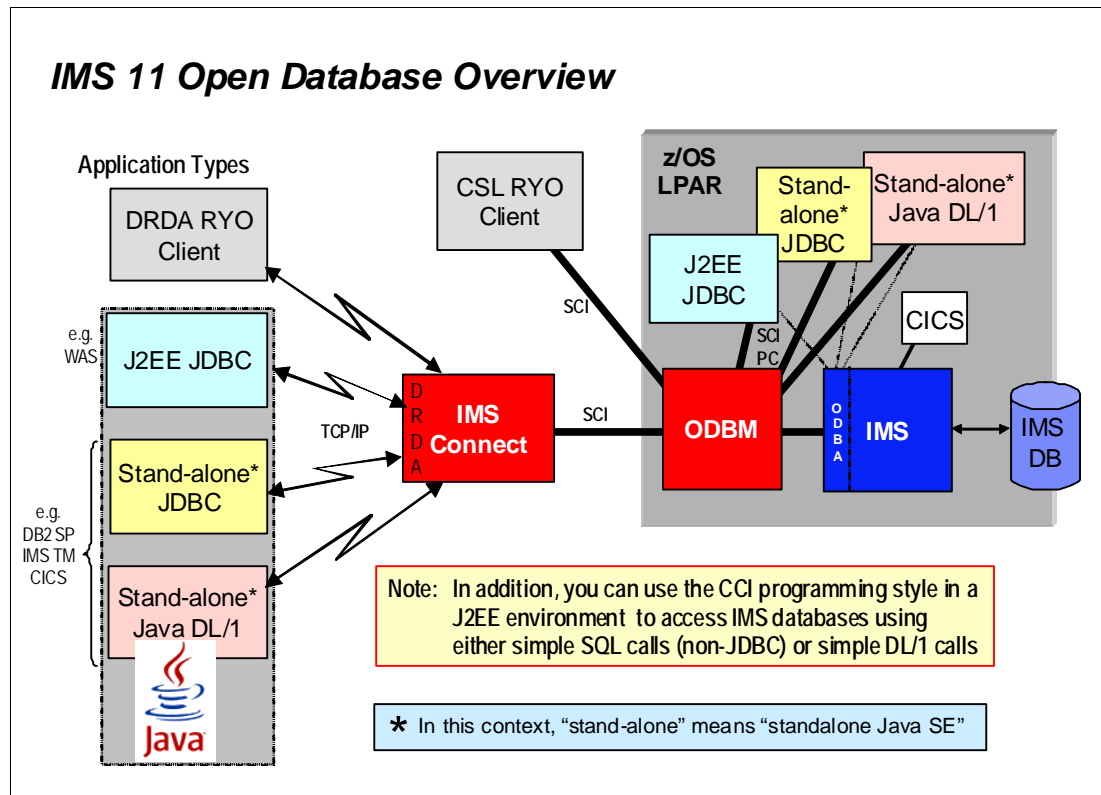


Figure 2-1 IMS 11 Open Database Overview

## 2.2 Accessing IMS DB in Versions 9 and 10

The interfaces to Online IMS DB in IMS 9 and 10 are:

- ▶ Open Data Base Access (ODBA) which is used by WAS/z and DB2 SP and with these Resource Recovery Services (RRS) is required to be used as the Sync Point coordinator.
- ▶ The transaction manager Customer Information Control System (CICS) uses CCTL as its interface with CICS itself being the sync point coordinator and there is no requirement for RRS.
- ▶ Both ODBA and CCTL use the Database Resource Adapter (DRA). Some IMS modules reside in ODBA application or CICS address spaces, in particular the DRA Startup Table (Assembled DFSPRP macro)
- ▶ IMS Java provides two IMS DB “Classic” Drivers:
  - For Java dependent regions, CICS, and DB2 Stored Procedures used with DLI calls or JDBC SQL
  - For J2EE Environments the interface used is the JDBC Resource Adapter.

Figure 2-2 shows methods of accessing Online IMS DB in V9 and V10.

## Ways of Accessing Online IMS DB in IMS 9/10

- (Non-java) Dependent Region (MPP or BMP)
- Java Dependent Region (JMP or JBP)
- (Non-java) ODBA (requires RRS)
  - e.g. COBOL DB2 Stored Procedure
- Java ODBA (requires RRS)
  - e.g. WAS/z or Java DB2 Stored Procedure
- (Non-java) CCTL
  - e.g. CICS DBCTL
- Java CCTL
  - e.g. Java CICS DBCTL
- Distributed WAS (requires WAS/z and RRS)

CICS is  
syncpoint  
co-ordinator

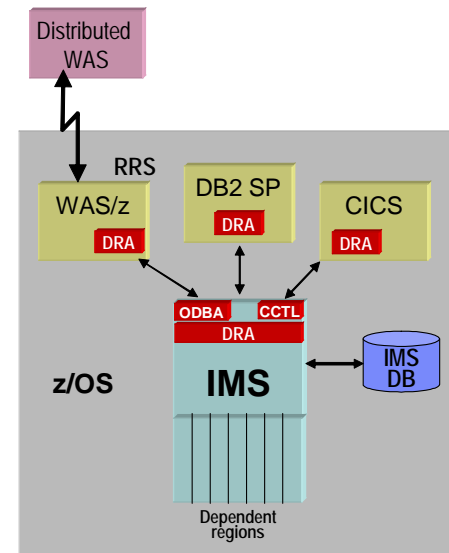


Figure 2-2 Methods of accessing online IMS DB in V9 and V10

There are several problems associated with the IMS 9/10 methods

- ▶ If an ODBA application terminates in the middle of a DL/1 call, there is the possibility of crashing IMS with a 113 abend
- ▶ When WAS/z is used it must run in same LPAR as IMS and this can effect machine capacity and licence charges
- ▶ If remote WebSphere solutions are being used WAS/z is needed and “helper” EJB running in WAS/z relays remote request to ODBA.
- ▶ Distributed applications accessing IMS DB, must execute on WAS there is no support for other distributed platforms

## 2.3 Evolution in IMS 11

The intent of this section is to show the topology prior to IMS 11 and illustrate the evolution to IMS 11, pointing out the enhancements at each step. As a point of fact, WebSphere Application Server (WAS) z/OS cannot take advantage of the cross-LPAR feature of ODBM unless WAS itself embraces Structured Call Interface (SCI). Applications can use the ‘out-of-the-box’ compatibility mode to use AERTDLI and have those calls routed to an ODBM which still prevents the U113 abend – but WAS and the ODBM address space will still need to be on the same LPAR. It is just an illustrative example showing what can be possible with WAS z/OS as an ODBM client.

The IMS 9 and 10 solution (whether or not we are talking about distributed or local access to IMS DB) leverages Open Database Access (ODBA) as the API to access IMS database

resources. ODBA is capable of making address space to address space calls (PC calls) in the same logical partition. This means that the ODBA modules need to be on the same LPAR as the IMS CTL region. These modules (ODBA) are loaded in the address space of the application, which is in turn loaded in the address space of the container. In this case the container is WAS. As a result of this the WAS installation has to be on the same LPAR as the IMS DB itself. There is no isolation

Figure 2-3 shows the architecture prior to IMS 11.

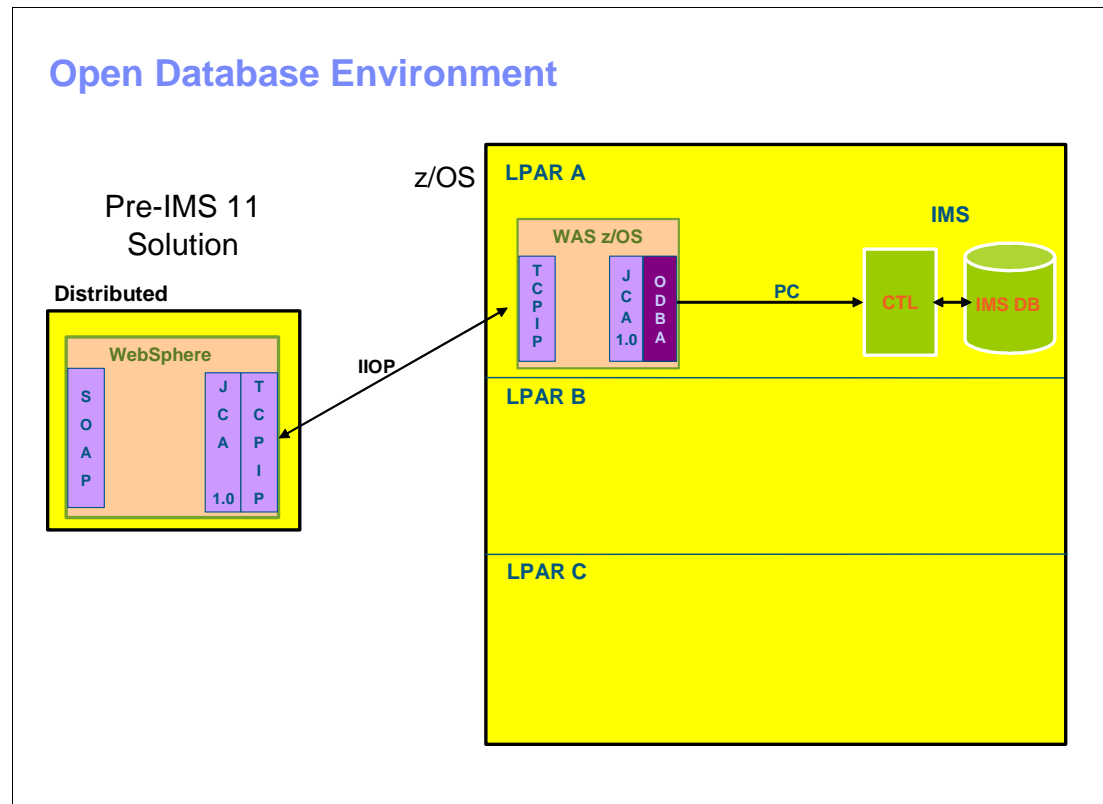


Figure 2-3 Architecture prior to IMS 11

By leveraging SCI, the applications can be on any LPAR in an IMSplex. SCI uses either PC or Cross-System Coupling Facility (XCF) calls to communicate with other SCI components. XCF allows calls to go across LPARs in an IMSplex. This allows applications (and their containers) to be isolated on their own LPARs.

Figure 2-4 shows the effect of leveraging the SCI.

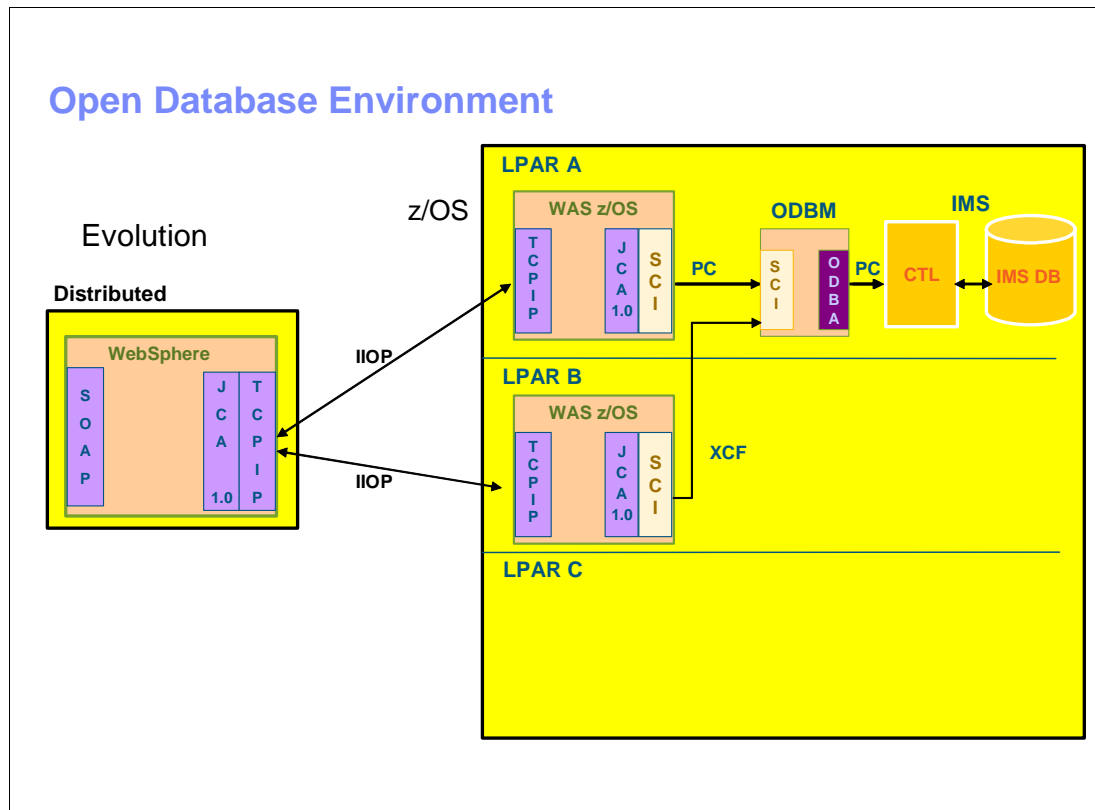


Figure 2-4 Effect of leveraging the SCI

IMS 11 has a new CSL address space to house the ODBA modules. This interface use SCI as its communication mechanism. The ODBA modules are no longer tightly coupled with the applications themselves (and therefore the containers).

Figure 2-5 shows the new CSL address space (IMS Connect).

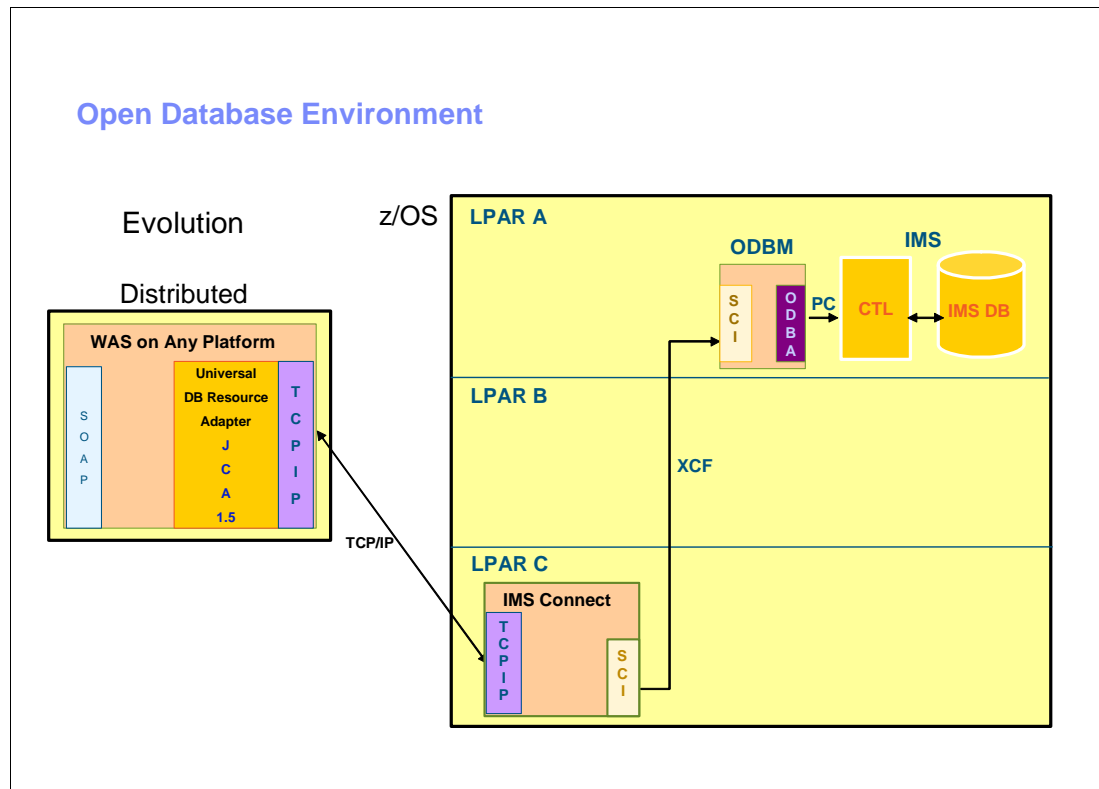


Figure 2-5 The new CSL address space (IMS Connect)

## 2.4 IMS 11 Architecture

This leads us to our real goal, which is to use IMS Connect as the complete gateway solution for IMS TM, OM, and now DB. IMS Connect is augmented to be an ODBM client. This allows distributed applications to use the Transmission Control Protocol/Internet Protocol (TCP/IP) to communicate with IMS Connect, which can then access any database in the entire IMSplex.

Figure 2-6 shows the final architecture.

IMS Connect has become the IMS Gateway to both IMS TM and IMS DB.

WebSphere and DB2 Stored Procedures no longer have to be on the same LPAR with IMS when they interface with the IMS ODBM address space. The ODBM address space must be on the same LPAR with IMS due to the use of the ODBA interface.

Distributed clients would now have the option of going directly to IMS Connect for IMS DB requests.

Existing DB Resource Adapter applications are unaffected by Open Database. In order to exploit Open Database from existing DB Resource Adapter applications, a migration to the JCA 1.5 programming model would have to be done.

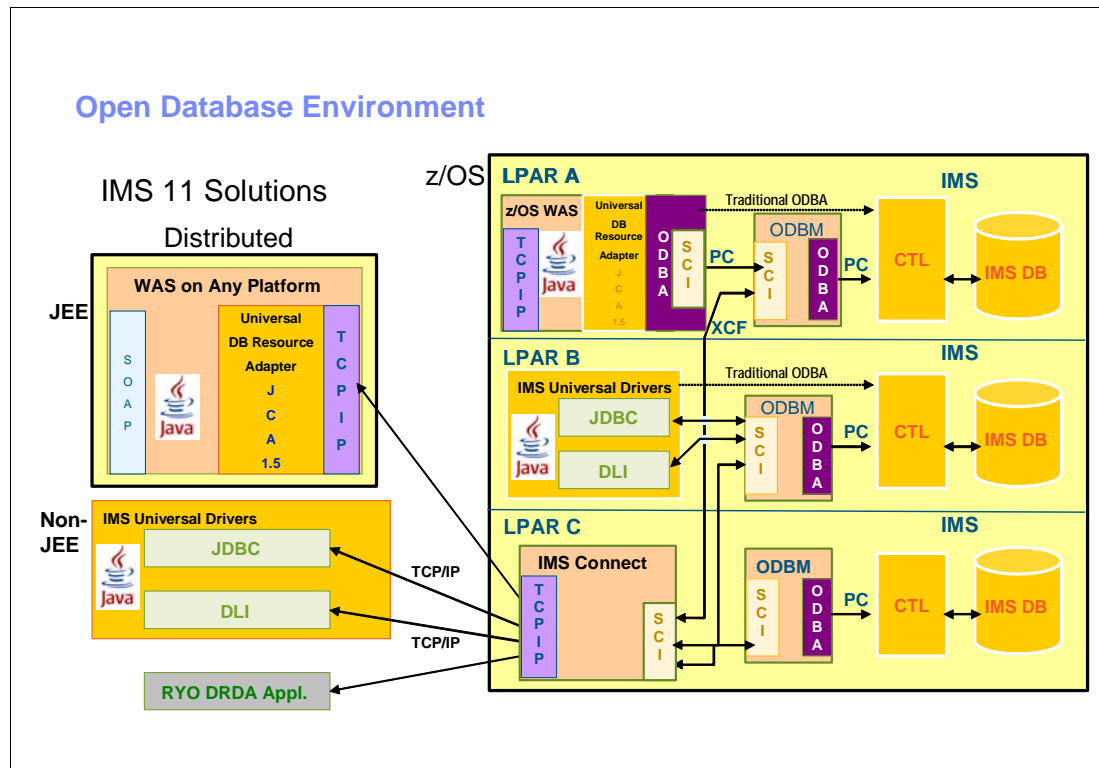


Figure 2-6 The final architecture

## 2.5 Open Database functions

In this section we describe the following Open Database functions:

- ▶ IMS Open Database uses DRDA
- ▶ Open DataBase Manager
- ▶ IMS Connect
- ▶ Distributed sync pointing
- ▶ Distributed Data Management

### 2.5.1 IMS Open Database uses DRDA

Distributed Relational Database Architecture (DRDA) is an open vendor-independent architecture that provides a set of protocols and functions allowing connectivity between a client and database servers. DRDA is used for IMS DB access through TCP/IP, IMS Connect and ODBM and provides:

- ▶ Communication protocol
- ▶ Two-Phase commit protocol
- ▶ Security

IMS Connect serves as the TCP/IP server or router for DRDA messages sent by way of the TCP/IP protocol.

ODBM routes the database connection requests received from IMS Connect to the IMS systems that are managing the requested database.

The Distributed Data Management (DDM) architecture provides the command and reply structure for accessing distributed databases.

ODBM translates the incoming database requests from the DDM protocol submitted by the IMS-provided connectors and user-written DRDA applications into the DL/I calls expected by IMS.

Manuals and further information about DRDA and DDM can be found at

<http://www.opengroup.org/dbiop>

## 2.5.2 Open DataBase Manager

Open DataBase Manager (ODBM) is a new optional IMSplex Common Service Layer (CSL) component that runs in its own address space. ODBM uses the Structured Call Interface (SCI) services of the CSL for communications and Operations Manager (OM) services of the CSL for command processing. ODBM provides distributed and local access to IMS databases that are managed by IMS DB systems running in either the DBCTL or DB/TM environments in an IMSplex.

One ODBM instance must be defined in the IMSplex to use ODBM functions. Each z/OS image can have more than one ODBM. If multiple instances of ODBM are defined in the IMSplex, any ODBM instance can perform work from any z/OS image in the IMSplex.

ODBM routes the database connection requests received from IMS Connect to the IMS systems that are managing the requested database. Before establishing the connection to the IMS system, ODBM translates the incoming database requests from the DDM protocol submitted by the IMS-provided connectors and user-written DRDA applications into the DL/I calls expected by IMS. When ODBM returns the IMS output to the client, ODBM translates the response to the DDM protocol. From the ODBM perspective, application programs that interact directly with ODBM, such as IMS Connect, are ODBM clients. Users can create their own ODBM clients by using the new ODBM CSLDMI API. ODBM client application programs can access databases that are managed by IMS DB on any LPAR in an IMSplex.

Independently and together with IMS Connect, ODBM supports various interfaces to ease the development of application programs that access IMS databases from many different distributed and local environments. Supported ODBM interfaces include:

- ▶ IMS Universal DB resource adapter
- ▶ IMS Universal JDBC driver
- ▶ IMS Universal DL/I driver
- ▶ The ODBA interface
- ▶ The ODBM CSLDMI interface for user-written ODBM client application programs

Figure 2-7 shows an IMS configuration that includes ODBM.

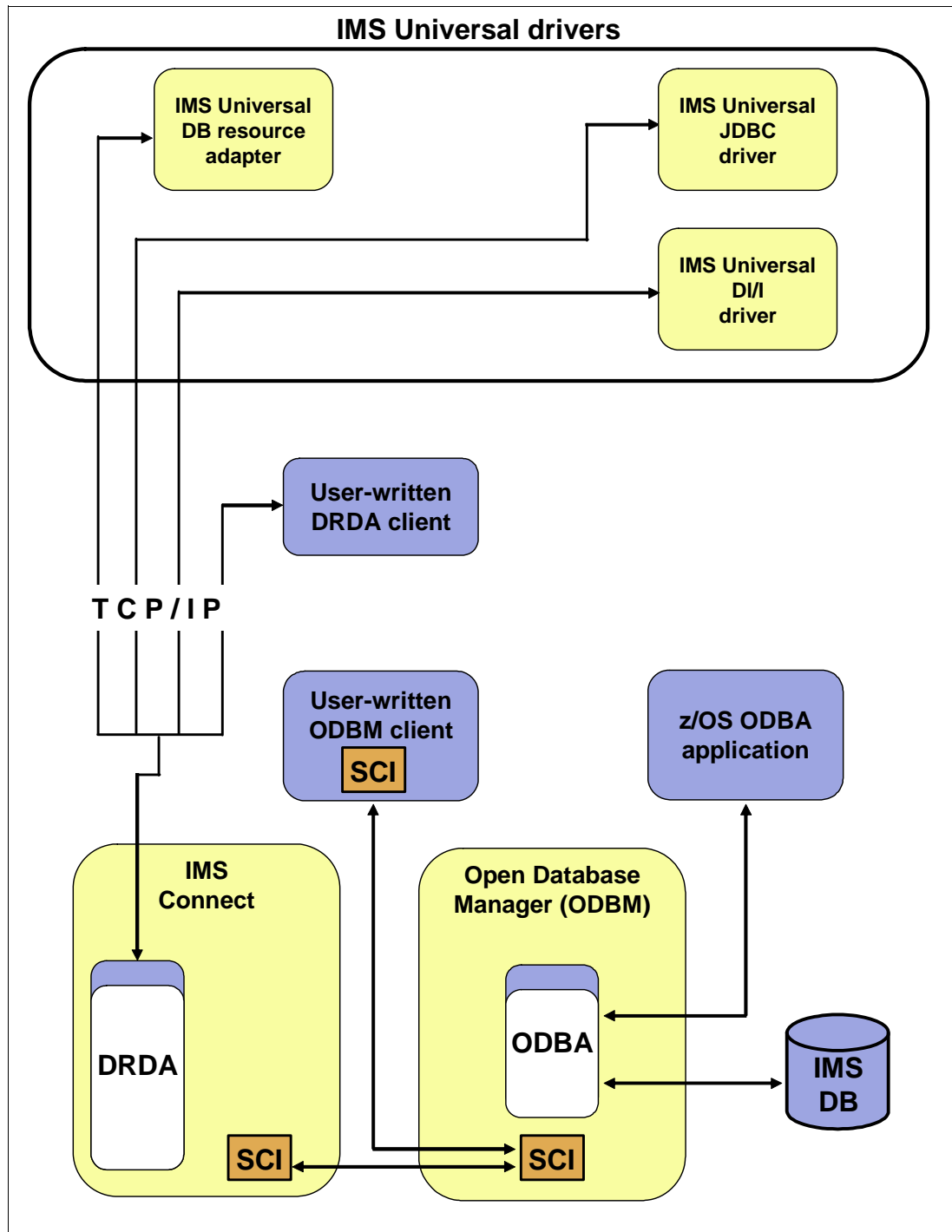


Figure 2-7 Overview of an IMS configuration that includes ODBM

Additionally as well as supporting the above interfaces, the following functions are provided by ODBM:

- Supports 1PC (one phase commit) with or without RRS as the sync point coordinator. If the ODBM parameter RRS=N is specified, the Database Resource Adapter (DRA) is used to communicate with IMS.



- ▶ Receives incoming database requests to IMS systems based on an alias name. Alias names for IMS systems are defined to ODBM on the CSLDCxxx PROCLIB member and specified on the incoming database requests by the client application programs.
  - If the client application program does not specify an alias name on a database access request, ODBM routes the request among multiple IMS systems by using a round-robin method of distribution, in which ODBM routes each incoming request to the active IMS systems that are defined to ODBM.
- ▶ Before establishing the connection ODBM translates the incoming database requests from the DDM protocol submitted by the IMS-provided connectors and user-written DRDA applications into the DLI calls expected by IMS.
- ▶ Translates responses to the client into the DDM protocol
- ▶ Enables ODBM client application programs to access databases from other LPARs within an IMSplex.
- ▶ Manages connections to ODBA
- ▶ Protects IMS control regions from the unexpected termination during DL/I processing of z/OS application programs that use the ODBA interface through ODBM.
- ▶ Functions as an RRMS resource manager capable of participating in RRS controlled sync point functioning for ODBA applications that are configured to use ODBM.
- ▶ Functions as a resource manager and issues the necessary calls to RRS for single-phase commit processing for local RRS transactions.
- ▶ Functions together with IMS Connect as a complete Distributed Relational Database Architecture (DRDA) target server for client application programs that use the DRDA specification.

Because ODBM and IMS Connect support DRDA, you can develop your own DRDA source application server that communicates with IMS by using the Distributed Data Management (DDM) Architecture commands that are part of the DRDA specification.

From the ODBM perspective, application programs that interact directly with ODBM, such as IMS Connect, are ODBM clients. Users can create their own ODBM clients by using the new ODBM CSLDMI API. ODBM client application programs can access databases that are managed by IMS DB on any LPAR in an IMSplex.

ODBM does not perform user authentication or authorization itself.

If you are using IMS Connect with ODBM, you can authenticate the user IDs on request messages before they reach ODBM by using IMS Connect security. In addition to being able to call RACF directly, IMS Connect provides the IMS DB Security user exit routine (HWSAUTH0) to facilitate the customizing of the security checking for communications with IMS DB.

You can have IMS check the authority of a user to allocate a PSB or access IMS resources by using APSB and Resource Access Control (RAS) security.

APSB security is enabled by specifying the ODBASE parameter. RAS security is specified by the ISIS parameter. Both the ODBASE and ISIS parameters can be specified on the IMS or DBC startup procedure or the DFSPBxxx PROCLIB member. If the ODBASE parameter is set to 'Y' whatever value is specified for the ISIS parameter is ignored.

When using APSB you need to do the following:

1. Define the PSBs that you want protected by RACF to the AIMS or Axxxxxxx general resource class (where xxxxxx is the value specified on the RCLASS= parameter of the IMS SECURITY macro).
2. Specify RCLASS=IMS | RACFCOM | RACFTERM | RASRACF | RAS on the IMS SECURITY macro at IMS system definition time.

When RAS is used the actions you need to perform depends on the value specified for the ISIS parameter. See Table 2-1.

Table 2-1 Options for defining RAS security for applications that use ODBA

Specifications	Actions to perform
ISIS=0   N and ODBASE=N	No action required. No PSB security checking is performed.
ISIS=R and ODBASE=N	Define the PSBs that you want protected by RACF to the IIMS or lxxxxxxx resource class, and then define the user IDs of the dependent region that you want authorized to access the PSBs. The ODBA support for IMS uses the security environment (ACEE) passed in the dependent region's task (TCBSENV), if present, or the dependent region's address space (ASXBSENV), if the ACEE is not present at the task level.
ISIS=C and ODBASE=N	Create a Resource Access Security exit routine that is named DFSRAS00. This routine must determine if the user is authorized to use the PSB.
ISIS=A and ODBASE=N	Define the PSBs that you want protected by RACF to the IIMS or lxxxxxxx resource class, and then define the user IDs of the dependent region that you want authorized to access the PSBs. The ODBA support for IMS uses the security environment (ACEE) passed in the dependent region's task (TCBSENV), if present, or the dependent region's address space (ASXBSENV), if the ACEE is not present at the task level. Create a Resource Access Security exit routine that is named DFSRAS00. This routine must determine if the user is authorized to use the PSB RACF is called first, and then the exit routine is called.

If a user-written ODBM client passes a security object for a security product such as RACF, ODBM invokes RACROUTE REQUEST=VERIFY to create an Accessor Environment Element (ACEE) for the APSB thread. IMS can then use the ACEE during APSB or RAS authorization for allocating PSBs or access to other resources.

## 2.5.3 IMS Connect

IMS Connect provides high performance TCP/IP and local z/OS communications between one or more IMS Connect clients and one or more IMS systems. IMS Connect, which previously provided access to only IMS Transaction Manager (IMS TM), through Open Transaction Manager Access (OTMA), in IMS 11 now also provides access to IMS DB, through ODBM. IMS Connect for Open Database enables:

- Distributed clients to exchange messages with IMS DB by using TCP/IP connections and ODBM.
- IMS operators that use the IMS Control Centre to issue commands to an IMSplex and receive command replies by using TCP/IP and the IMS Operations Manager (OM).

Communication with ODBM and OM requires the use of the IMS SCI.

With reference to WebSphere Application Server, WAS/z can run in same or different LPAR from IMS. When located in a different LPAR, it uses TCP/IP through IMS Connect.

- Distributed WAS does not require WAS/z, communication is through IMS Connect

- ▶ Distributed applications do not require a J2EE (e.g. WAS) environment

### **IMS Connect enhancements for Open Database support**

IMS Connect has had the following enhancements in support of IMS Open Database:

- ▶ IMS Connect Configuration member HWSCFGxx
  - New ODACCESS statement
    - DRDA ports, timeout value, IMSplex name etc.
- ▶ Changes to existing commands
  - VIEWHWS, VIEWDS, VIEWPORT
- ▶ New Commands
  - STARTOD, STOPOD, STARTIA, STOPIA, VIEWIA, SETOAUTO
- ▶ New User Exits
  - HWSROUT0 – Routing Exit for ODBM
    - can override the IMS alias and optionally select the ODBM target
  - HWSAUTH0 – Security Exit for ODBM
    - can perform the authentication of the userid

IMS Connect, with ODBM, supports the following types of clients in accessing IMS DB:

- ▶ Application programs using the IMS Universal DB resource adapter for the J2EE platform
- ▶ Application programs using the IMS Universal JDBC driver
- ▶ Application programs using the IMS Universal DL/I driver
- ▶ User-written client application programs using the open standard DRDA communications architecture

For IMS Connect clients, such as the IMS Universal drivers, that access databases that are managed by IMS DB in DBCTL and DB/TM environment, IMS Connect manages TCP/IP connections and routes incoming access requests among the instances of the ODBM and the IMS DB systems in an IMSplex.

IMS Connect is the TCP/IP server and front-end IMSplex message router for the IMS Universal drivers, which include:

- ▶ IMS Universal DB resource adapter for the J2EE platform
- ▶ IMS Universal JDBC driver
- ▶ IMS Universal DL/I driver

Because IMS Connect supports a subset of the DRDA protocol and, with ODBM, can be considered a DRDA target server, you can write application programs to the DRDA protocol directly; however, the IMS Universal DB resource adapter for the J2EE platform is the recommended API for accessing IMS databases through TCP/IP from a distributed environment.

IMS Connect support for the IMS Universal drivers includes support for global two-phase commit transactions.

IMS Connect supports communication with the IMS Universal drivers only on dedicated DRDA ports and only through shareable persistent sockets.

IMS Connect security support includes the IMS Connect DB Security user exit routine (HWSAUTH0), which can be used for greater control over the authentication of user IDs on connections that access IMS DB. RACF is also supported.

IMS Connect support for the IMS Universal drivers is defined by the ODACCESS configuration statement in the IMS Connect configuration PROCLIB member and requires at least one instance of ODBM running in the same IMSplex as IMS Connect.

Figure 2-8 shows an overview of IMS Connect support for IMS DB systems.

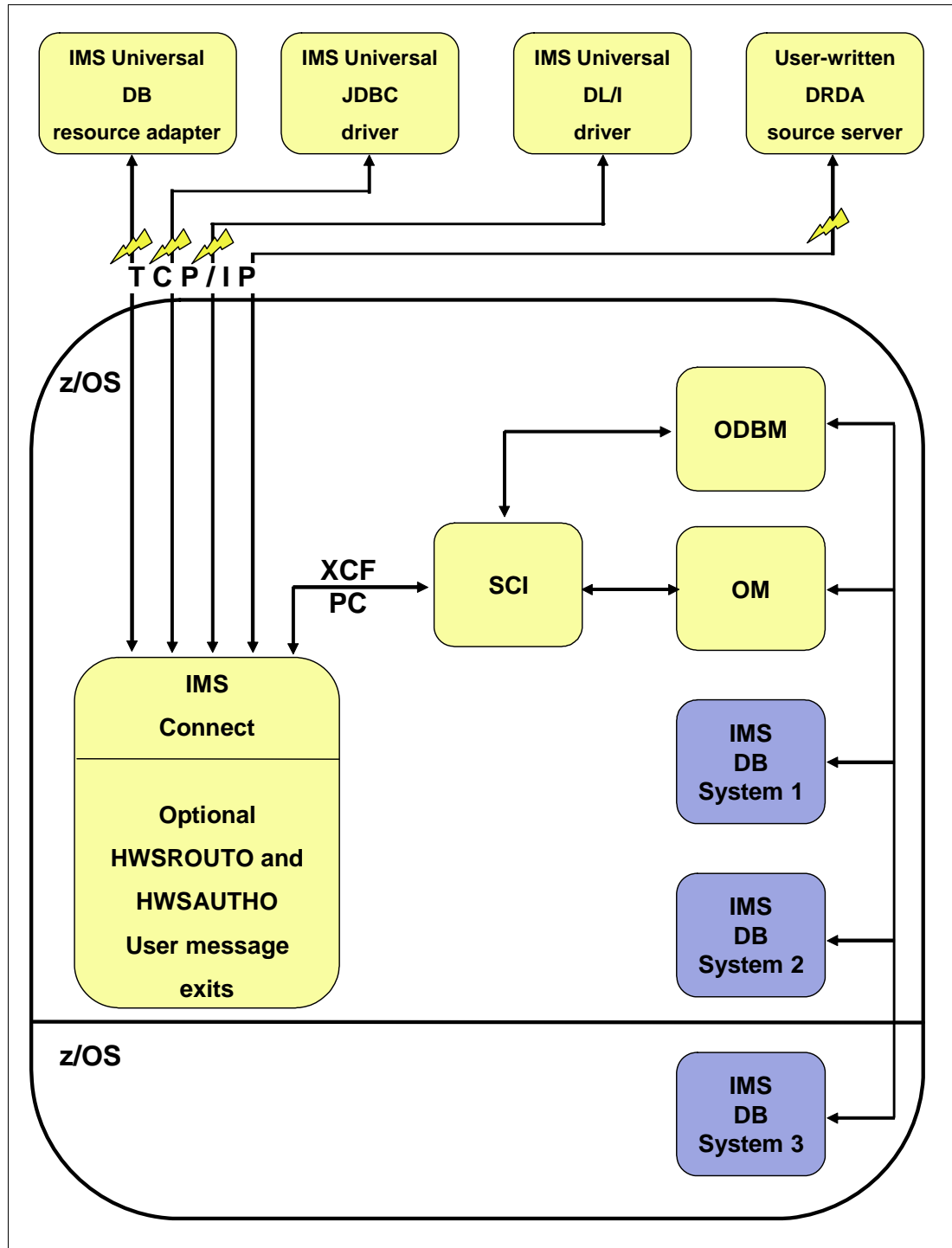


Figure 2-8 Overview of IMS Connect support for IMS DB systems

IMS Connect invokes the routing user exit first to allow the exit to select an ODBM and optionally override the IMS ALIAS.

- If the routing exit selects an ODBM, IMS Connect uses that ODBM and does not perform the round robin routing method.

- ▶ If the routing exit does not select an ODBM, IMS Connect selects an ODBM and perform the round robin routing method based upon the explicit ALIAS name as well as the blanked ALIAS name.
- ▶ If the routing exit overrides the IMS ALIAS, IMS Connect uses that IMS ALIAS.
- ▶ IMS Connect validates the ALIAS and the ODBM upon returning from the exit.

For clients that connect to IMS DB through ODBM, such as the IMS Universal drivers and clients using the Distributed Relational Database Architecture (DRDA), IMS Connect authenticates the user, but does not check the authority of the user to perform any actions.

To authenticate a user ID for an IMS DB client, IMS Connect can use the IMS Connect DB Security user exit routine (HWSAUTH0), a security product such as RACF, or both.

- ▶ HWSAUTH0 user exit is always called by IMS Connect, regardless of whether RACF or another security product is enabled. If RACF support is included in your IMS Connect configuration, IMS Connect calls the HWSAUTH0 user exit before invoking RACF.
- ▶ HWSAUTH0 performs the authentication of the userid/passticket of the ODBM client.
- ▶ HWSAUTH0 can override the input userid with a different userid and is able to provide a RACF groupid to be authenticated further by IMS Connect.
- ▶ HWSAUTH0 is a BPE type-1 user exit and is refreshable.
- ▶ Password is passed in the clear.

RACF is enabled in IMS Connect for IMS DB and IMS TM clients by specifying RACF=Y in the IMS Connect configuration member or by issuing the IMS Connect command SETRACF ON.

IMS Connect does not support Secure Sockets Layer (SSL) directly for clients that connect to IMS DB. To secure connections to IMS DB with SSL, use IBM z/OS Communications Server Application Transparent Transport Layer Security feature (AT-TLS). The use of AT-TLS is transparent to IMS Connect.

IMS Connect provides several features to help you manage RACF passwords. Some of these features only apply when IMS Connect is configured to call RACF directly.

- ▶ When IMS Connect is configured to call RACF directly, users of the user message exit routines HWSSMPL0, HWSSMPL1, and HWSJAVA0 can change RACF passwords by submitting a client message that includes a password change request keyword.
- ▶ IMS Connect supports mixed-case passwords.
- ▶ An alternative to the RACF password is a PassTicket. PassTicket allows you to communicate with a host without using a RACF password. When IMS Connect is configured to call RACF directly, you can use PassTicket to authenticate user IDs and log on to computer systems that contain RACF.

For information on these features refer to *IMS Version 11 Communications and Connections*, SC19-2433.

**Note:** If you configure IMS Connect to call RACF, you should evaluate the impact of the RACF calls on IMS Connect performance

## IMS Connect workload distribution

IMS Connect can also provide workload distribution by routing database connection requests to ODBM based on an alias name that is submitted by the client application program. To the client application, the alias name represents the IMS system, or data store, to which the

application program connects. Depending on the value of the alias name submitted, IMS Connect either routes the incoming connection request to a specific ODBM instance or distributes the incoming connection request to any available instance of ODBM in an IMSplex.

- ▶ ODBM clients can specify an IMS “ALIAS” in the message
  - Alias represents the IMS datastore that the client wants to send the message to
    - Multiple Alias names for an IMS datastore can be defined in the ODBM configuration member
- ▶ If the client sends a message with a blank alias, IMS Connect routes the message to an ODBM using a round robin algorithm
- ▶ If an alias points to multiple ODBMs, IMS Connect routes the message to one of those ODBMs using a round robin algorithm

## 2.5.4 Distributed sync pointing

Figure 2-9 shows how Open Database achieves cross LPAR transaction management.

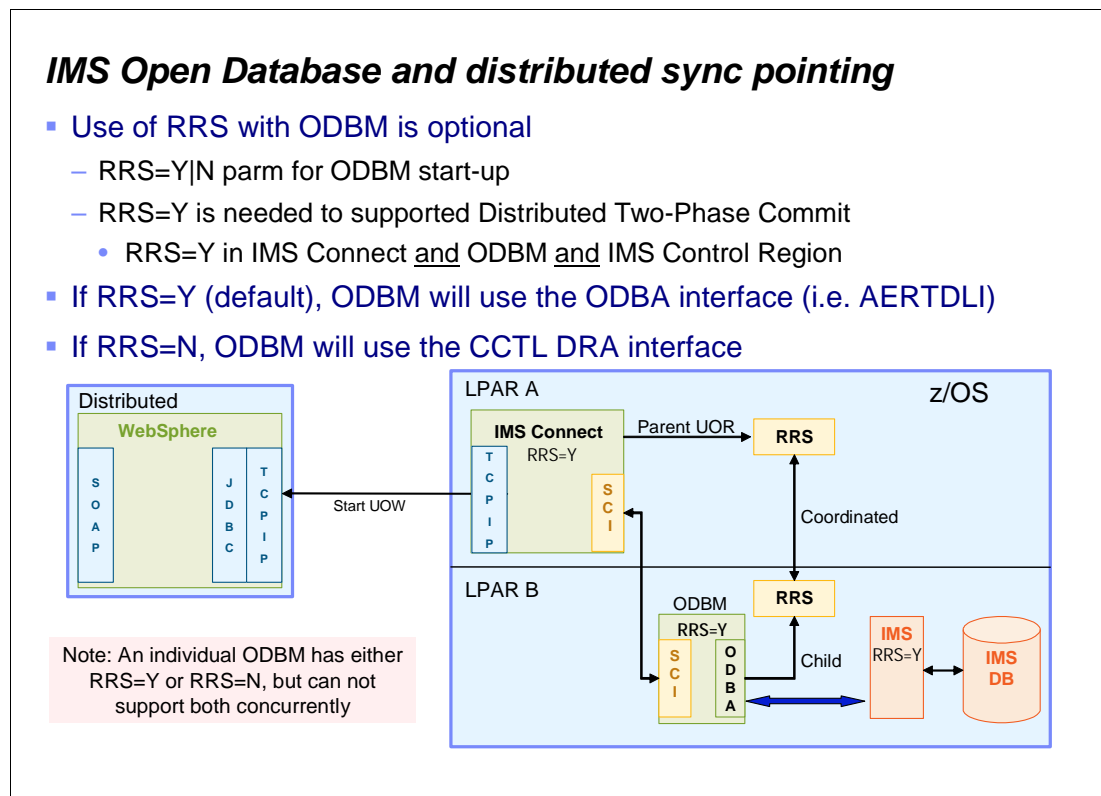


Figure 2-9 Open database cross LPAR transaction management

RRS=Y must be specified in the configuration for a global transaction.

When a client establishes a connection through IMS Connect to ODBM several things are done to establish a coordinated Unit of Work.

- ▶ First IMS Connect creates the parent Unit of Recovery.
- ▶ IMS Connect then sends the UR token to ODBM.
- ▶ ODBM then expresses interest in the UR as a child.
- ▶ At this point we have a coordinated Unit of Recovery established.

For each request to access IMS data, connection information on the IMS host system, port number, and a valid user ID and password must be supplied in order to establish communication with IMS. A socket connection is first established to connect to the host IMS Connect system. When IMS Connect receives the request, it proceeds to authenticate the user based on the supplied user ID and password. After successful authentication, necessary information on the socket, such as PSB name and IMS alias (database subsystem) is sent to ODBA in order to allocate the PSB to connect to the database. An actual connection to an IMS database is only established when a PSB is allocated. Authorization for a particular PSB is done by the ODBA component during the allocation of a PSB.

Distributed Syncpoint (global transaction) requires RRS on z/OS.

Use of RRS with ODBM is optional

- ▶ RRS=YIN parameter for ODBM start-up
  - If RRS=Y (also the default), ODBM uses the ODBA interface (i.e. AERTDLI)
  - If RRS=N, ODBM uses the DRA interface like CICS

Global transactions are not supported if RRS=N

Distributed Relational Database Architecture (DRDA) is set of protocols and functions providing connectivity between a client and database servers.

- ▶ Communication protocol
- ▶ Two-Phase commit protocol
- ▶ Security

DRDA is used for IMS DB access through TCP/IP, IMS Connect and ODBM

IMS Connect serves as the TCP/IP server or router for DRDA messages sent by way of the TCP/IP protocol.

## 2.5.5 Distributed Data Management

IMS supports the Distributed Data Management (DDM) architecture of the Distributed Relational Database Architecture (DRDA). You can develop your own source DDM server that communicates with the IMS target DDM server to provide access to databases managed by IMS DB in DBCTL and DB/TM IMS systems.

The IMS documentation for the DDM architecture includes only the DDM structures that are required to connect to and communicate with IMS and the DDM structures that have been changed or defined by IMS.

For the complete documentation of the DDM, see DRDA, Version 4, Volume 3: Distributed Data Management (DDM) Architecture, which is available from The Open Group at [www.opengroup.org](http://www.opengroup.org).

The DDM architecture includes the following elements or terms:

- ▶ Commands
- ▶ Command objects
- ▶ Reply objects
- ▶ Reply messages

Each term, whether it is a command, command object, reply object, parameter, or message, is represented by a codepoint, a hexadecimal value that represents and identifies the component in communication between a source server and the target server. For example,



the EXCSAT command is represented by X'1041', the EXCSATRD reply object is represented by X'1443', the SRVNAM parameter is represented by X'116D', and so on.

As an open standard, the DRDA specification requires that products that use the specification must conform to the conventions, protocols, standards, and so on, of its architecture. However, the DDM architecture that is a part of the DRDA specification allows products to create product-unique extensions, in which a product, such as IMS, uses a subset of the existing DDM-defined commands, parameters, and messages, as well as product-unique structures that are defined by the product. When creating a product-unique extension that has product-unique structures, the product must conform to the DDM architecture.

The product-unique extension for IMS conforms to both the DDM architecture and the DRDA specification. IMS uses a subset of the existing DDM-defined commands, parameters, and messages, as well as a variety of IMS-defined structures that conform to the DDM architecture, but are unique to IMS.

## 2.6 IMS 11 Universal Drivers

Figure 2-10 shows the new IMS Universal Drivers

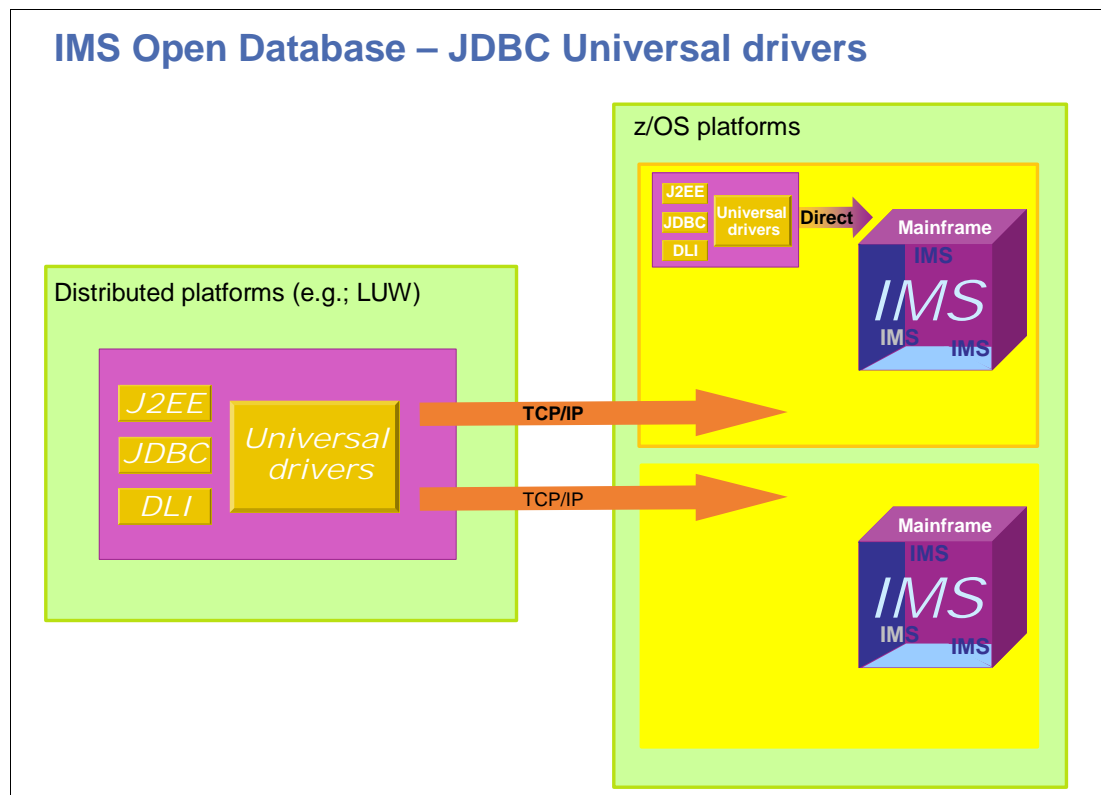


Figure 2-10 IMS Universal drivers

The Universal drivers have a framework capable of processing any of the three main programming models: J2EE, JDBC, DLI. The Universal drivers are able to connect to any IMS subsystem on any mainframe system. The same application can have active connections to any number of IMS systems on any number of mainframe installations.

When running in a distributed environment on a server such as WebSphere Application Server for distributed platforms, or in a remote z/OS environments on a server such as

WebSphere Application Server for z/OS, the IMS Universal drivers connect to IMS using a type-4 connection architecture, which supports TCP/IP communications and socket management.

When running locally on the same logical partition (LPAR) as IMS, the IMS Universal drivers connect to IMS by using a type-2 connection architecture, which supports direct communication with IMS through the IMS Open Database Access (ODBA) and IMS database resource adapter (DRA) interfaces.

WebSphere Application Server supports all of the IMS Universal drivers in both distributed and z/OS environments.

CICS and DB2 for z/OS support the IMS Universal JDBC driver and the IMS Universal DL/I driver.

The IMS Universal drivers are software components that provide Java applications with direct, non-transactional connectivity and access to IMS databases from z/OS and distributed environments through TCP/IP. Applications using the IMS Universal drivers can reside on the same logical partition (LPAR) or on a different LPAR from the IMS databases.

## Programming approaches

The IMS Universal drivers provide an application programming framework that offers multiple options for access to IMS data. These programming options include:

- ▶ **IMS Universal DB resource adapter**  
Provides connectivity to IMS databases from a Java Platform, Enterprise Edition (Java EE) environment, and access to IMS data using the Common Client Interface (CCI) and Java Database Connectivity (JDBC) interfaces.
- ▶ **IMS Universal JDBC driver**  
Provides a stand-alone JDBC 3.0 driver for making SQL-based database calls to IMS databases.

## IMS Universal DL/I driver

Provides a stand-alone Java API for writing granular queries to IMS databases using programming semantics similar to traditional DL/I calls.

## Open standards

The IMS Universal drivers are built on the following industry open standards and interfaces:

- ▶ **Java EE Connector Architecture (JCA)**  
JCA is the Java standard for connecting Enterprise Information Systems (EISS) such as IMS into the Java EE framework. Using JCA, you can simplify application development and take advantage of the services that can be provided by a JAVA EE application server, such as connection management, transaction management and security management. The Common Client Interface (CCI) is the interface in JCA that provides access from JAVA EE clients such as Enterprise JavaBean (EJB) applications, JavaServer Pages (JSP), and Java servlets, to backend IMS subsystems.
- ▶ **Java Database Connectivity (JDBC)**  
JDBC is the SQL-based standard interface for database access. It is the industry standard for database-independent connectivity between the Java programming language and any database that has implemented the JDBC interface.
- ▶ **Distributed Relational Database Architecture (DRDA) specification**

DRDA is an open architecture that enables communication between applications and database systems on disparate platforms. These applications and database systems can be provided by different vendors and the platforms can be different hardware and software architectures. DRDA provides distributed database access with built-in support for distributed, two-phase commit transactions.

- Distributed and local connectivity with the IMS Universal drivers

The IMS Universal drivers support distributed (type-4) and local (type-2) connectivity to IMS databases.

### **Distributed connectivity with the IMS Universal drivers**

With type-4 connectivity, the IMS Universal drivers can run on any platform that supports TCP/IP and a Java Virtual Machine (JVM), including z/OS. To access IMS databases using type-4 connectivity, the IMS Universal drivers first establish a TCP/IP-based socket connection to IMS Connect. IMS Connect is responsible for routing the request to the IMS databases using the Open Database Manager (ODBM), and sending the response back to the client application. The DRDA protocol is used internally in the implementation of the IMS Universal drivers. You do not need to know DRDA to use the IMS Universal drivers.

The IMS Universal drivers support two-phase commit (XA) transactions with type-4 connectivity. IMS Connect builds the necessary Recovery Resource Services (RRS) structure to support the two-phase commit protocol. If two-phase commit transactions are not used, RRS is not required.

When establishing a connection to IMS, the `driverType` connection property must be set to indicate distributed (type-4) connectivity to IMS.

After successful authentication, the IMS Universal drivers sends other socket connection information, such as program specification block (PSB) name and IMS database subsystem, to IMS Connect and ODBM in order to allocate the PSB to connect to the database.

A connection to an IMS database is established only when a program specification block (PSB) is allocated. Authorization for a particular PSB is done by the ODBM component during the allocation of a PSB.

The IMS Universal drivers support connection pooling with type-4 connectivity, which limits the time that is needed for allocation and deallocation of TCP/IP socket connections. To maximize connection reuse, only the socket attributes of a connection are pooled. These attributes include the IP address and port number that the host IMS Connect is listening on. As a result, the physical socket connection can be reused and additional attributes can be sent on this socket in order to connect to an IMS database. When a client application of the IMS Universal drivers using type-4 connectivity makes a connection to IMS, this means:

- A one-to-one relationship is established between a client socket and an allocated PSB that contains one or more IMS databases.
- A one-to-many relationship is established between IMS Connect and the possible number of database connections it can handle at one time.
- IMS Connect does the user authentication.
- ODBM ensures that the authenticated user is authorized to access the given PSB.

You can also use the IMS Universal drivers with type-4 connectivity if your Java clients are running in a z/OS environment but are located on a separate logical partition from the IMS subsystem. Use type-4 connectivity from a z/OS environment if you want to isolate the application runtime environment from the IMS subsystem environment.

## Local connectivity with the IMS Universal drivers

Local (or type-2) connectivity with the IMS Universal drivers is targeted for the z/OS platform and runtime environments. Use type-2 connectivity when connecting to IMS subsystems in the same logical partition (LPAR).

Table 2-2 shows the z/OS runtime environments that support client applications of the IMS Universal drivers using type-2 connectivity.

*Table 2-2 z/OS runtime environment support for IMS Universal drivers with type-2 connectivity*

z/OS Runtime Environment	IMS Universal Drivers with Type-2 connectivity supported
WebSphere Application Server for z/OSS	IMS Universal DB resource adapter
IMS Java dependent regions (JMP and JBP regions), DB2 for z/OS stored procedures and CICS	IMS Universal DL/I driver IMS Universal JDBC driver

The DRDA protocol is not used to establish type-2 connectivity. Instead, type-2 connectivity from WebSphere Application Server for z/OS and DB2 for z/OS stored procedures environments is established using Open Database Access (ODBA). Type-2 connectivity from a CICS environment uses the database resource adapter (DRA).

Because it runs on the same LPAR as the IMS subsystem, during connection time, a client application of the IMS Universal drivers using type-2 connectivity does not need to supply an IP address, port number, user ID, or password. The driverType property must be set to indicate local (type-2) connectivity to IMS.

## RRSLocalOption connectivity type

In addition to type-4 and type-2 connectivity, the RRSLocalOption connectivity type is supported by the IMS Universal DB resource adapter running on WebSphere Application Server for z/OS. With RRSLocalOption connectivity, applications using the IMS Universal DB resource adapter do not issue commit or rollback calls. Instead, transaction processing is managed by WebSphere Application Server for z/OS.

## Comparison of IMS Universal drivers programming approaches for accessing IMS

Depending on your IT infrastructure, solution architecture, and application design, choose the IMS Universal drivers programming approach that is best for your development scenario.

Table 2-3 lists the recommended IMS Universal drivers programming approach to use, based on the application programmer's choice of application platform, data access method, and transaction processing option.

Table 2-3 Comparison of programming approaches

Application platform	Data access method	Transaction processing required	Recommended approach
WebSphere Application Server for distributed platforms or WebSphere Application Server for z/OS	CCI programming interface to perform SQL or DL/I data operations.	Local transaction processing only.	Use the IMS Universal DB resource adapter with local transaction support (imsudbLocal.rar), and make SQL calls with the SQLInteractionSpec class or DL/I calls with the DLInteractionSpec class.
	CCI programming interface to perform SQL or DL/I data operations.	Two-phase (XA) commit processing* or local transaction processing.	Use the IMS Universal DB resource adapter with XA transaction support (imsudbXA.rar), and make SQL calls with the SQLInteractionSpec class or DL/I calls with the DLInteractionSpec class.
	JDBC programming interface to perform SQL data operations.	Local transaction processing only.	Use the IMS Universal JCA/JDBC driver version of the IMS Universal DB resource adapter with local transaction support (imsudbJLocal.rar), and make SQL calls with the JDBC API
	JDBC programming interface to perform SQL data operations.	Two-phase (XA) commit processing* or local transaction processing.	Use the IMS Universal JCA/JDBC driver version of the IMS Universal DB resource adapter with XA transaction support (imsudbJXA.rar), and make SQL calls with the JDBC API.
Standalone Java application (outside a Java EE application server) that resides on a distributed platform or a z/OS platform	JDBC programming interface to perform SQL data operations.	Two-phase (XA) commit processing** or local transaction processing.	Use the IMS Universal JDBC driver (imsudb.jar), and make SQL calls with the JDBC API.
	Traditional DL/I programming semantics to perform data operations.	Two-phase (XA) commit processing** or local transaction processing.	Use the IMS Universal DL/I driver (imsudb.jar), and make DL/I calls with the PCB class.
Standalone non-Java application that resides on a distributed platform or a z/OS platform	Data access using DRDA protocol.	Two-phase (XA) commit processing or local transaction processing.	Use a programming language of your choice to issue DDM commands to IMS Connect. The application programmer is responsible for implementing the two-phase commit mechanism.
<p>* XA transaction support is available only with type-4 connectivity.</p> <p>** The driver is enabled for local and XA transactions, but the application programmer is responsible for implementing the two-phase commit mechanism. XA transaction support is available only with type-4 connectivity</p>			

## Generating the runtime Java metadata class using the IMS Enterprise Suite DLIModel utility plug-in

To connect to an IMS database using the IMS Universal drivers or the IMS classic JDBC driver provided by the classic Java APIs for IMS, you need to include on your Java classpath the Java metadata class that provides the database view.

The Java metadata class is a subclass of `com.ibm.ims.db.DLIDatabaseView` that is generated using the IMS Enterprise Suite DLIModel utility plug-in. The Java metadata class represents the application view information specified by a program specification block (PSB) and its

related Program Control Blocks (PCBs). The Java metadata class provides a one-to-one mapping to the segments and fields defined in the PSB.

To generate the metadata class, use the DLIModel utility plug-in to import the application PSB source and related DBD source files. The Java metadata class must be compiled and made available through the classpath for any Java application attempting to access IMS data using that PSB.

During database connection setup, pass the name of this metadata class to the resource adapter or JDBC driver. The Java metadata class is used at runtime by the IMS Universal drivers and the IMS classic JDBC driver to process both SQL and Java-based DL/I calls.

Chapter 4, “Generating IMS metadata class with IMS Enterprise Suite DLIModel Utility” on page 75 provides the implementation details.



## System environment

IMS Open Database uses several IMS components that all interface with each other when a Java application calls IMS to retrieve data. In this chapter, we detail the setup, configuration and management of each component. In addition, we discuss how to secure access to IMS data when using the IMS Open Database capability. We cover these topics in the following sequence:

- ▶ Required environment setup for IMS Open Database
- ▶ Common Service Layer components
  - Base Primitive Environment configuration
  - Structured Call Interface
  - Operations Manager
  - Open Database Manager
    - ODBM commands
- ▶ IMS Connect
  - First-time implementation: setup and configuration
  - Modifying existing IMS Connect definitions for IMS Open DB Support
  - IMS Connect considerations for IMSplex
- ▶ Using IMS applications to help set up CSL and IMS Connect
  - Installation Verification Program
  - Syntax Checker
- ▶ Security considerations

## 3.1 Required environment setup for IMS Open Database

In order for a Java application to access IMS data, it must send a request to IMS in a format that IMS can understand and in turn, IMS must return the requested data in a format that the application can understand. The IMS Enterprise Suite DLIModel utility plug-in translates your existing IMS source data, consolidating it into a class file. Using this class file along with the IMS Universal Drivers now available in IMS 11, a Java application can speak the language of IMS and retrieve the data using the IMS Open Database capability.

**Note:** The installation, setup and use of the IMS Enterprise Suite DLIModel utility plug-in is covered separately in Chapter 4, “Generating IMS metadata class with IMS Enterprise Suite DLIModel Utility” on page 75.

As for the flow of a Java application’s request for IMS data, it first sends a message to IMS Connect over TCP/IP using DRDA protocol. IMS Connect then interfaces with the Open Database Manager (ODBM) using the Structured Call Interface (SCI). ODBM in turn uses SCI to route the request to the IMS that contains the requested data so that the data can be retrieved. The ODBM component is the key capability that enables this data access, whose configuration can be dynamically updated using Operations Manager (OM) commands known as type-2 commands, which we discuss later.

**Important:** The address spaces associated with the components mentioned here need to be started in the following order:

1. SCI
2. OM
3. IMS control region
4. ODBM
5. IMS Connect

Working together, these components bridge the gap between the more modern Java development environment and the data that it requires, contained on the mainframe in IMS. We now explore how to setup, configure, and enable these components, beginning with the Common Service Layer.

For details on the architecture of IMS Open Database, see Chapter 2, “Open Database architecture” on page 17.

## 3.2 Common Service Layer components

The Common Service Layer (CSL) is an architecture that simplifies IMSplex management by providing a single point of control from a single system image, as well as plex-wide resource sharing. The Open Database capability introduces a new CSL address space called Open Database Manager (ODBM), which handles incoming database requests from distributed applications. Like other IMSplex members, the ODBM address space communicates with other members using SCI, another CSL component mentioned in the previous introduction section. Finally, the new ODBM address space can be queried and updated via type-2 commands, which are sent to IMS using the Operations Manager (OM) CSL component.

In this section, we cover the setup for the CSL address spaces used with IMS Open Database: SCI, OM and ODBM. Each of these are based on the Base Primitive Environment



(BPE), and therefore you must specify a BPE configuration member in each component's startup JCL.

**Note:** In addition to completing setup for each individual CSL address space, you must also define the name of your IMSplex in the DFSDFxxx PROCLIB member with the IMSPLEX parameter. Each CSL address space has an initialization member that also contains an IMSPLEX parameter, which should match the IMSplex name that you specified in the DFSDFxxx member. You can also specify this value in the DFSCGxxx PROCLIB member.

The xxx suffix of DFSDFxxx will be specified on the CSLG startup parameter of IMS, so IMS will know which member will contain the CSL definitions (either DFSDFxxx or DFSCGxxx). If you specify your CSL definitions in both of these PROCLIB members, DFSCGxxx will take precedence. For instructions on how to find a sample member definition, refer to the section entitled "OM, SCI, and DFSCGxxx samples" on page 66.

This section includes details about each of the address spaces required to use the IMS Open Database capability:

- ▶ Base Primitive Environment configuration
- ▶ Structured Call Interface
- ▶ Operations Manager
- ▶ Open Database Manager and commands used to dynamically manage it

### 3.2.1 Base Primitive Environment configuration

The Base Primitive Environment (BPE) configuration member is used to enable tracing and defines the BPE execution environment values for running CSL address spaces as well as the IMS Connect address space. BPE trace records can be written to internal (memory only) trace tables and to external data sets. Use the TRCLEV parameter to indicate the type, level and number of storage pages allocated (optional) for the trace table. The default setting is for the records to be written internally. The sample BPE configuration member for SCI and OM, named BPECONFG, takes this default in the bottom portion of Example 3-1. In Example 3-5 and Example 3-7 we later reference the BPECONFG configuration member with the BPECFG= parameter (respectively) in our SCI and OM startup procedures.

*Example 3-1 Sample BPE configuration member for SCI and OM address spaces*

```
*****
* BPE CONFIGURATION FILE FOR SCI AND OM (BPECONFG) *
*****
LANG=ENU                                /* LANGUAGE FOR MESSAGES */
                                         /* (ENU = U.S. ENGLISH) */
#
# DEFINITIONS FOR BPE SYSTEM TRACES
#
TRCLEV=(*,LOW,BPE)                      /* DEFAULT ALL TRACES TO LOW */
# NOTE: KEEP THE FOLLOWING FOR COMPATIBILITY WITH 6.1 BPE
TRCLEV=(STG,LOW,BPE)                    /* STORAGE TRACE */
TRCLEV=(CBS,LOW,BPE)                    /* CONTROL BLK SRVCS TRACE */
TRCLEV=(DISP,LOW,BPE)                   /* DISPATCHER TRACE */
TRCLEV=(AWE,LOW,BPE)                    /* AWE SERVER TRACE */
TRCLEV=(LATC,LOW,BPE)                   /* LATCH TRACE */
TRCLEV=(SSRV,LOW,BPE)                   /* SYSTEM SERVICES TRACE */
TRCLEV=(USRX,LOW,BPE)                   /* USER EXIT SERVICES TRACE */
```

```
#
# DEFINITIONS FOR OM/SCI TRACES
#
TRCLEV=(*,LOW,OM)           /* DEFAULT OM TRACES TO LOW */
TRCLEV=(*,LOW,SCI)          /* DEFAULT SCI TRACES TO LOW */
```

---

We have chosen to use a different BPE configuration member named BPEODBM for initializing our ODBM address space, which is shown below in Example 3-2. Later, in Example 3-10, you will see that we reference this configuration member with the BPECFG= parameter in our ODBM startup procedure.

*Example 3-2 Sample BPE configuration member for ODBM address space*

---

```
*****
* CONFIGURATION FILE FOR BPE FOR ODBM (BPEODBM) *
*****
LANG=ENU                      /* LANGUAGE FOR MESSAGES */
                              /* (ENU = U.S. ENGLISH) */

#
# DEFINITIONS FOR BPE SYSTEM TRACES
#
TRCLEV=(*,HIGH,BPE,PAGES=20) /* DEFAULT TRACES TO HIGH */
TRCLEV=(STG,MEDIUM,BPE)     /* STORAGE TRACE */
TRCLEV=(CBS,MEDIUM,BPE)     /* CONTROL BLK SRVCS TRACE */
TRCLEV=(DISP,HIGH,BPE)       /* DISPATCHER TRACE */
TRCLEV=(AWE,HIGH,BPE)        /* AWE SERVER TRACE */
TRCLEV=(SSRV,HIGH,BPE)       /* SYSTEM SERVICE TRACE */

#-----#
# DEFINITIONS FOR ODBM TRACES #
#-----#
TRCLEV=(*,HIGH,ODBM)         /* SET DEFAULT FOR ALL ODBM */
                              /* TRACES TO HIGH. */
```

---

Next, we include here in Example 3-3 a final sample of a BPE configuration member named BPECFGIV that we later use when initializing IMS Connect. To see how this member is specified in our IMS Connect startup procedure, refer to Example 3-12 on page 62.

*Example 3-3 Sample BPE configuration member for IMS Connect address space*

---

```
*****
* CONFIGURATION FILE FOR BPE FOR IMS CONNECT (BPECFGIV) *
*****
LANG=ENU                      /* LANGUAGE FOR MESSAGES */
                              /* (ENU = U.S. ENGLISH) */

#
# DEFINITIONS FOR BPE SYSTEM TRACES
#
TRCLEV=(*,HIGH,BPE,PAGES=20) /* DEFAULT TRACES TO HIGH */
TRCLEV=(STG,MEDIUM,BPE)     /* STORAGE TRACE */
TRCLEV=(CBS,MEDIUM,BPE)     /* CONTROL BLK SRVCS TRACE */
TRCLEV=(DISP,HIGH,BPE)       /* DISPATCHER TRACE */
TRCLEV=(AWE,HIGH,BPE)        /* AWE SERVER TRACE */
TRCLEV=(SSRV,HIGH,BPE)       /* SYSTEM SERVICE TRACE */

#
# DEFINITIONS FOR IMS CONNECT TRACES
#
```

```

TRCLEV=(*,HIGH,HWS,PAGES=20)      /* DEFAULT TRACES TO HIGH */
TRCLEV=(HWSI,HIGH,HWS,PAGES=100)   /* OTMA COMM ACTIVITY TRACE */
TRCLEV=(HWSN,HIGH,HWS,PAGES=100)   /* LOCAL OPT DRIVER ACTIVITY */
TRCLEV=(HWSW,HIGH,HWS,PAGES=100)   /* TCP/IP DRIVER ACTIVITY */
TRCLEV=(OTMA,HIGH,HWS,PAGES=100)   /* XCF CALLS TRACE */
TRCLEV=(TCPI,HIGH,HWS,PAGES=100)   /* TCP/IP CALLS TRACE */

```

**Note:** The IVP application contains a job that defines a sample BPE configuration member and adds it to IMS PROCLIB. Refer to “BPE, ODBM and IMS Connect samples” on page 66 to determine how to locate this job within the IVP.

### 3.2.2 Structured Call Interface

Structured Call Interface (SCI) is the component of the CSL that enables IMSplex members to communicate with one another. Therefore, it is required that the SCI address space is started before any other IMSplex address spaces are started on a z/OS image. Each member must first register with SCI before it is allowed to join the IMSplex, even in the case of a single IMS system. We now discuss SCI implementation.

#### Configuration

The SCI initialization member CSLSIxxx must first be defined and configured. You must select a suffix to use with this member which you will later reference on the SCI startup procedure. Refer to Example 3-4 for a sample configuration of the SCI initialization member, which we have named CSLSI000.

*Example 3-4 Sample configuration of the SCI initialization member*

```

*-----*
* SCI INITIALIZATION PROCLIB MEMBER - CSLSI000 *
*-----*
ARMRST=N,          /* SHOULD ARM RESTART SCI ON FAILURE */
SCINAME=SCI1,      /* SCI NAME (SCIID = SCI1SC) */
IMSPLEX(NAME=PLEXB) /* IMSPLEX NAME (CSLPLEXB) */
*-----*
* END OF MEMBER CSLSI000 *
*-----*

```

The parameters that you see defined in the sample are required. There is also an optional FORCE parameter associated with global interface storage that you could specify here, but it is not shown for simplicity. Let's examine our specifications in a bit more detail here:

- ▶ **ARMRST=N:** the SCI address space is not automatically restarted by the z/OS Automatic Restart Manager (ARM) capability in the event that a system abend occurs
- ▶ **SCINAME=SCI1:** the SCI address space name SCI1 (can be 1-6 characters) will be used to define an SCIID of SCI1SC for use in SCI processing
- ▶ **IMSPLEX(NAME=PLEXB):** the IMSplex group name is PLEXB (can be 1-5 characters) and should match the IMSplex group name specified in the OM, ODBM, IMS initialization and IMS Connect configuration members; note that the characters “CSL” will be attached to the beginning of the IMSplex group name to create the ultimate name of CSLPLEXB

There must be one SCI address space on each z/OS image where the CSL is active, to allow communication with other components. These components can reside both within the same logical partition (LPAR) and on other z/OS images on other LPARs. For more information

about configuring the SCI initialization member, refer to *IMS Version 11 System Definition*, GC19-2444.

**Note:** The IVP application contains a job that defines a sample SCI initialization member and adds it to IMS PROCLIB. Refer to “OM, SCI, and DFSCGxxx samples” on page 66 to determine how to find this job within the IVP.

### Starting the SCI address space

To start the SCI address space as a task, execute the SCI procedure and specify the xxx suffix of the configuration member CSLSIxxx on the SCIINIT= parameter. Refer to Example 3-5 for a sample procedure that starts SCI as a task.

*Example 3-5 Sample SCI startup procedure JCL*

---

```

/*-----*
/*  SCI                                         *
/*-----*
/*  PARAMETERS:                               *
/*  BPECFG  - NAME OF BPE MEMBER               *
/*  SCIINIT - SUFFIX FOR YOUR CSLSIxxx MEMBER  *
/*  ARMRST  - INDICATES IF ARM SHOULD BE USED  *
/*  SCINAME - NAME OF THE SCI BEING STARTED    *
/*-----*
/*
//IEFPROC EXEC PGM=BPEINIO,REGION=3000K,
//  PARM=('BPECFG=BPECFG','BPEINIT=CSLSINIO','SCIINIT=000',
//        'ARMRST=N','SCINAME=SCI1')
/*
//STEPLIB DD DSN=IMS11B.SDFSRESL,DISP=SHR
/*
//PROCLIB DD DSN=IMS11B.PROCLIB,DISP=SHR
/*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
/*

```

---

You must also designate the BPE configuration and initialization members associated with the SCI address space with the BPECFG and BPEINIT parameter values, respectively. We have specified SCIINIT=000 in our sample, so the SCI initialization member CSLSI000 is read when the procedure is invoked. You can override certain parameter values defined in the CSLSI000 initialization member by specifying them here. For example, the ARMRST and SCINAME parameters specified in this member can override the values included in the SCI initialization member. Had the values shown in our sample procedure differed from those specified in CSLSI000, they would prevail over those other settings when the procedure is invoked.

## 3.2.3 Operations Manager

OM is the component of the CSL that provides the single point of command entry into an IMSplex and is the focal point for operations management and automation. It uses SCI to communicate with other IMSplex members for command routing and sending a consolidated response to the command originator. The Open Database capability uses the ODBM address space to handle incoming IMS database requests. You can use the type-2 QUERY and UPDATE commands to display ODBM-related information and update its configuration

settings. You must use an OM interface when entering type-2 commands to one or more IMS systems that exist in an IMSplex. Therefore we strongly recommend that you start an OM address space in your IMSplex (again, even if you only have a single IMS system) to enable this capability. Let's now discuss the steps required to implement OM.

## Configuration

Just like the SCI component of the CSL, OM must also be configured before we start its address space. Begin by defining the OM initialization member CSLOIxxx and defining its associated parameter values. We have included a sample OM initialization member in Example 3-6, named CSLOI000.

*Example 3-6 Sample configuration of the OM initialization member*

---

```

*-----*
* OM INITIALIZATION PROCLIB MEMBER - CSLOI000                                *
*-----*
ARMRST=N,                                /* SHOULD ARM RESTART OM ON FAILURE */
CMDLANG=ENU,                            /* USE ENGLISH FOR COMMAND DESC    */
CMDSEC=N,                              /* NO COMMAND SECURITY              */
OMNAME=OM1,                            /* OM NAME (OMID = OM1OM)          */
IMSPLEX(
    NAME=PLEXB,                        /* IMSPLEX NAME (CSLPLEXB)         */
    AUDITLOG=SYSLOG.OM2Q01.LOG),      /* MVS LOG STREAM                  */
CMDTEXTDSN=IMS11B.SDFSDDATA          /* CMD TEXT DATASET                */
*-----*
* END OF MEMBER CSLOI000                                                    *
*-----*

```

---

Let's look at our initialization settings more closely now:

- ▶ **ARMRST=N:** the z/OS Automatic Restart Manager (ARM) is not to restart the OM address space after an abend
- ▶ **CMDLANG=ENU:** the English language is used for IMS command text that is distributed to OM automation clients upon request, which affects only the command descriptions that are displayed on a workstation SPOC that requests command text from OM
- ▶ **CMDSEC=N:** we do not want any security checking to be done when a command is entered to the IMSplex from an OM API; other settings could specify that RACF or an OM security user exit is used for command security
- ▶ **OMNAME=OM1:** the OM address space name OM1 (can be 1-6 characters) is used to define an OMID of OM1OM for use in OM processing
- ▶ **IMSPLEX(NAME=PLEXB):** the IMSplex group name is PLEXB (can be 1-5 characters) and should match the IMSplex group name specified in the SCI, ODBM, IMS initialization and IMS Connect configuration members; note that the characters "CSL" will be attached to the beginning of the IMSplex group name to create the ultimate name of CSLPLEXB
- ▶ **IMSPLEX(AUDITLOG=SYSLOG.OM2Q01.LOG):** the name of our z/OS system logger log stream that OM will write all IMSplex activity to, including command input/output and unsolicited messages (this function is optional and is not required to use the Open Database capability)
- ▶ **CMDTEXTDSN=IMS11B.SDFSDDATA:** the data set name for our PDS containing the command syntax translatable text (can be 1-44 characters and must be a PDS with fixed length record members)

There must be one OM address space per IMSplex. You are not required to have an OM on each z/OS image, as is the case with SCI. We do recommend that you have a backup OM in the IMSplex, to be used in the event that the primary OM experiences a failure. For more information about configuring the OM initialization member, refer to *IMS Version 11 System Definition*, GC19-2444.

**Note:** The IVP application contains a job that defines a sample OM initialization member and adds it to IMS PROCLIB. Refer to “OM, SCI, and DFSCGxxx samples” on page 66 to determine how to find this job within the IVP.

### Starting the OM address space

Use the OM startup procedure to start the OM address space as a task, specifying the xxx suffix of the OM initialization member CSLOIxxx to be used. Refer to Example 3-7 for a sample OM startup procedure.

*Example 3-7 Sample OM startup procedure JCL*

---

```

/*-----*
/*      OM                                     *
/*-----*
/*      PARAMETERS:                         *
/*      BPECFG  - NAME OF BPE MEMBER        *
/*      OMINIT  - SUFFIX FOR YOUR CSLOIxxx MEMBER *
/*      ARMRST  - INDICATES IF ARM SHOULD BE USED *
/*      CMDLANG - LANGUAGE FOR COMMAND DESCRIPTION TEXT *
/*      CMDSEC  - COMMAND SECURITY METHOD      *
/*      OMNAME  - NAME OF THE OM BEING STARTED *
/*-----*
/*
//IEFPROC EXEC PGM=BPEINI00,REGION=3000K,
//  PARM=('BPECFG=BPECFG','BPEINIT=CSLOI000','OMINIT=000',
//      'ARMRST=N','CMDSEC=N','OMNAME=OM1')
/*
//STEPLIB DD DSN=IMS11B.SDFSRESL,DISP=SHR
/*
//PROCLIB DD DSN=IMS11B.PROCLIB,DISP=SHR
/*
//SYSPRINT DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
/*

```

---

Since OM is a CSL address space and it is based on BPE, you must specify the BPE configuration and OM initialization members to be used. For example, we have specified OMINIT=000, so we know that the OM initialization member CSLOI000 will be read when we start this procedure. The ARMRST, CMDSEC, and OMNAME parameter values can all be specified in this procedure to override the definitions included in the CSLOI000 initialization member.

## 3.2.4 Open Database Manager

The Open Database capability uses the Open Database Manager (ODBM) to handle IMS database access requests from distributed applications as well as z/OS applications. It uses

SCI for communication and OM for command processing capability. We'll now review the steps required to define and enable the ODBM component.

## Configuration

One ODBM instance must be defined in the IMSplex to use ODBM functions. Each LPAR containing an IMS control region should have at least one ODBM address space, but is not limited to that single instance. If multiple instances of ODBM are defined in the IMSplex, any of them can fulfill requests received from other z/OS images in the IMSplex. Two ODBM-related PROCLIB members should be defined before starting the ODBM address space: CSLDIxxx and CSLDCxx, which we now review.

### CSLDIxxx

Like the other CSL components, each ODBM has an initialization member named CSLDIxxx which must be defined, similar to our sample member CSLDI001 shown in Example 3-8.

*Example 3-8 Sample ODBM initialization member*

---

```
*****
**  CSLDI001 member:                                *
**  This PROCLIB member is specified by the ODBMINIT=001      *
**  value on the ODBM start up procedure.                    *
**                                                         *
**  Parameters specified here are used for ODBM initialization. *
**                                                         *
**  ODBM configuration parameters are specified in the        *
**  CSLDC001 PROCLIB member which can be specified by either  *
**  the ODBMCFG=001 EXEC parameter or in this PROCLIB member  *
**  on the ODBMCFG=001 parameter.                            *
**                                                         **
*****
ODBMNAME=IMSBBO
IMSPLEX(NAME=PLEXB)
ODBMCFG=001
```

---

Looking more closely at our parameter definitions:

- ▶ ODBMNAME=IMSBBO: the name of our ODBM address space IMSBBO (can be 1-6 characters) will be used to define an ODBMID of IMSBBOOD for use in ODBM processing; note that if you have additional ODBM address space instances in your IMSplex, you must define a separate CSLDIxxx initialization member for each one and ensure that a unique name is used (or designate a unique name on the startup procedure)
- ▶ IMSPLEX(NAME=PLEXB): the IMSplex group name is PLEXB (can be 1-5 characters) and should match the IMSplex group name specified in the SCI, OM, IMS initialization and IMS Connect configuration members; note that the characters “CSL” will be attached to the beginning of the IMSplex group name to create the ultimate name of CSLPLEXB
- ▶ ODBMCFG=001: by specifying a suffix of “001”, we are setting the name of our ODBM configuration member to be CSLDC001, which contains definitions for our Open Database Access (ODBA) connection initialization parameters as well as additional ODBM configuration statements which we explore in the following section

You are also able to specify the ARMRST parameter in the ODBM initialization member, which indicates whether the z/OS Automatic Restart capability should automatically restart the ODBM address space in the event of a failure.



Also not shown in our example is the RRS= parameter, which is used to indicate whether ODBM uses z/OS Resource Recovery Services (RRS). While it is optional to specify this keyword, ODBM runs with RRS by default. This is significant because in this case, both IMS Connect and the IMS control region must also run with RRS.

**Note:** RRS is the z/OS sync point manager and is responsible for coordinating all of the resource managers that are associated with a Unit Of Recovery (UOR). When ODBM runs with RRS, it uses a sync point protocol called two-phase commit (2PC), which ensures that either all or none of an application program's resource updates are made to a set of resources. Refer to *IMS Version 11 Application Programming*, SC19-2428 for more detail about 2PC syncpoint protocol.

ODBM is required to run with RRS when ODBA access is needed, as is the case when IMS data is requested by DB2 stored procedures or from an application running in WebSphere Application Server (WAS) on z/OS. When ODBM runs without RRS, it functions like a Coordinator Controller (CCTL) that connects to IMS data via a database resource adapter (DRA) interface.

Refer to the manual *IMS Version 11 System Definition*, GC19-2444 for details about the CSLDLxxx member.

**Tip:** You can find a sample job that defines an ODBM initialization member and adds it to IMS PROCLIB within the IVP application. Refer to “BPE, ODBM and IMS Connect samples” on page 66 for more detail.

Now that we have defined our ODBM initialization member to be used in starting the ODBM address space, we now must setup the ODBM configuration member, CSLDCxxx. Defining this member will connect ODBM to one or more IMS systems, referred to as data stores in this arena. The ODBMCFG parameter defined in the ODBM CSLDLxxx initialization will point to this configuration member, which we now review.

### **CSLDCxxx**

ODBM must be configured to recognize the IMS data stores that are referenced as alias names by incoming IMS database requests from application programs. The CSLDCxxx member establishes these associations, and contains a global section with settings that apply to all IMS data stores and a local section with settings that apply to specific data stores (which override the global setting if specified). In the previous section, we specified ODBMCFG=001 in our CSLDI001 ODBM initialization member, indicating that CSLDC001 will be used for our ODBM configuration member. We show a sample of this member in Example 3-9.

#### *Example 3-9 Sample ODBM configuration member*

```
*****
** This PROCLIB member is specified by the ODBMCFG=001          **
** value on the ODBM start up procedure or from the            **
** ODBMCFG=001 parameter in the DFSDI001 PROCLIB member        **
**                                                              **
** Parameters specified here are used for ODBM configuration.   **
**                                                              **
** This member is split into 2 sections.                        **
**                                                              **
** <SECTION=GLOBAL_DATASTORE_CONFIGURATION>                    **
** <SECTION=LOCAL_DATASTORE_CONFIGURATION>                     **
**                                                              **
```



```

*****
**  <SECTION=GLOBAL_DATASTORE_CONFIGURATION>          **
**                                                    **
**  This section defines configuration parameters that will be **
**  used when their corresponding parameters are not present in **
**  the local data store section.                        **
**                                                    **
**  Defaults shown                                       **
**  IDRETRY=0 (Can only be specified here.)             **
**  TIMER=60 (Can only be specified here.)             **
**  MAXTHRDS=1                                           **
**  FPBUF=000                                           **
**  FPBOF=000                                           **
**  CNBA=325                                           **
*****
**  <SECTION=LOCAL_DATASTORE_CONFIGURATION>          **
**                                                    **
**  This section defines ODBM configuration parameters.  **
**                                                    **
**  Multiple ODBMs may be appended, each with its unique **
**  ODBMNAME=odbmname. Optional parameters that are not **
**  specified in an ODBM configuration statement group will **
**  take on the corresponding parameter value specified in **
**  the global data store section, if one was specified.  **
**                                                    **
**  ODBM(NAME=odbmname,                                **
**      DATASTORE(NAME=datastorename),                **
**      ALIAS(NAME=aliasname),                          **
**      FPBUF=nnn,                                       **
**      FPBOF=nnn,                                       **
**      CNBA=nnn,                                       **
**      MAXTHRDS=nnn                                   **
**      )                                               **
**                                                    **
**  MAXTHRDS=                                           **
**  FPBUF=                                              **
**  FPBOF=                                              **
**  CNBA=                                              **
*****
<SECTION=LOCAL_DATASTORE_CONFIGURATION>
ODBM(NAME=IMSB0
DATASTORE(NAME=IMSB
ALIAS(NAME=IMS2)
)
)

```

---

The comments at the beginning of the sample indicate which defaults will be used for all ODBM instances if the parameter settings are not specified. It also indicates which sections the parameters can be specified in -- either global, local or both. In our local section, we establish the association between our ODBM instance and a specific IMS control region, or data store. Multiple ODBMs can be defined here and each one's associated data store can have multiple alias names using the ALIAS parameter. Parameters not defined in the local section will default to the values defined in the global section.

As you can see, we have specified our ODBM name to be IMSBBO and created an association with the IMSB control region using the DATASTORE statement. An application program will not know the name of the data store (IMSID, or control region name) and can only reference it as an alias name. The IMSB data store can be referenced as IMS2 by incoming application program requests for IMS data, since we have specified an alias name of IMS2 using the ALIAS statement. Note that the alias name can be different than the data store name. Specify these settings in the local section of the configuration member, the start of which is designated by the <SECTION=LOCAL\_DATASTORE\_CONFIGURATION> header.

**Note:** The ALIAS parameter is optional. If one is not specified in this section, an application can reference the data store name and IMS will internally create an ALIAS name using the data store name.

Next, there are certain parameters that you can only specify in the global section, which begins with the <SECTION=GLOBAL\_DATASTORE\_CONFIGURATION> header. Specifically, you can use the IDRETRY parameter to specify the number of attempts an ODBM instance should make to connect to a data store when it does not connect right away, and use the TIMER parameter to indicate how many seconds should elapse before the connection is attempted again. As you can see, we have taken the defaults in our sample.

You can define the remaining parameters in the configuration member in either the local or the global section. As a reminder, any parameter defined in the local section will override any definitions in the global section. In our sample, we have only defined the CNBA global parameter (specifies the total number of Fast Path NBA buffers for ODBM use), whereas we have taken the default setting for each of the others. Use the MAXTHRDS parameter to indicate how many concurrently active threads an IMS data store can have, and the FPBUF and FPBOF parameters to show how many Fast Path DEDB buffers and Fast Path DEDB overflow buffers are allocated per thread, respectively. You can find more information about how ODBM configuration parameters are defined in CSLDCxxx in the manual *IMS Version 11 System Definition*, GC19-2444.

**Tip:** You can find a sample job that defines an ODBM configuration member and adds it to IMS PROCLIB within the IVP application. Refer to “BPE, ODBM and IMS Connect samples” on page 66 for more detail.

## Starting the ODBM address space

Use the ODBM startup procedure to start the ODBM address space as a task, specifying the BPE configuration member and suffix of the ODBM initialization member to be used. Keep in mind that startup parameters specified here in the procedure will override those predefined in our CSLDI001 ODBM initialization member. Refer to Example 3-10 for a sample of the ODBM startup procedure that we used.

*Example 3-10 Sample ODBM startup procedure JCL*

---

```
//*****
//*      ODBM Procedure
//*
//*
//*      Parameters:
//*      BPECFG  - Name of BPE member
//*      BPEINIT - CSLDINI0, the module that contains the ODBM start up values
//*      ODBMINIT - Suffix for your CSLDIxxx member
//*      PARM1   - other override parameters:
```

```

//*          ARMRST - Indicates if ARM should be used
//*          ODBMNAME - Name of ODBM being started
//*          ODBMCFG - Suffix for your CSLDCxxx member
//*          RRS      - Indicates RRS is (Y) or is not (N) used
//*
//*          example:
//*          PARM1='ARMRST=Y,ODBMNAME=ODBM1,ODBMCFG=000,RRS=N'
//*
//IMSBOPD  PROC RGN=0M,TME=1440,SOUT=*,
//          BPECFG=BPEODBM,
//          ODBMINIT=001,
//          CSLDI=CSLDINI0,
//          PARM1='RRS=Y,ARMRST=N'
//*
//ODBMPROC EXEC PGM=BPEINI00,REGION=&RGN,TIME=&TME,
//  PARM='BPECFG=&BPECFG,BPEINIT=&CSLDI,ODBMINIT=&ODBMINIT,&PARM1'
//STEPLIB DD DSN=IMS11B.SDFSRESL,DISP=SHR
//          DD DSN=SYS1.CSSLIB,UNIT=SYSALLDA,DISP=SHR
//PROCLIB DD DSN=IMS11B.PROCLIB,DISP=SHR
//SYSPRINT DD SYSOUT=&SOUT
//SYSUDUMP DD SYSOUT=&SOUT

```

---

We have designated BPECFG=BPEODBM, so we know that the BPEODBM member defined in Example 3-2 on page 44 will be used in starting the ODBM address space. We have also specified ODBMINIT=001, so the ODBM initialization member CSLDI001 shown in Example 3-8 on page 49 will be read when we start this procedure. The BPE initialization member that will be used is CSLDINI0 (not shown in this book). The ARMNST and RRS parameter values can both be included in this procedure to override the definitions contained in the CSLDI001 initialization member.

### ODBM user exits

You are able to customize and monitor your ODBM environment by writing one or more of the following ODBM user exits, called and managed by BPE:

- ▶ CSL ODBM Initialization and Termination user exit: called when an ODBM or IMSplex initializes or terminates; can be used to track the timing of these events
- ▶ CSL ODBM Input user exit routine: called when a CSLDMI request (from the ODBM API) is issued; can change segment search arguments (SSAs), an I/O area, or an application interface block (AIB)
- ▶ CSL ODBM Output user exit routine: called when an ODBM is returning a response to an ODBM client that sent in a request; can view and modify response data before sending it to client
- ▶ CSL ODBM Client Connect and Disconnect user exit routine: called when an ODBM client registers to or de-registers from ODBM
- ▶ CSL ODBM statistics available through BPE statistics user exit: tracks statistics about both BPE and ODBM

For additional details about any of these exits, refer to the manual *IMS Version 11 Exit Routines*, SC19-2437.

### ODBM commands

You can issue type-2 commands to display ODBM information with the QUERY ODBM command and update the configuration settings with the UPDATE ODBM command. These

are type-2 commands, so you must enter them through an OM interface such as the IMS TSO SPOC application (also referred to as just “TSO SPOC”), which we will show several examples of shortly. Using this application, you are able to send these commands to the entire IMSplex. To invoke the TSO SPOC, select option 1 from the IMS application menu, shown in Figure <\$paran<\$paranum64.

**Note:** Before you can issue IMS commands from the TSO SPOC application, you must set your desired user preferences from the OPTIONS menu.

## QUERY ODBM

You can display information about ODBM in several different aspects, including:

- Alias names that are associated with all ODBM instances in the IMSplex. Refer to Figure 3-1 for sample output from the `QUERY ODBM TYPE(ALIAS)` command, which shows that two alias names `IMS2` and `IMSZ` are associated with our ODBM instance.

[illegible]

Figure 3-1 Sample output from `QUERY ODBM TYPE(ALIAS) SHOW(ALL)` command

- Configuration information for all ODBM instances drawn from each one's CSLDCxxx configuration member. See Figure 3-2 for sample output from the QUERY ODBM TYPE(CONFIG) command, which shows the parameter definitions. Scrolling to the right displays two more columns named Timer and Aliases, not shown here. For this command, the output is sorted by configuration member name.

[illegible]

Figure 3-2 Sample output from `QUERY ODBM TYPE(CONFIG) SHOW(ALL)` command

- Configuration information for ODBM drawn from its CSLDCxxx configuration member, displayed by data store. See Figure 3-3 for sample output from the QUERY ODBM TYPE(DATASTORE) command, which shows how we defined the parameter values. Scrolling to the right displays two more columns: the CNBA, MaxThrds, IdRetry, and Timer values, not shown here. For this command, the output is sorted by data store name.

[illegible]

Figure 3-3 Sample output from `QUERY ODBM TYPE(DATASTORE) SHOW(ALL)` command

- Clients of ODBM, as shown in the sample output for the `QUERY ODBM TYPE(SCIMEMBER)` command in Figure 3-4. Note that the command response shows one ODBM client. IMS Connect.

```

File Action Manage resources SPOC View Options Help
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
PLEXB                      IMS Single Point of Control
Command ==>

    sssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
    Response for: QUERY ODBM TYPE(SCIMEMBER) SHOW(ALL)
MbrName SCIMbrName  CC Type      ThreadCount
IMSBB00D IMSBBHWS   0 IMSCON      0

F1=Help      F3=Exit      F4=Showlog    F6=Expand    F9=Retrieve  F12=Cancel

```

Figure 3-4 Sample output from `QUERY ODBM TYPE(SCIMEMBER) SHOW(ALL)` command

- Individual ODBM threads. You can filter the command response to show only those threads that are associated with certain specified PSBs, ODBM clients such as IMS Connect, data stores, and threads that have a specific status. Refer to the sample response to the QUERY ODBM TYPE(THREAD) command in Figure 3-5 and note that we have not included a filter for the command. Instead we chose to display the IMS Connect name and PSB associated with the ODBM thread currently running and have scrolled to the right to display the information of interest.

[illegible]

Figure 3-5 Output from `QUERY ODBM TYPE(THREAD) SHOW(PSB SCIMEMBER)` command

- Trace status, which you can see an example of in the response to the `QUERY ODBM TYPE(TRACE)` command shown in Figure 3-6. As you can see, we currently do not have an active trace running for our ODBM instance.

```
File Action Manage resources SPOC View Options Help
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
PLEXB                               IMS Single Point of Control
Command ==>

    sssssssssssssssssssss Plex . .      Route . .          Wait . .
Response for: QUERY ODBM TYPE(TRACE) SHOW(ALL)
MbrName   DatastoreName   CC TraceStatus
IMSBBOOD  IMSB           0 INACTIVE

F1=Help       F3=Exit     F4>Showlog    F6=Expand    F9=Retrieve  F12=Cancel
```

Figure 3-6 Sample output from `QUERY ODBM TYPE(TRACE) SHOW(ALL)` command

## UPDATE ODBM

You can dynamically change ODBM configuration settings with the type-2 UPDATE ODBM command. You can also use this command to start and stop data stores, aliases, and tracing activity. We now discuss each type of UPDATE ODBM command in detail:

- If you need to dynamically stop a connection between ODBM and IMS, you can do so with the `UPDATE ODBM STOP(CONNECTION)` command. On this command, you can specify one or more data store or alias names. See Figure 3-7 for a sample. When the connection is broken, you will also see a message issued to the system console similar to the following: `CSL4009I ODBM Disconnected from IMS data store IMSB IMSBBOOD`.

```

File Action Manage resources SP0C View Options Help
ssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssssss
PLEXB                      IMS Single Point of Control                Keylist saved
Command ==>

    sssssssssssssssssssssssss    Plex . .          Route . .          Wait . .
Response for: UPDATE ODBM STOP(CONNECTION) DATASTORE(IMSB)
MbrName  DatastoreName  CC
IMSB000  IMSB          0

F1=Help      F3=Exit      F4=Showlog    F6=Expand    F9=Swap      F12=Cancel

```

Figure 3-7 Sample response for UPDATE ODBM STOP(CONNECTION) DATASTORE(IMSB)

You can dynamically start the connection between ODBM and IMS again with the UPDATE ODBM START(CONNECTION) command. See Figure <\$paran<\$paranum59 for a sample. When the connection is established, you will also see a message issued to the system console similar to the following: CSL4004I ODBM Connected to IMS data store IMSB IMSBBOOD.

**Note:** You can start and stop connections between ODBM and a data store, or between ODBM and an alias. These two are mutually exclusive and it is important to understand that when you start or stop a connection between ODBM and alias, it does not impact ODBM's connection to the actual data store. Therefore, it is possible to stop an ODBM connection to an alias, but the ODBM will still be connected to the associated data store afterwards. The ability to start and stop a connection to an alias is useful in that it allows you to isolate programs using a specific alias, while other aliases still connected to ODBM may still access the underlying data store.



[illegible]

Figure 3-8 Sample response for UPDATE ODBM START(CONNECTION) DATASTORE(IMSB)

The UPDATE ODBM commands allow you also to dynamically change the ODBM's configuration. To update the configuration, use the commands in the following sequence:

```
UPDATE ODBM STOP(CONNECTION) DATASTORE(*)
UPDATE ODBM TYPE(CONFIG) MEMBER(xxx)
UPDATE ODBM START(CONNECTION) DATASTORE(*)
```

This sequence reads and activates PROCLIB member CSLDCxxx. If you change the suffix xxx, you must also remember to change member CSLDIxxx or the ODBM task JCL, otherwise ODBM will revert to the old settings when it is next started.

**Note:** As you can infer from the command sequence example above, ODBM requires the existing data stores to be stopped before the configuration can be changed.

You can find more command examples in the manual *IMS Version 11 Commands, Volume 2: IMS Commands N-V*, SC19-2431.

### 3.3 IMS Connect

IMS Connect uses ODBM to enable distributed DRDA application access to IMS data. You may already be using IMS Connect for TCP/IP access to IMS transactions and/or commands.

But if you have not yet implemented IMS Connect, you must do so in order to use the Open Database capability. This section examines both scenarios.

**Important:** The user ID associated with a request for IMS data from a distributed application should be authenticated in IMS Connect. Refer to 3.5, “Security considerations” on page 71 for recommendations and further detail about how to accomplish this.

### 3.3.1 First-time implementation: setup and configuration

If you are already using IMS Connect, go to “IMS Connect user exits” on page 62. Otherwise, begin by defining two configuration members: one for BPE and one for IMS Connect. Once these members are in place, you can then start the IMS Connect address space as a task. We now discuss each of these steps in more detail, beginning with the BPE configuration member.

#### BPECFxxx

First, we define our IMS Connect BPE execution environment settings in the BPECFxxx member. This member contains IMS Connect message language information and tracing specifications for IMS Connect’s internal trace tables. Refer to Example 3-3 on page 44 for a sample of the member named BPECFGIV that we will use for our initializing our IMS Connect address space.

#### HWSCFxxx

Next, you must define your IMS configuration member to designate IMS Connect’s environment settings. This establishes how IMS Connect will communicate with TCP/IP, the CSL and ODBM, among other components. You must define several parameters in this member, which are very thoroughly documented in the manual *IMS Version 11 System Definition*, GC19-2444 as well as in the book *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity*, SG24-6794. Let’s now take a look at the way we have defined our IMS Connect configuration member HWSCFODB in Example 3-11.

*Example 3-11 Sample IMS Connect configuration member*

---

```
*****
* CONFIGURATION FILE FOR IMS CONNECT (HWSCFODB)                                *
*****

HWS=(ID=IMSBHWS,XIBAREA=100,RACF=N,RRS=Y)
TCP/IP=(HOSTNAME=TCP/IP)
ODACCESS=(ODBAUTOCONN=Y,IMSPLEX=(MEMBER=IMSBHWS,TMEMBER=PLEXB),
          DRDAPORT=(ID=5555,PORTTMOT=6000),ODBMTMOT=6000)
```

---

We have defined several parameters here:

- ▶ HWS=
  - (ID=IMSBHWS): The name of our IMS Connect address space is IMSBBHWS.
  - (XIBAREA=100): We allow 100 full words to be allocated for the XIB user area.
  - (RACF=N): The user ID is passed to IMS without calling RACF for authentication.

**Note:** If no user ID is passed by the calling application, IMS Connect will not set a default user ID. Instead, the address of the ODBM address space will be used for any security checks that later occur in IMS. See Chapter 3.5, “Security considerations” on page 71 for further detail about security methods used with the Open Database capability.

- (RRS=Y): Two-phase commit is enabled by Resource Recovery Services (RRS).
- TCPIP=
  - (HOSTNAME=TCPIP): The jobname of our TCP/IP host z/OS address space is TCPIP.
- ODACCESS=
  - (ODBAUTOCONN=Y): When our IMS Connect initializes, it will automatically register with all current and future ODBM instances. We can later disable this function by issuing the IMS Connect command SETOAUTO.
  - IMSPLEX=(MEMBER=IMSBHWS,TMEMBER=PLEXB): The name of our IMS Connect instance within the IMSplex is IMSBHWS. The IMSplex name is PLEXB and must be the same as the IMSPLEX name specified in the SCI initialization member, which you can see in Example 3-4 on page 45. If you are using OTMA for IMS Transaction Manager access and have specified the name of the IMS data store using the ID parameter on the DATASTORE statement, keep in mind that it must be different than the value you define for the T MEMBER on this IMSPLEX statement.
  - (DRDAPORT=(ID=5555,PORTTMOT=6000)): The port number IMS Connect will receive incoming requests for IMS data on is 5555 and must be unique among all other port specifications. Our IMS Connect instance will wait 6000 hundredths of seconds for the next input message (after having already received the initial message) from a client application connected on a DRDA port. After this time has elapsed, it will disconnect the client. This is useful because it prevents IMS Connect from unnecessarily waiting for requests when a client is looping or hung. Not shown in our example is the KEEPAV parameter, which defines the frequency interval at which a packet is sent by the z/OS TCP/IP layer to port 5555, thus maintaining a connection when it is otherwise idle. Since we did not specify this parameter on the DRDAPORT statement, it will be set to 0, meaning that we will use the KeepAlive value defined in the z/OS TCP/IP stack.
  - (ODBMTMOT=6000): Our IMS Connect instance will wait 6000 hundredths of seconds for each response message from ODBM and also for the initial message after a client application establishes a socket connection with it. When ODBM does not respond within 60 seconds, a message HWSJ2530W is sent to the client but the socket connection is retained. On the other hand, if the client application does not send data to IMS Connect after making the initial socket connection within 60 seconds, then the connection is terminated.

**Note:** If you already have IMS Connect set up for use with other IMS capabilities such as Open Transaction Manager Access (OTMA) or the IMS Control Center, you simply need to add the ODACCESS statement to your existing HWSCFGxx configuration member as just detailed in this section.

For additional detail about defining parameters in the *HWSCFGxx* member, refer to the manual *IMS Version 11 System Definition*, GC19-2444. If this is your first time setting up IMS Connect, we recommend reviewing the book *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity*, SG24-6794, as it has extensive detail about IMS Connect functionality and additional setup recommendations. We are now ready to start our IMS Connect address space, which we now explore.

## Starting the IMS Connect address space

Use the IMS Connect startup procedure to start the IMS Connect address space as a task, specifying the BPE configuration member and suffix of the IMS Connect configuration member to be used. Refer to Example 3-12 for a sample IMS Connect startup procedure.

*Example 3-12 Sample startup procedure for the IMS Connect address space*

---

```
//IMSBHWS PROC RGN=OM,TME=1440,SOUT=S,
//          BPECFG=BPECFGIV,HWSCFG=HWSCFODB
//*
//* FUNCTION: START IMS CONNECT REGION.
//*
//HWSREGN EXEC PGM=HWSHWS00,REGION=&RGN,TIME=&TME,
//          PARM='BPECFG=&BPECFG,HWSCFG=&HWSCFG'
//STEPLIB DD DSN=IMS11B.SDFSRESL,DISP=SHR
//*        DD DSN=CEE.SCEERUN,UNIT=SYSALLDA,DISP=SHR
//*        DD DSN=SYS1.CSSLIB,UNIT=SYSALLDA,DISP=SHR
//PROCLIB DD DSN=IMS11B.PROCLIB,DISP=SHR
//SYSPRINT DD SYSOUT=&SOUT
//SYSUDUMP DD SYSOUT=&SOUT
//HWSRCORD DD DSN=IMS11B.HWSRCRD,DISP=SHR
//*
```

---

As you can see, we have defined our BPE configuration member with BPECFG=BPECFGIV. This member is shown in Example 3-3 on page 44 and contains the tracing definitions that will be used for this IMS Connect address space instance. We have also specified HWSCFG=HWSCFODB, which is the IMS Connect configuration member shown in Example 3-11 on page 60 and will be read when we run this procedure.

**Tip:** Make sure that your IMS Connect load library is APF-authorized and allow your IMS Connect to run in authorized supervisor state (key 7) by updating the program properties table (PPT) in SYS1.PARMLIB.

## IMS Connect user exits

You can use several user exits with IMS Connect. With IMS Open Database, you can use the HWSROUT0 user exit to route an IMS data request to a specific IMS. Either the calling DRDA application or the IMS Universal Driver can specify this IMS, overriding the IMS alias that would have otherwise been selected. You can also use the exit routine to route requests to a specific ODBM instance during processing. For detail about this and other IMS Connect user exits, refer to *IMS Version 11 Exit Routines*, SC19-2437.

**Tip:** As a final step in setting up IMS Connect, install the default user exits into the IMS Connect resident library.

## IMS Connect considerations for IMSplex

When defining an IMS Connect instance in an IMSplex environment, there are a few things to keep in mind:

- ▶ You are not limited to defining an IMS Connect instance to a single IMSplex. You can include multiple IMSPLEX statements within the HWSCFGxx configuration member to define a single IMS Connect instance to more than one IMSplex.

- ▶ An IMS Connect instance can register with an IMSplex with only one name. As previously stated, you can define an IMS Connect to multiple IMSplexes using the same name, but ensure that each statement designates a different IMSplex.
- ▶ Defining an SCI instance within the IMSplex enables communication between IMS Connect and the ODBM address space either within the same LPAR or cross-LPAR. When they reside on the same LPAR, SCI uses the z/OS Program Call (PC) function to enable communication, but on separate LPARs SCI uses the cross-system coupling facility (XCF).

### 3.3.2 Modifying existing IMS Connect definitions for IMS Open DB support

You may already be using IMS Connect for TCP/IP access to IMS transactions and/or commands. In this case you already have IMS Connect installed. To use the IMS Open Database capability, you simply need to modify your IMS Connect configuration member `HWSCFGxx` to include the `ODACCESS` statement as described in “`HWSCFxxx`” on page 60 and shown in Example 3-11 on page 60. Once you have updated the configuration member, start IMS Connect as you usually do, using JCL similar to Example 3-12 on page 62.

## 3.4 Using IMS applications to help set up CSL and IMS Connect

The IMS product comes with two applications that can assist you in defining the PROCLIB members associated with the different IMS Open Database components that we have just covered. In this section, we discuss the Installation Verification Program and the Syntax Checker applications.

### 3.4.1 Installation Verification Program

IMS provides a program that you can use to install a sample IMS system and try out different IMS functions. This program is called the IMS Installation Verification Program (IVP) and you can use it as a reference when setting up the required PROCLIB members for the CSL and IMS Connect address spaces. In it, you will find sample JCL that starts each of these address spaces.

**Note:** While the IVP starts the CSL and IMS Connect address spaces as jobs, you can also start them as started tasks.

The IVP also includes a sample application that exercises the IMS Open Database capability, which you can test if you would like to use it as more than just a reference for defining IMS Open Database-related PROCLIB members and procedures.

To begin, invoke the IVP using option 6 from the IMS Application menu shown in Figure 3-9.

[illegible]

*Figure 3-9 The IMS Application Menu*

Follow the prompts until you reach Execution mode, which lists a series of jobs and tasks that start the sample IMS system and test different functions. In our case, we selected a Database and Transaction Management (DB/DC) environment as well as the IMS Connect and Open Database Sample sub-options, shown below in Figure 3-10.

[illegible]

Figure 3-10 Selecting the sub-options of IMS Connect and the Open Database Sample in the IVP

## Sample PROCLIB member definitions

Using the IVP, you can view sample PROCLIB member definitions for all of the components associated with the IMS Open Database capability. Scroll down to the *IV3E* series of jobs/tasks within the Execution panel and look for the IV3E302J and IV3E303J jobs, shown in Figure 3-11.

[illegible]

Figure 3-11 The IV3E302J and IV3E303J jobs show examples adding required PROCLIB members

### ***BPE, ODBM and IMS Connect samples***

If you browse the IV3E302J job you will see several examples of the IVP defining members to IMS PROCLIB:

- ▶ HWSCFODB specifies IMS Connect configuration statements.
- ▶ CSLDI001 and CSLDC001 define ODBM initialization and configuration statements, respectively.
- ▶ BPEODBM is the sample BPE configuration member which is specified later in the IVP's sample IMS Open Database application when starting the ODBM address space.

### ***OM, SCI, and DFSCGxxx samples***

Next, browse the IV3E303J job to see sample PROCLIB definitions for the SCI and OM initialization members: CSLOI000 and CSLSI000, respectively. Within this job, you will also find two examples of how to define a DFSCGxxx PROCLIB member, which is important in designating the IMSplex name that all CSL components will be a part of.

**Attention:** As of IMS V10, you can specify your CSL definitions in the CSL section of the DFSDFxxx PROCLIB member. If the CSL definitions are present in both DFSDFxxx and DFSCGxxx members, those contained in DFSCGxxx will take precedence.

Now that we have seen examples of how the Open Database components' initialization and configuration members are defined to IMS PROCLIB, we will now review how to find sample jobs that start the associated address spaces within the IVP application.



## Sample startup JCL

You can find the jobs that start SCI, OM, ODBM and IMS Connect in the *IV3T* series of IVP jobs shown in Figure 3-12.

[illegible]

*Figure 3-12 The IV3T series of the IVP contains jobs that start IMS Open Database components*

For more information about how to use the IVP refer to the manual entitled *IMS Version 11 Installation*, GC19-2438.

### 3.4.2 IMS Syntax Checker

You can also use the IMS Syntax Checker to assist you in defining, verifying and validating the parameters contained in your PROCLIB members. In this section we use the IMS Connect configuration member HWSCFODB as an example and show how to use Syntax Checker to validate the settings contained in it.

First, invoke the IMS Application Menu and select option 5 for the IMS Syntax Checker shown in Figure 3-9.

On the next panel, enter the name of the data set that contains the member definitions you would like to validate with the Syntax Checker, as shown in Figure 3-13. Our members are contained within the IMS11B.PROCLIB data set, so we have specified it here with single quotes.

[illegible]

*Figure 3-13 Specifying the data set name containing our member definitions to be validated*

The next panel displays a member list, from which you can select a member to review, as shown in Figure 3-14. In our example, we would like to check the syntax of our IMS Connect configuration member, `HWSCFODB`.

[illegible]

Figure 3-14 Selecting the IMS Connect configuration member for review within Syntax Checker

The next panel will prompt you to indicate whether this member is a BPE exit list member or an IMS Connect configuration member; select the latter as shown in Figure 3-15.

[illegible]

Figure 3-15 The Syntax Checker requesting input regarding the member type

Then select IMS 11.1 from the next menu as shown in Figure 3-16.

```
File Help  

SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS  

DFSSCSRT      IMS Parameter Syntax Checker  

Command ==>  
  

Enter the following information and press enter.  
  

IMS Release . . . . . 1   1. IMS 11.1  

                       2. IMS 10.1  

                       3. IMS 9.1
```

F1=Help    F3=Exit    F12=Cancel

Figure 3-16 Syntax Checker prompting for IMS version level

Next, the Syntax Checker displays a list of the values you can define within the member along with a brief explanation of each parameter's function. We have scrolled down to the ODACCESS section of the HWSCFODB member in our sample shown in Figure 3-17. You can alter the display by choosing to collapse or expand certain sections.

[illegible]

Figure 3-17 Syntax Checker view of our IMS Connect configuration member, HWSCFODB

One very useful feature within Syntax Checker is the F1=Help function. If you place your cursor on a field and press F1, it displays detailed information about the parameter meaning, as well as what values are valid. Refer to Figure 3-18 for an example of the help panel for the PORTTMOT parameter.

```
File Help
DFSSCSRT      HELP FOR PORTTMOT
Command ==>

Keyword: PORTTMOT

PORTTMOT=
    Defines the amount of time that IMS Connect waits for the next input message from a client application that is connected on a DRDA port before IMS Connect disconnects the client.
```

The timeout interval is specified as a decimal integer in hundredths of a second. Valid values for PORTTMOT are from 0 to 2 147 483 647 (X'FFFFFFFF'). The default is 18 000 (3 minutes). A 0 disables the timeout function.

Specifying a timeout value can avoid hang conditions when a client stops sending messages as expected due to, for example:

- \* A looping client application

```
F1=Help       F3=Exit        F7=Backward   F8=Forward    F12=Cancel
```

Figure 3-18 Help panel for the PORTTMOT parameter of the ODACCESS statement

### 3.5 Security considerations

When a data request comes into IMS through ODBM from either a DRDA application or one of the IMS Universal Drivers, it must go through security checking before the data can be retrieved. This process is two-fold. First, use IMS Connect security to authenticate the user before the request reaches ODBM. This authentication confirms that users really are who they claim they are. Second, determine whether the user is authorized to access the IMS resource using either allocate PSB (APSB) security or Resource Access Security (RAS). ODBM will be able to retrieve the requested data from IMS only if these security checks are passed.

We now explore how to use IMS Connect to authenticate a user ID, then how to authorize this user to allocate a PSB or access an IMS resource using the APSB and RAS facets of IMS security.

## Authenticating a user ID using IMS Connect security

IMS Connect security gives you two options to authenticate users who are requesting IMS data: using the Security Authorization Facility (SAF) interface or the IMS Connect DB Security user exit routine, HWSAUTH0.

## Using the Security Authorization Facility

IMS Connect can call a SAF interface such as RACF to authenticate a user. To activate this option, define RACF=Y in the HWSCFGxx configuration member (this setting can later be changed using the IMS Connect SETRACF command). When an application sends in a

request for IMS data, it will include a user ID (if it is not being provided by an exit) and either a password or a RACF PassTicket.

**Note:** A RACF PassTicket is a one-time-only password that is generated by the calling application or a user exit, and is an alternative to the RACF password. This is considered to be more secure than using the RACF password because it removes the need to send the password across the network in clear text.

IMS Connect will then perform authentication with a call to RACF. When the request reaches ODBM, the user will have already been authenticated and will only then need to be authorized to access the APSB or the particular IMS resource.

### ***IMS Connect DB Security user exit (HWSAUTH0)***

Instead of SAF, you can use the IMS Connect DB Security user exit routine (HWSAUTH0) to perform the necessary user ID security checking when a message is received from an application program. This refreshable exit is shipped with IMS Connect and link-edited into the IMS Connect RESLIB, and will always be called even if RACF=Y has been specified and RACF is called afterwards.

It can authenticate a user ID that has been passed in by an application, and can even designate a different user ID to replace the original before it is sent to ODBM. The exit can also provide a RACF group ID to be authenticated further by IMS Connect.

HWSAUTH0 is a standard BPE type-1 exit routine and you can manage it using the BPE DISPLAY USEREXIT and REFRESH USEREXIT commands. To use the exit, you must first create or modify your current BPE exit list PROCLIB member using a name of your choice -- we have used HWSICNX0. Also in the member, define HWSAUTH0 as an exit similar to what we show in Example 3-13. Note that the example contains all values exactly as you should code them, but you can specify a number of your choice for the ABLIM= parameter, which tells IMS how many times the exit can abend before it becomes disabled.

*Example 3-13 Defining the HWSAUTH0 user exit within the BPE exit list PROCLIB member*

---

```
EXITDEF (TYPE=ODBAUTH, EXITS=(HWSAUTH0), ABLIM=8, COMP=HWS)
```

---

Finally, add an EXITMBR statement to your BPE configuration member, specifying the name of your new exit, similar to our sample shown in Example 3-14. Note that we have designated that our new exit, HWSICNX0, be used.

*Example 3-14 Adding a user exit to the BPE configuration member with the EXITMBR statement*

---

```
EXITMBR=(HWSICNX0,HWS)      /* IMS CONNECT EXITS */
```

---

### **Authorizing a user ID to access a resource using IMS security**

Once a user ID has been authenticated by IMS Connect, the request for IMS data flows to ODBM, which in turn passes the request to the appropriate IMS system. At this point, we must determine whether the user ID is authorized to allocate the PSB or access another IMS resource. The path we take here depends on what value we have specified for our ODBASE= execution parameter in our DFSPBxxx PROCLIB member. If we have specified ODBASE=Y and our ODBM is running with RRS, we will use APSB security. Otherwise if we specified ODBASE=N (the default) or our ODBM is not running with RRS, we will use RAS security.

**Attention:** If you choose to track statistics for a RACF user ID, keep in mind that performance could be affected.

Let's now look at how each of these security types function.

### ***Allocate PSB (APSB) security***

To protect APSBs from unauthorized access, protect them in the AIMS RACF class and grant the user access to each protected resource. See Example 3-15 for a simple model of what these RACF definitions might look like.

*Example 3-15 Sample RACF definitions for authorizing a user to access a protected APSB*

---

```
ADDUSER APPLUID1
RDEFINE AIMS APSBX UACC(NONE)
PERMIT APSBX CLASS(AIMS) ID(APPLUID1) ACCESS (READ)
```

---

In our example, the user ID of our calling application is APPLUID1. We protect an APSB named APSBX in the AIMS RACF class with an RDEFINE statement, then we grant our APPLUID1 user access to this APSB with a PERMIT statement. Before the PSB can be dynamically allocated, the application's user ID will first be authorized against the APSB.

**Restriction:** You can only use APSB security if ODBM is running with Resource Recovery Services (RRS), which is the default setting.

### ***Resource access security (RAS)***

As we previously mentioned, RAS is used if you specified ODBASE=N in either your IMS startup procedure or your DFSPBxxx PROCLIB member. It will also be used if your ODBM is running without RRS. RAS determines whether the user passed to ODBM is authorized to access a requested PSB. You can specify the type of security that RAS will use with the ISIS parameter, also defined in either the IMS startup procedure or the DFSPBxxx member. Here are the possible values that you can define for the ISIS parameter:

- ▶ ISIS=N disables RAS security
- ▶ ISIS=R will use RACF for RAS security
- ▶ ISIS=C will use the DFSRAS00 user exit for RAS security
- ▶ ISIS=A will use both RACF and the DFSRAS00 user exit for RAS security

**Note:** In previous IMS releases, you were able to specify ISIS=0, ISIS=1, or ISIS=2 which were related to SMU AGN security. After IMS V9, SMU security is no longer supported and if you specify any of these no longer supported values, they are internally translated to ISIS=N and RAS is disabled.

You can also activate RAS with the SECURITY macro's TYPE parameter. Here are the possible values:

- ▶ NORAS disables RAS security
- ▶ RASRACF will use RACF for RAS security
- ▶ RASEXIT will use the DFSRAS00 user exit for RAS security
- ▶ RAS will use both RACF and the DFSRAS00 user exit for RAS security

**Important:** The ISIS parameter always overrides any RAS-related specification on the SECURITY macro. To use the SECURITY macro's designation, you must omit the ISIS parameter entirely.

For more information about the SECURITY macro, refer to *IMS Version 11 System Definition*, GC19-2444.

When you use RAS with RACF, the user ID is authorized to access a PSB protected in the RACF class IIMS (or the JIMS RACF grouping class for PSBs). The user ID passed to ODBM can vary depending on its source, which can be from any of the following:

- ▶ IMS Connect, the user ID is passed with the request, and is typically that of the application
- ▶ An ODBA application such as WebSphere Application Server (WAS) or DB2 stored procedure, the user ID is that of the user
- ▶ A batch ODBA application, the user ID is the batch job's user ID
- ▶ If no user ID was passed on the ODBM request, the user ID is the ODBM address space's user ID, specified on the USER= parameter of the ODBM address space's startup JCL

If you specify that both RACF and the exit are used, the exit will always be called after RACF is called. For more information about how to use the DFSRAS00 user exit routine, refer to the manual entitled *IMS Version 11 Exit Routines*, SC19-2437.





## Generating IMS metadata class with IMS Enterprise Suite DLIModel Utility

In this chapter, we explain the functionalities that the IMS Enterprise Suite offers for Open Database support. The IMS Enterprise Suite DLIModel Utility is required to generate the IMS database metadata file.

This chapter is intended to be read by the person responsible for generating the IMS metadata files. Depending on how the tasks are assigned in your organization, this can be the IMS system programmer, the IMS database administrator or the application developer which has some knowledge of IMS and has access to the necessary resources.

Several SOA technologies (both existing and as improved by IMS 11) are now packaged in the *IMS Enterprise Suite* product, made available with IMS 11. IMS Enterprise Suite V1.1 (program numbers 5665-T60 and 5665-T61) contains the following items:

- ▶ IMS SOAP Gateway
- ▶ IMS Connect API for Java and C
- ▶ JMS API
- ▶ DLIModel Utility

In this chapter we concentrate on the new IMS Enterprise Suite DLIModel utility's functions, which we will cover in the following order:

- ▶ Introduction
- ▶ Overview of IMS Enterprise Suite DLIModel utility
- ▶ Download and installation
- ▶ Setup for sample scenarios included in this book
- ▶ Using the IMS Enterprise Suite DLIModel utility
  - Generating metadata for Car Dealer database
  - Editing the AUTPSB11 Project
  - Export the metadata as Jar file
- ▶ Additional considerations for the IMS Enterprise Suite DLI Model Utility

## 4.1 Introduction

The IMS Enterprise Suite consists of components that are designed to support open integration technologies to enable new application development and extend the access to IMS transactions and data. Here, we focus on a single component of the IMS Enterprise Suite: the IMS Enterprise Suite DLIModel utility plug-in (also referred to as the DLIModel utility). The plug-in is based on Eclipse and can therefore be integrated into an existing Eclipse development environment.

Once installed, it can take existing IMS source files such as COBOL copybooks and PL/I structures and convert them into metadata, which describes the layout and contents of these source files. You can then use this metadata to write Java applications in Eclipse, IBM Rational Application Developer for WebSphere, or IBM Rational Developer for System z with no requirement to create or work with z/OS control statements.

In this chapter, we show how to download, install and use the DLIModel utility for this purpose. First, let's discuss a more detailed overview of the utility, including its requirements, restrictions and a brief history of its evolution.

## 4.2 Overview of IMS Enterprise Suite DLIModel utility

IMS data is traditionally stored in mainframe-based IMS source files such as PSBs (program specification blocks) and DBDs (database descriptions). To use this data in a modern Java application, it must first be transformed into a format that can be understood by the application. As of IMS V11, IMS does not have a catalog that contains these transformed definitions. Rather, the DLIModel utility is the mechanism that performs this transformation. It takes IMS source files such as PSB libraries, DBD libraries and COBOL and PL/I copybooks as input and generates metadata, which describes the IMS data. More specifically, it generates a Java class called the IMS Java metadata class, which is a subclass of the `com.ibm.ims.db.DLIDatabaseView` class. This class can then be used within a Java application to access the IMS data which was previously inaccessible by the application due to the mainframe-based format.

Not only can the DLIModel utility generate metadata for use with IMS Open Database, but it can also generate elements that you can use with other functions, like exposing IMS database operations as Web services. For example, the utility can:

- ▶ Generate a graphical UML model that illustrates the IMS database structure in an interactive tree model
- ▶ Generate annotated XML schemas of IMS databases, which are used to retrieve XML data from or store XML data in IMS databases
- ▶ Incorporate additional field information from COBOL or PL/I copybooks
- ▶ Incorporate additional PCB, segment, and field information, or override existing information through a graphical view of the PCB
- ▶ Generate a DLIModel database report, which assists Java application programmers in developing applications based on existing IMS database structures
- ▶ Generate an optional DLIModel trace log
- ▶ Provide a configuration editor as a launching point for generating IMS database Web services artifacts
- ▶ Generate XMI descriptions of the PSB and its databases

Refer to Figure 4-1 for a representation of the input that the DLIModel utility accepts and what it generates as output.

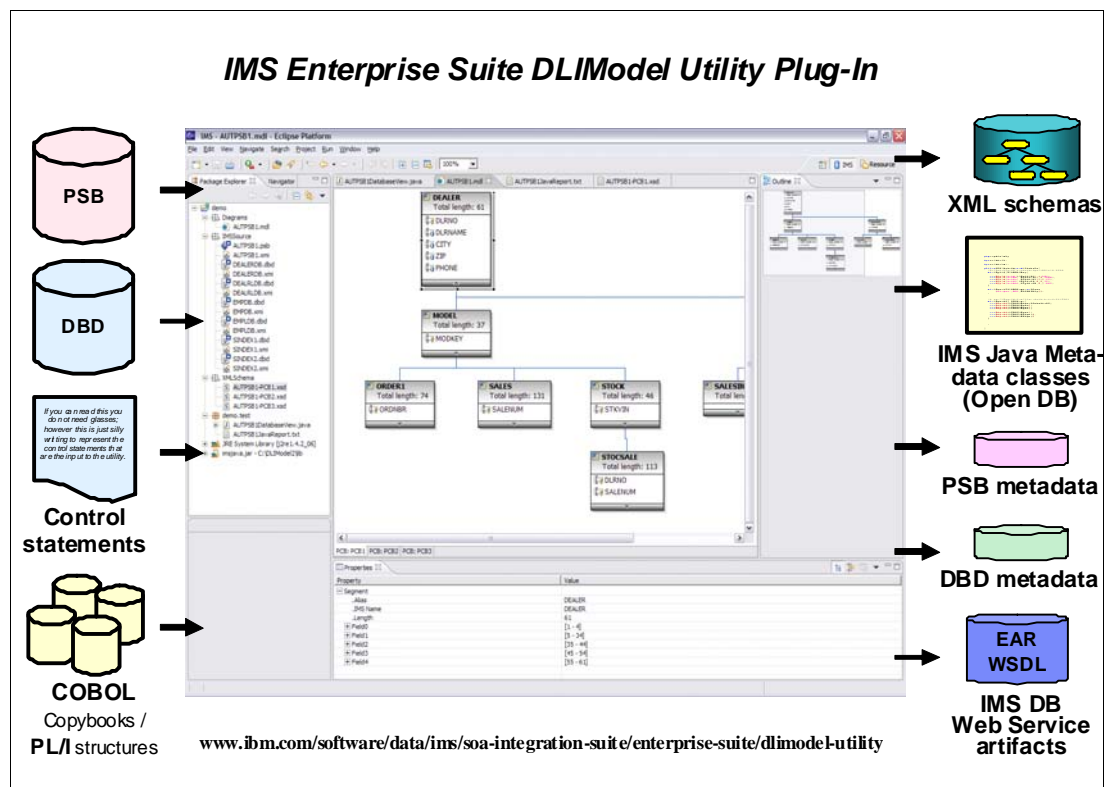


Figure 4-1 Input and output associated with the IMS Enterprise Suite DLIModel utility

For more detail about the utility's generation capabilities, access the online IMS Information Center by going to the IMS homepage at <http://www.ibm.com/ims> and entering the search string *DLIModel utility*. To locate the Information Center from the IMS homepage, click the Library link on the left as shown in Figure 4-2.

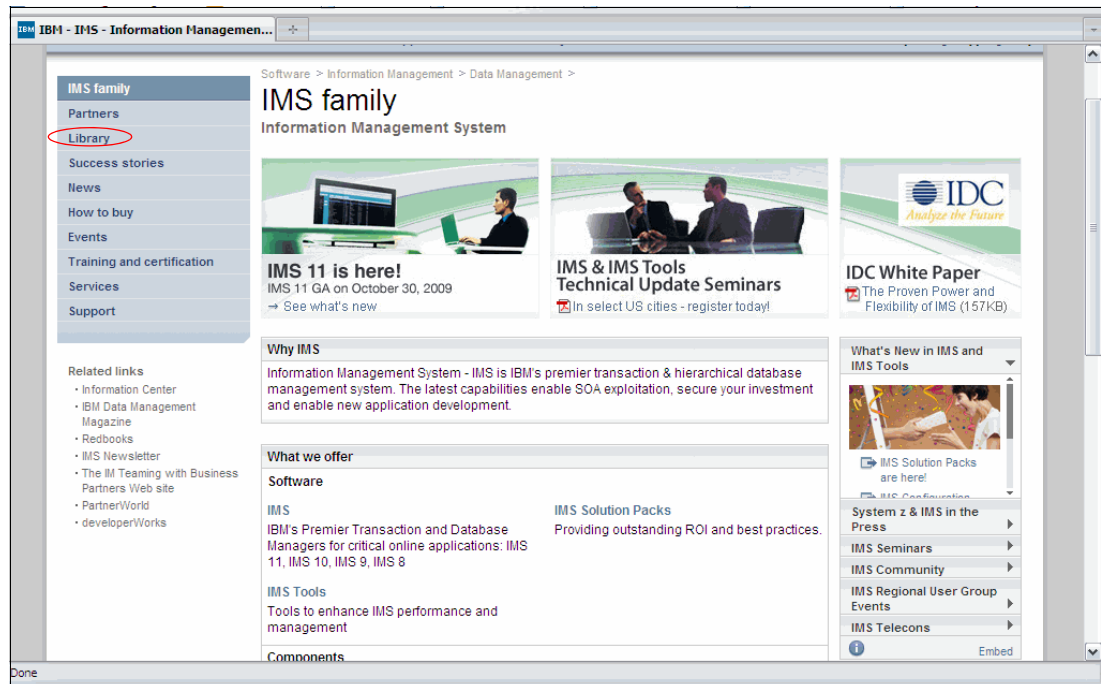


Figure 4-2 IMS family main panel

Then click the *Information centers* link shown on the next page. Now that we have described a bit about the DLIModel utility's capabilities, let's now take a look at its requirements.

## 4.2.1 Requirements

To use the DLIModel utility, certain software requirements must be met. This includes the scenario where you are migrating from an older version of the utility to the current one. You must be using:

- ▶ Eclipse Version 3.4.1
- ▶ Graphical Editor Framework (GEF) 3.4.1
- ▶ Eclipse Modeling Framework (EMF) 2.4.2

There are also other general requirements that need to be met before you can use the DLIModel utility:

- ▶ Eliminate all errors from your PSB and DBD source code, as well as from your COBOL or PL/I copybooks before running the utility, as it does not validate the input data.
- ▶ Obtain your IMS source files first, then copy them to your distributed platform.
- ▶ Include a name for all PCBs in each PSB definition using statement labels or the PCBNAME parameter.
- ▶ Ensure that all PSBs and DBDs either directly or indirectly related to resources included in the IMS source files are accessible. This includes DBDs that are indirectly referenced by a PSB in the case of secondary indexing on a main database, as well as DBDs that are externally referenced by other DBDs due to a logical relationship.
- ▶ Specify PROCOPT=P in your PCB statements or in the SENSEG statement for the root segment during PCB generation if your application includes JDBC calls that span more than one segment in a hierarchical path.

- ▶ When your application is using IMS Java hierarchical database interfaces (SSA database access) to retrieve IMS data, ensure that you choose appropriate PSB processing options since you are controlling path calls.
- ▶ Keep track of the length field when you have variable length segments for INSERT or UPDATE statements.
- ▶ Ensure that your COBOL and PL/I copybook files that supply additional information about field layouts describe physical segments. The files cannot describe logical database segment layouts.

Now that we are familiar with the requirements for using the DLIModel utility, we will now review some restrictions that need to be adhered to when using the utility.

## 4.2.2 Restrictions

When using the DLIModel utility to transform IMS data, there are a few items to keep in mind. We previously mentioned that utility can process PSB libraries, DBD libraries and COBOL and PL/I copybooks.

In the case of the copybooks, you can only import these if you have installed the utility plug-in into IBM Rational Application Developer for WebSphere Software or IBM Rational Developer for System z Version 7.5 or later.

In the case of DBD libraries, it can process all database organizations except MSDB, HSAM, and SHSAM databases. It can process all types and implementations of logical relationships, and secondary indexes, except for shared secondary indexes. Lastly, it is not able to process the PROCOPT=K option in a PSB SENSEG statement, which makes an application sensitive only to the segment key of a segment.

If you would like to modify actual IMS mainframe data, you must first acquire the respective source files from the z/OS system either by copying them from the mainframe using FTP copy or by copying them over to a distributed platform. The DLIModel utility cannot manipulate this data.

The DLIModel utility does not use DLTypeInfoList classes in its generated classes. If you want to define repeating groups of fields in segments other than by explicitly defining each group of fields separately, you must create the classes manually or modify the classes that are generated by the DLIModel utility.

Field types will vary depending on how they were created. When it is defined by a DBD, it is a CHAR field type but when imported from a COBOL or PL/I copybook, it is defined by the COBOL or PL/I copybook. You can change the field type by modifying it manually.

## 4.2.3 History

Prior to IMS 11, the DLIModel utility could be shipped with IMS and run from UNIX System Services or from the z/OS BPXBATCH utility. It required knowledge of z/OS development and also required writing control statements. However, this version of the utility is no longer supported (after IMS 10), so your sole option for obtaining the utility is via Web download.

The Web download version of the DLIModel utility was also available prior to IMS 11 as a plug-in, but it was packaged differently in that it was not yet part of the IMS Enterprise Suite. It was simply called the “IMS DLIModel utility plug-in”. This version of the utility is still available for download, but it does not include enhancements made to the newly packaged version of the utility, which is currently called the “IMS Enterprise Suite DLIModel utility plug-in”. These

latest enhancements include a shell-sharing capability with an existing Rational Application Developer Software for WebSphere or Rational Developer for System z. We therefore recommend that you use the latest version of the DLIModel utility and download the IMS Enterprise Suite DLIModel utility from the Web using the IBM Installation Manager. This will allow you to take advantage of the most current utility features.

## 4.3 Download and installation

To obtain the IMS Enterprise Suite DLIModel utility plug-in, you must use the IBM Installation Manager and configure its installation repository. Begin by navigating to the IMS homepage located at <http://www.ibm.com/ims>.

On this page, scroll down to find the IMS SOA Integration Suite link, as shown in Figure 4-3.

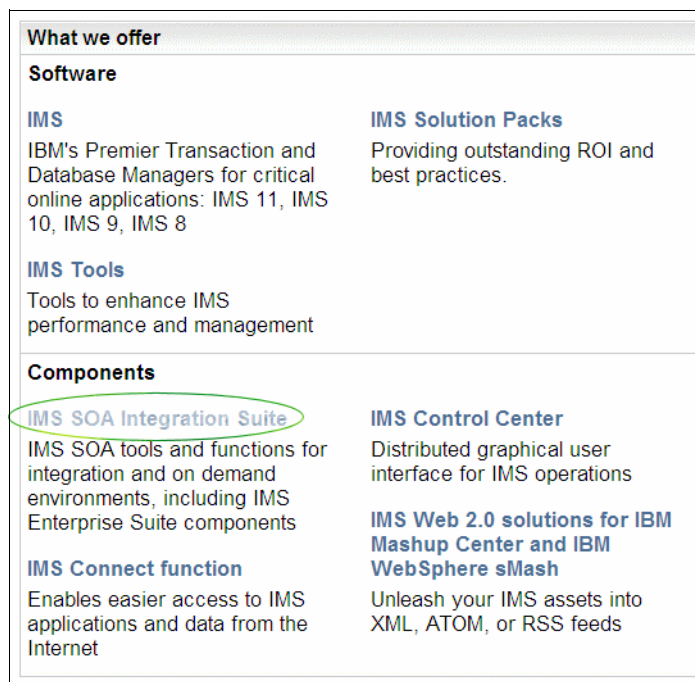


Figure 4-3 Starting point for downloading the IMS Enterprise Suite DLIModel utility plug-in

Follow the “IMS Enterprise Suite” link on the next two panels to navigate to the IMS Enterprise Suite download site. You will be prompted to sign in with your IBM ID (or create one if you do not yet have an IBM ID) and agree to the license terms before proceeding. The next page will present a checkboxed list of items that you can select for download, as shown in Figure 4-4. To download both the IBM Installation Manager and the IMS Enterprise Suite DLIModel utility components using Download Director, select the following items in the list and then click the “I confirm” button at the bottom of the page:

- ▶ Installation Manager for Windows
- ▶ IMS Enterprise Suite DLIModel utility plug-in for Red Hat® Linux and Windows XP
- ▶ Agreement to the license terms

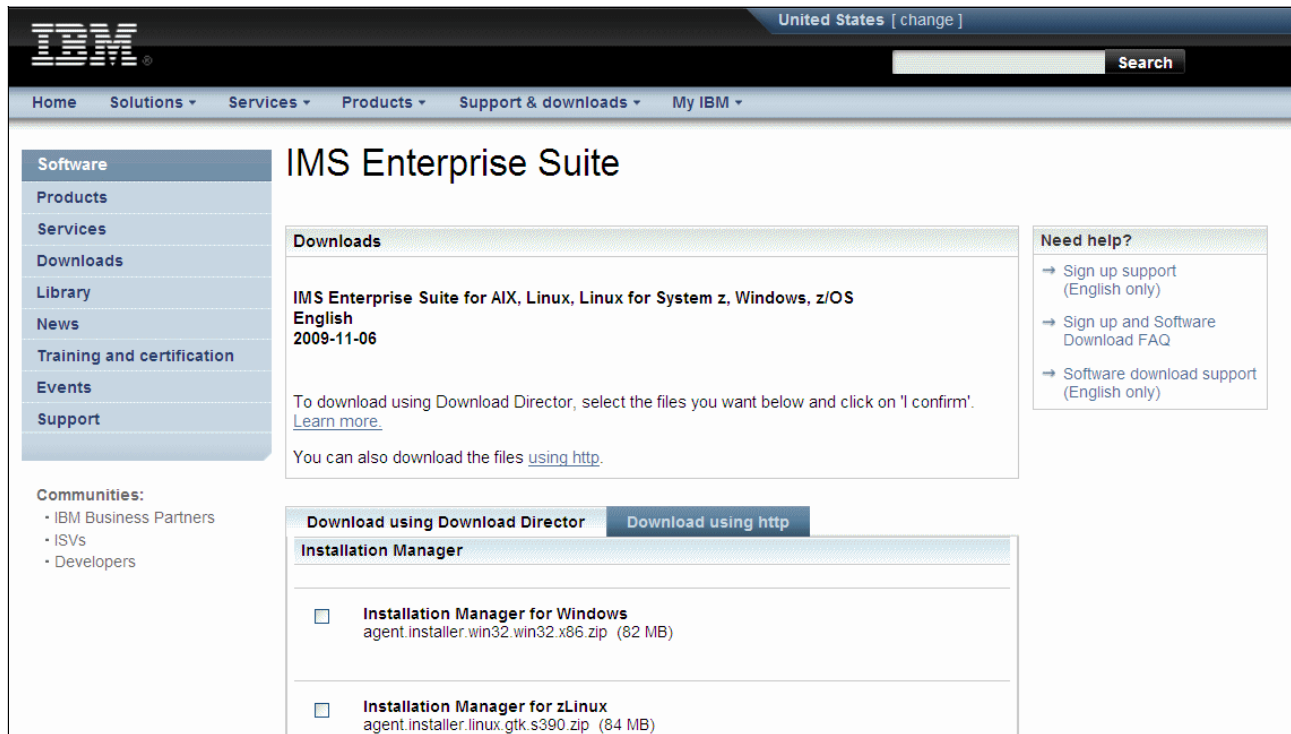


Figure 4-4 Download page for the IBM Installation Manager and IMS Enterprise Suite DLIModel utility plug-in

Note that on the page shown in the figure, there is one item in the list entitled “Contains all IMS Enterprise Suite features. SMP/E installable” under the “IMS Enterprise Suite (SMP/E)” heading. The DLIModel utility is part of this particular item, however if you select this, all of the other IMS Enterprise Suite components will be downloaded as well (and can be installed via SMP/E). We recommend that you select the individual DLIModel utility link as mentioned previously since the other IMS Enterprise Suite components do not play a role in the IMS Open Database capability.

Once you have clicked the “I confirm” button, two files will be downloaded:

- ▶ agent.installer.win32.win32.x86.zip (IBM Installation Manager)
- ▶ dlimodel\_distribute\_repository.zip (IMS Enterprise Suite DLIModel Utility)

Let’s now review the installation procedures for each of these.

### 4.3.1 Installing the IBM Installation Manager

To install the IBM Installation Manager, open the downloaded agent.installer.win32.win32.x86.zip file and use the “install.exe” file to initiate the installation process as highlighted in Figure 4-5.



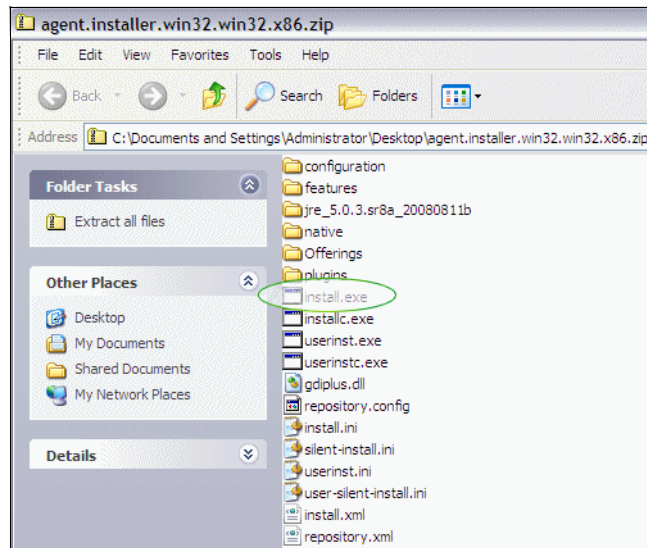


Figure 4-5 The *install.exe* file initiates the IBM Installation Manager installation process

Follow the prompts to complete the installation and agree to the terms of the license agreement. Once the IBM Installation Manager has been successfully installed, you will see the panel shown in Figure 4-6.

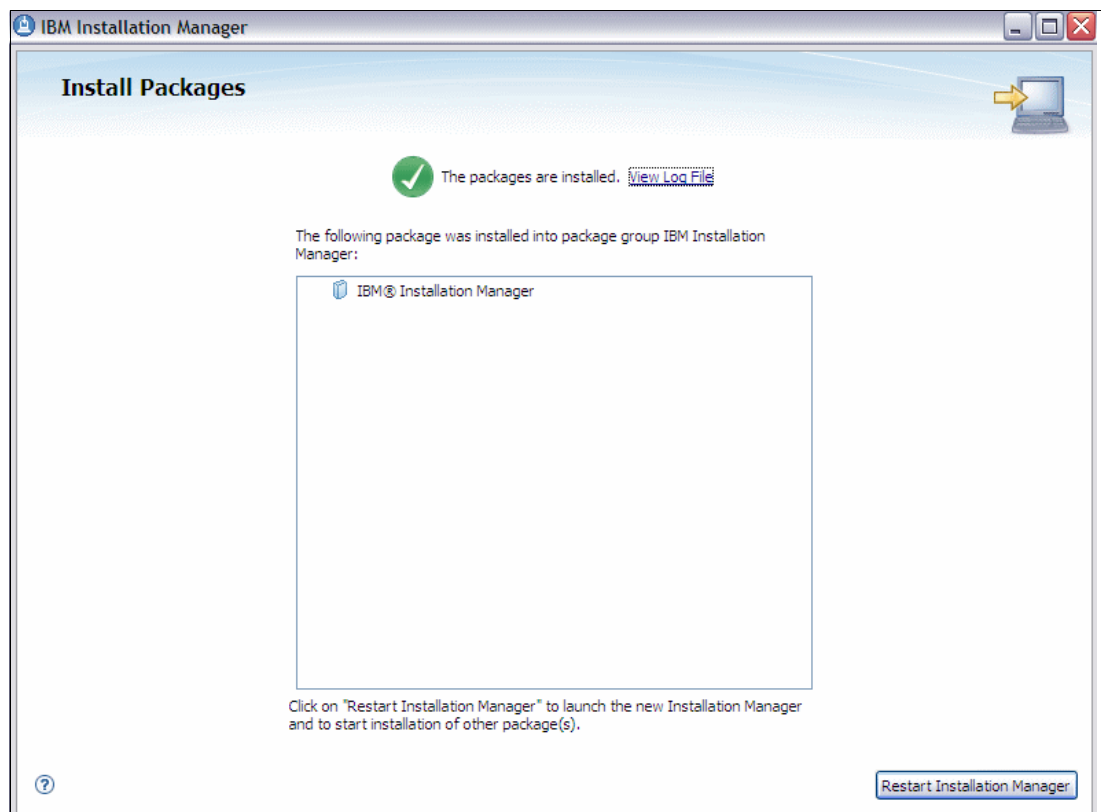


Figure 4-6 The confirmation panel after the IBM Installation Manager has been installed



**Attention:** Any program installed by the IBM Installation Manager will not appear in the standard Windows Control Panel “Add/Remove Programs” list. You must use the IBM Installation Manager to remove any program that it has installed.

### 4.3.2 Installing the IMS Enterprise Suite DLIModel utility

Now that you have installed the IBM Installation Manager, go ahead and launch it in order to install the IMS Enterprise Suite DLIModel utility. The IBM Installation Manager’s main menu is shown in Figure 4-7.



Figure 4-7 The main menu displayed when the IBM Installation Manager is launched

When you use the IBM Installation Manager to install software packages, you must connect to repositories that contain these software packages. To connect to the repository associated with the IMS Enterprise Suite DLIModel utility, select “Preferences...” from the File menu. Then click the “Add Repository...” button and select the `dlimodel_distribute_repository.zip` file that you just downloaded from the webpage in the previous section to add it to the dialog as shown in Figure 4-8.

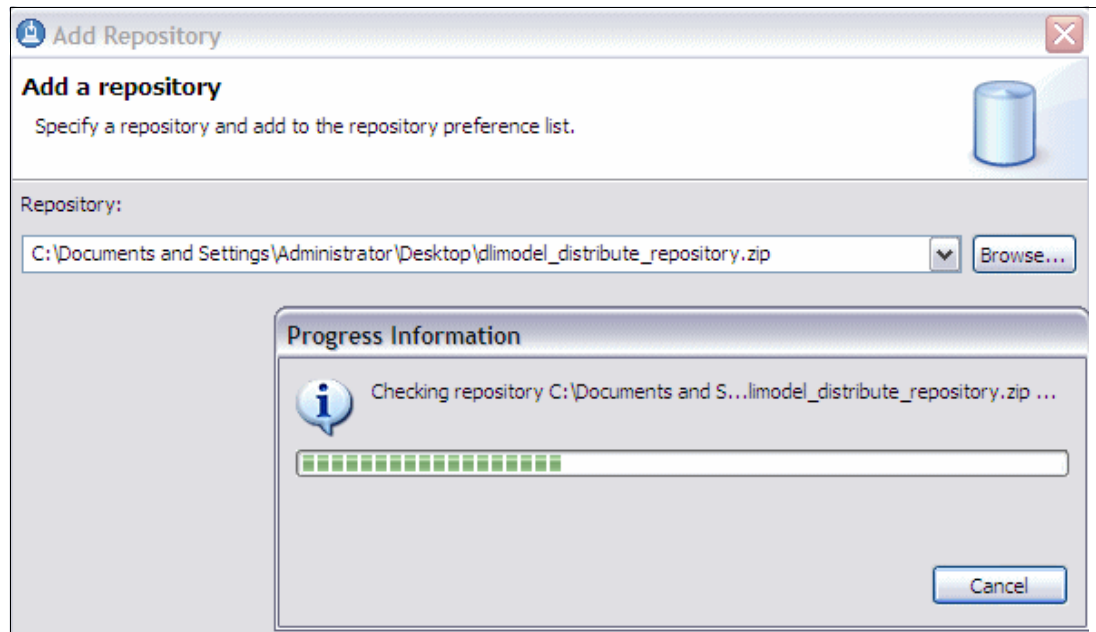


Figure 4-8 Adding the DLIModel utility repository to the IBM Installation Manager

Now that you have added the repository, click OK. After a few seconds you will be brought back to the main menu of the IBM Installation Manager, where you can now click the “Install” icon. The next panel displayed will contain the packages that are available to install. Check the DLIModel utility plug-in package as shown in Figure 4-9.

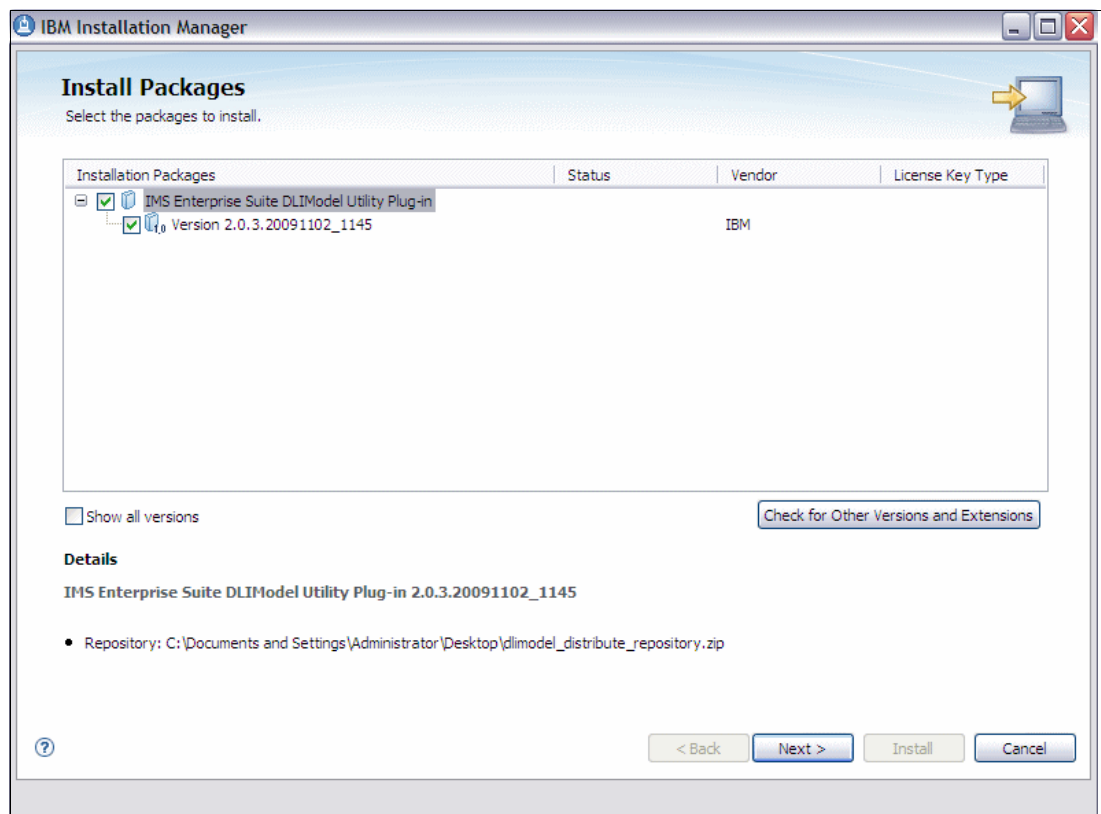


Figure 4-9 Selecting the IMS Enterprise Suite DLIModel Utility Plug-in package for installation

Click the “Next” button, accept the terms of the license agreement on the next page, and follow the rest of the dialog prompts to complete the installation process.

## Shell sharing

During the installation process, you will notice that one of the panels includes a checkbox that enables you to extend an existing Eclipse IDE or JVM, as shown in Figure 4-10. Leave this box unchecked unless you are already using Eclipse and simply want to add the Eclipse-based DLIModel utility features to your existing environment. In the latter case, check the box and specify the location of your Eclipse.exe and continue to follow the prompts.

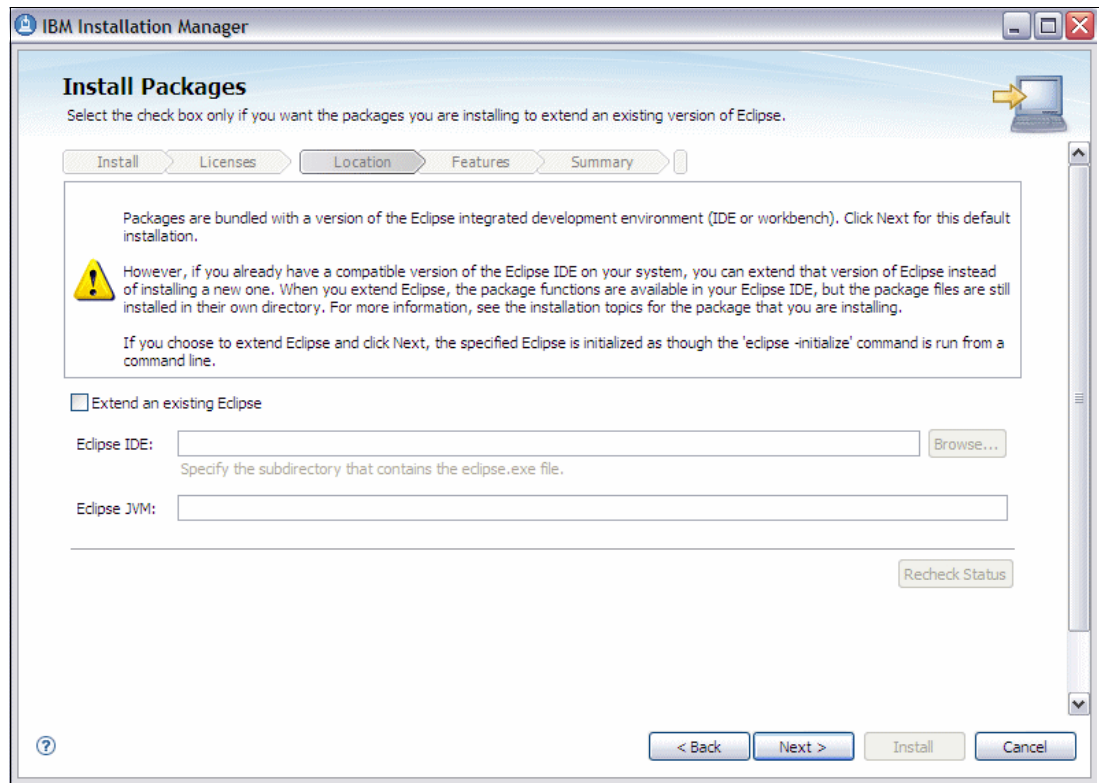


Figure 4-10 Option for the DLIModel utility to shell share with other Eclipse products

The ability to install products that will share a common environment is provided by the IBM Installation Manager and is known as “shell sharing”, which has several advantages. You are able to consolidate separate product features into one user interface, eliminating duplicate installations of the common environment on your machine. Not only does this save disk space and installation time, but it allows all products sharing the same shell to receive updates when they are installed by the IBM Installation Manager. Refer to section 4.6.3, “Integrate the DLIModel Utility with other Eclipse products” on page 94 for an alternate way to extend an existing Eclipse IDE.

**Restriction:** In order for the DLIModel utility plug-in to shell share with another Eclipse-based product, they both must be using the same version of Eclipse. In addition, the features associated with each product must not conflict with each other.

## 4.4 Setup for sample scenarios included in this book

In order to be able to follow the scenarios presented in the other chapters in this book, there are a few required prerequisite setup steps. These steps include downloading the sample database source, then using the DLIModel Utility plug-in to generate and edit its metadata, which you will ultimately export as a jar file for use with Java application development. We begin with describing how to download the database source.

### 4.4.1 Downloading the Car Dealer IVP database source

The IMS product includes a sample database called Car Dealer. The source associated with this database is included in the IVP and is located in the *HLQ.SDFSISRC* data set. In our case, the *HLQ* is 'IMS11B' so the data set we will use is named *IMS11B.SDFSISRC*.

You have several options to copy the database source from the host to your workstation. You can use the functions of your 3270 terminal emulation or you could use Rational Developer for System z to easily access the members in this data set. You can either copy/paste the members or use FTP to download them. In the following steps we show an example of downloading the data set members with the Microsoft Windows XP integrated FTP client:

- ▶ Click Windows **Start -> Run...** and type **cmd.exe** to open the command prompt.
- ▶ Start the FTP session by using the command **FTP <HOSTIP>** and when prompted, insert your TSO user name and your password. After login enter the following commands:

```
cd 'IMS11B.SDFSISRC'
lcd c:\temp to change to an existing directory on your system
ASC to set transfer mode to ascii
get DFSAUTDB AUTODB.dbd (for the DBD)
get DFSEMPDB EMPDB2.dbd (for the DBD)
get DFSEMLDB EMPLDB2.dbd (for the Logical DBD)
get DFSLAUTO AUTOLDB.dbd (for the Logical DBD)
get DFSIND22 SINDEXX22.dbd (for Index DBD)
get DFSIND11 SINDEXX11.dbd (for Index DBD)
get DFSAUT11 AUTPSB11.psb (for PSB)
```

- ▶ To exit the FTP session, type **quit** and to end the command prompt type **exit**.

You should now have the source members in your local directory, in this case *c:\temp*. You must rename the source members to match the DBD name statements that were included in the source data. To do this, use the 'get' command by specifying **get <sourcename on the current remote directory> <destination name on the current local directory>**.

For a detailed explanation of the Car Dealer database and its source code, refer to Appendix B, "Car Dealer IVP Database" on page 235. Now that you have obtained the sample database source files, you can use the DLIModel utility plug-in to generate metadata for it, which we will now cover.

## 4.5 Using the IMS Enterprise Suite DLIModel utility

To use the DLIModel utility plug-in to generate metadata and other artifacts that Java can understand, we will use the IMS source files obtained in the previous section as input. In this section, we discuss using the DLIModel wizard and DLIModel editor to generate and edit metadata associated with the sample database (respectively) as well as exporting this metadata as a jar file.

When working with the DLIModel utility, all associated data is contained within a 'DLIModel project', which you must first create. The DLIModel wizard assists you in creating a project and gives you the option of creating an XML schema, generating a result report, and writing a trace log. If you already have a DLIModel project that you are working with, you can import IMS PSB and DBD source files into it.

Once you have generated the metadata, you can use the DLIModel editor to modify it and also import information from COBOL or PL/I copybooks. The editor also allows you to search, save, and print the hierarchy of the IMS database.

### 4.5.1 Generating metadata for Car Dealer database

In this section, we examine how to use the DLIModel utility to generate the Java Metadata class file, which is required in the scenario use cases in this book. A prerequisite to this step is downloading the required source files, which we covered in 4.4, "Setup for sample scenarios included in this book" on page 86.

Once you have launched the Eclipse-based IMS Enterprise Suite DLI Model Utility, follow these steps to generate metadata for our Car Dealer sample database (which is included in the IVP):

- ▶ Click on **File->New->Projects** and select **DLIModel Utility Project** and click **Next**.
- ▶ Specify **AUTPSB11** for the project name and **samples.ims.openDb** for the Java package name. If you show the 'Advanced' section by clicking the button, you can then select additional options for the generation, including the ability to use an IMS control statement from the former version of the DLIModel utility to generate the Metadata Class file (see Figure 4-11).

**Important:** Be aware that the Java package is case-sensitive and is referenced in all scenarios in this book as the package and class name shown here.

**New DLIModel Utility Project**

Create a new DLIModel utility project.

Project name:

Java package:

**Project contents**

☒ Use default

Directory:

**Select optional output files**

☒ XML schema

☒ DLIModel utility result report

☐ Trace log

☐ Run utility from an IMS control statement [Find out more](#)

Location of control statement file:

Figure 4-11 New DLIModel Utility Project - Step 1

- ▶ Leave the defaults as they are and click **Next**.
- ▶ Highlight your local directory and select the downloaded AUTPSB11.psb file as shown in Figure 4-12. Since the DBD source files are in the same directory and will be named as defined in the PSB's DBDNAME statement, it will automatically find and select the associated DBDs. Otherwise, you must manually select all of the DBD source members associated with the databases and indexes. These members can have either file extension of .dbd or .psb or none.

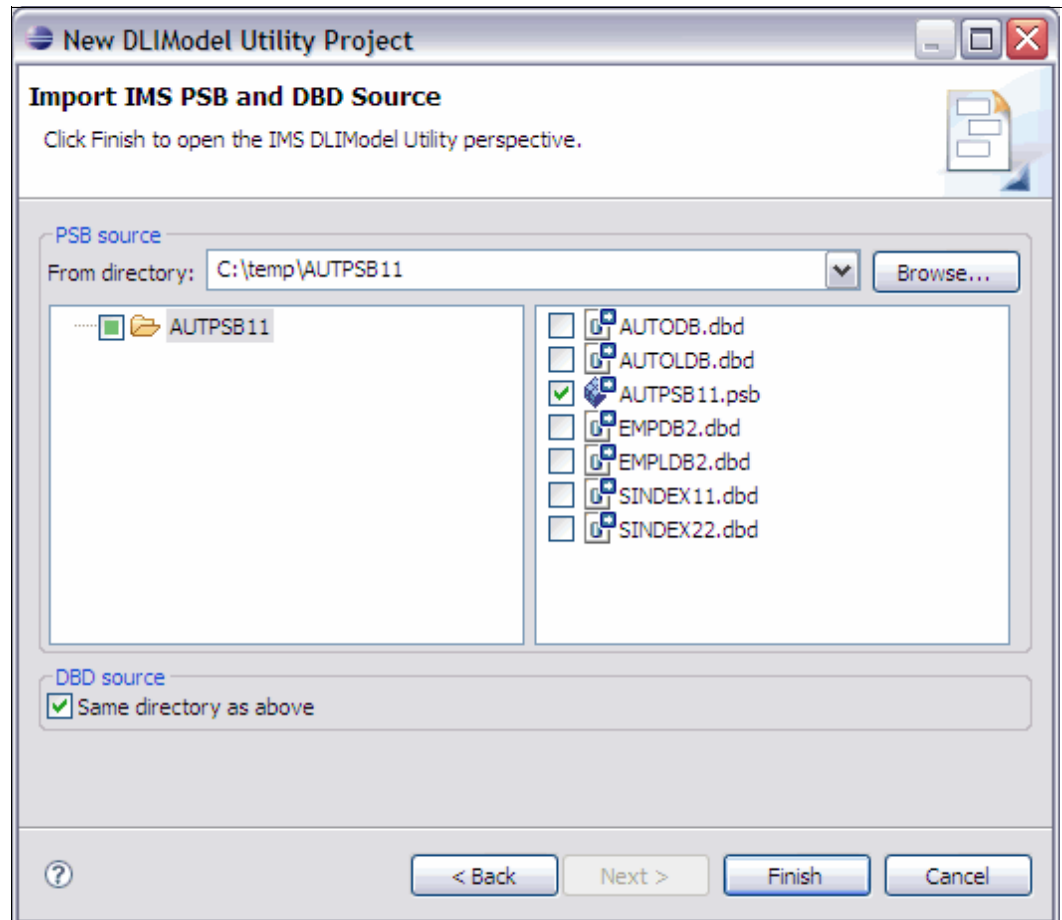


Figure 4-12 New DLIModel Utility Project - Step 2

- Click **Finish** and you will see the generated files in the Package Explorer as shown in Figure 4-13.

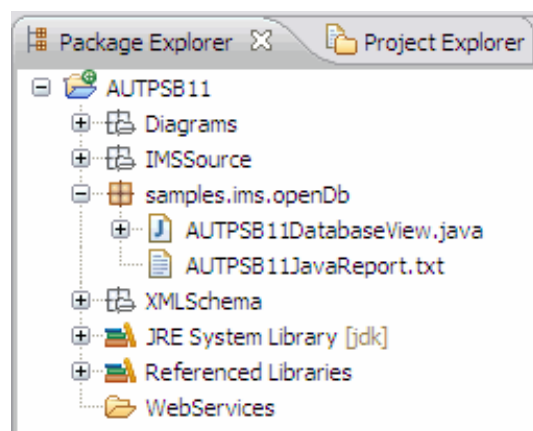


Figure 4-13 New DLIModel Utility Project - Step 3

- The required file is in the **samples.ims.openDb** Java package and is automatically called **AUTPSB11DatabaseView**.

## 4.5.2 Editing the AUTPSB11 Project

Since the DLIModel utility is not able to obtain all of the required data from the PSB and DBD source files, you must review and if necessary edit the files that have been generated.

After generating the files now contained in the Project, the Overview diagram should open itself automatically as shown in Figure 4-14. Notice that at the bottom of this view, you can switch between different PCBs associated with this PSB. Any saved changes that you make in the diagram will result in the regeneration of the files in the project.

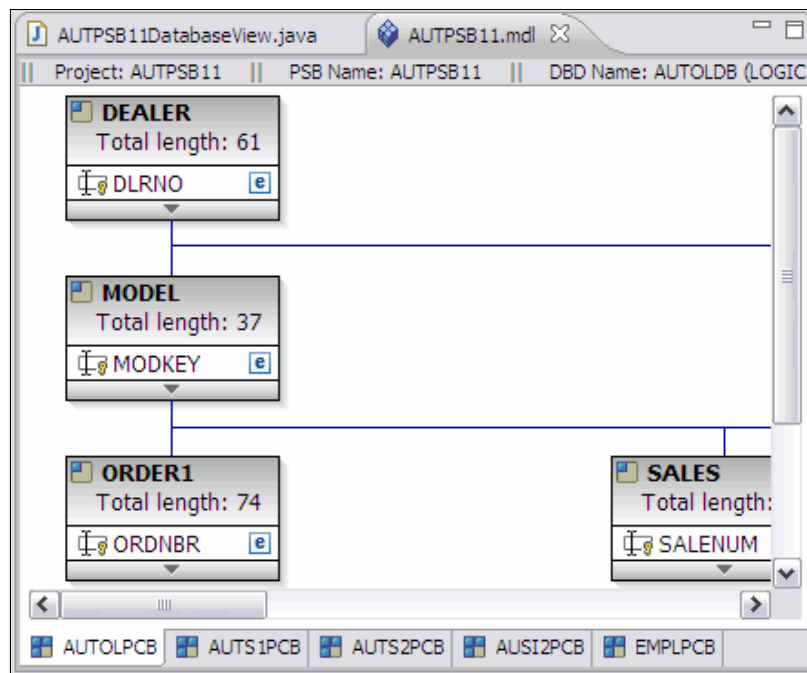


Figure 4-14 AUTPSB11 Overview diagram

When you look at the definition of the different fields, you may notice that the fields are all defined as Character Data Type. This is because IMS doesn't have a large variety of data types. The only differentiation which the DBDs and PSBs make concerning the data type is in the FIELD parameter and can be any of the following:

- ▶ X stands for Hexadecimal data
- ▶ P stands for Packed decimal data
- ▶ C stands for Alphanumeric data or a combination of types of data
- ▶ F stands for Binary fullword data (only MSDB)
- ▶ H stands for Binary halfword data (only MSDB)

However, it can be important to change the data type from Character to another data type -- for example if it is a Packed decimal field in IMS. This particular field format is normally handled by the application in IMS. For example, the Packed decimal field which needs five bytes can contain more than five numbers. The format of these fields is normally specified by the IMS application. Therefore, you have two options for correcting the data types:

- ▶ Use the COBOL copybook import function to get the required data out of the copybook. This is done by right-clicking on the diagram and selecting Import Copybook Fields as shown in Figure 4-15.



**Note:** The import COBOL copybook feature only works if your DLI Model Utility is running in Rational Application Developer or Rational Developer for System z.

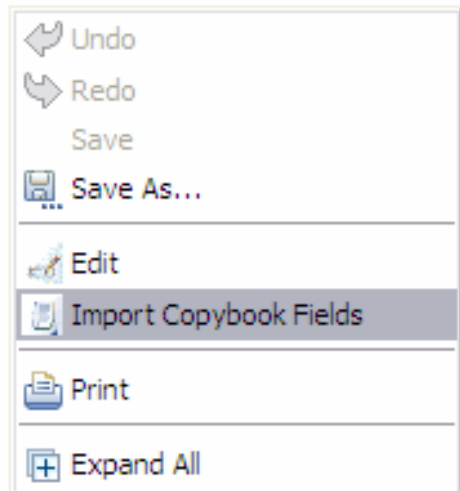


Figure 4-15 Import Copybook Fields

- The second option is to specify the values manually in the diagram by right-clicking a field and select Edit.

Open the **AUTODB.dbd** in the Project's **IMSsource** folder of the Package Explorer. When you have a look at the types you will find the following fields which are not TYPE=C in the ORDER segment:

```
FIELD NAME=MSRP,BYTES=5,START=27,TYPE=P
FIELD NAME=COUNT,BYTES=2,START=32,TYPE=P
```

and the following line the STOCK segment:

```
FIELD NAME=WRNTY,BYTES=1,START=46,TYPE=X    NEW (BOOLEAN)
```

You have two options for this example.

1. **Leave the fields as CHAR (recommended)**, as they are currently filled with CHAR data by the IVP jobs.
2. You can also delete the contents in the database manually (e.g. with a DFSDDLTO job) and adjust the types in the diagram. Since there is no copybook provided for this IVP sample database, you must adjust them manually. To do this, right-click in the diagram on one of the values and click **Edit** or use the Properties view to edit the fields. Specify **MSRP** as data type **PACKEDDECIMAL** with the representation of **S99999999V99** and **COUNT** as data type **PACKEDDECIMAL** with the representation of **S999**. Change the **WRNY** field to **BIT** as it is used here only as a Boolean field with two hexadecimal values

**Attention:** The MSRP and COUNT segments are currently loaded by the IVP with a few values in CHAR format which are not PACKEDDECIMAL as defined. So the type conversion does not work if you specify these fields as PACKEDDECIMAL.

### 4.5.3 Export the metadata as Jar file

After finishing editing the Project you can now export the project by following these steps:

- ▶ Click on **File->Export** and select **Java->Jar file** and click **Next**.
- ▶ By default the whole project should be selected for export. **Deselect the diagram** and the **WebService** folder. Select the **Export Java Source files and resources** checkbox to include the source files in your Jar file. This way, when you provide the Jar file to your application developer, (s)he will be able to review the generated Java Report and the source code used to generate the files. Also, if you have future database changes, you will be able to easily identify the correct version of the Jar file. Type in the path and the name where the Jar file should be exported. In Figure 4-16 you can see that we have specified c:\temp\AUTPSB11.jar.

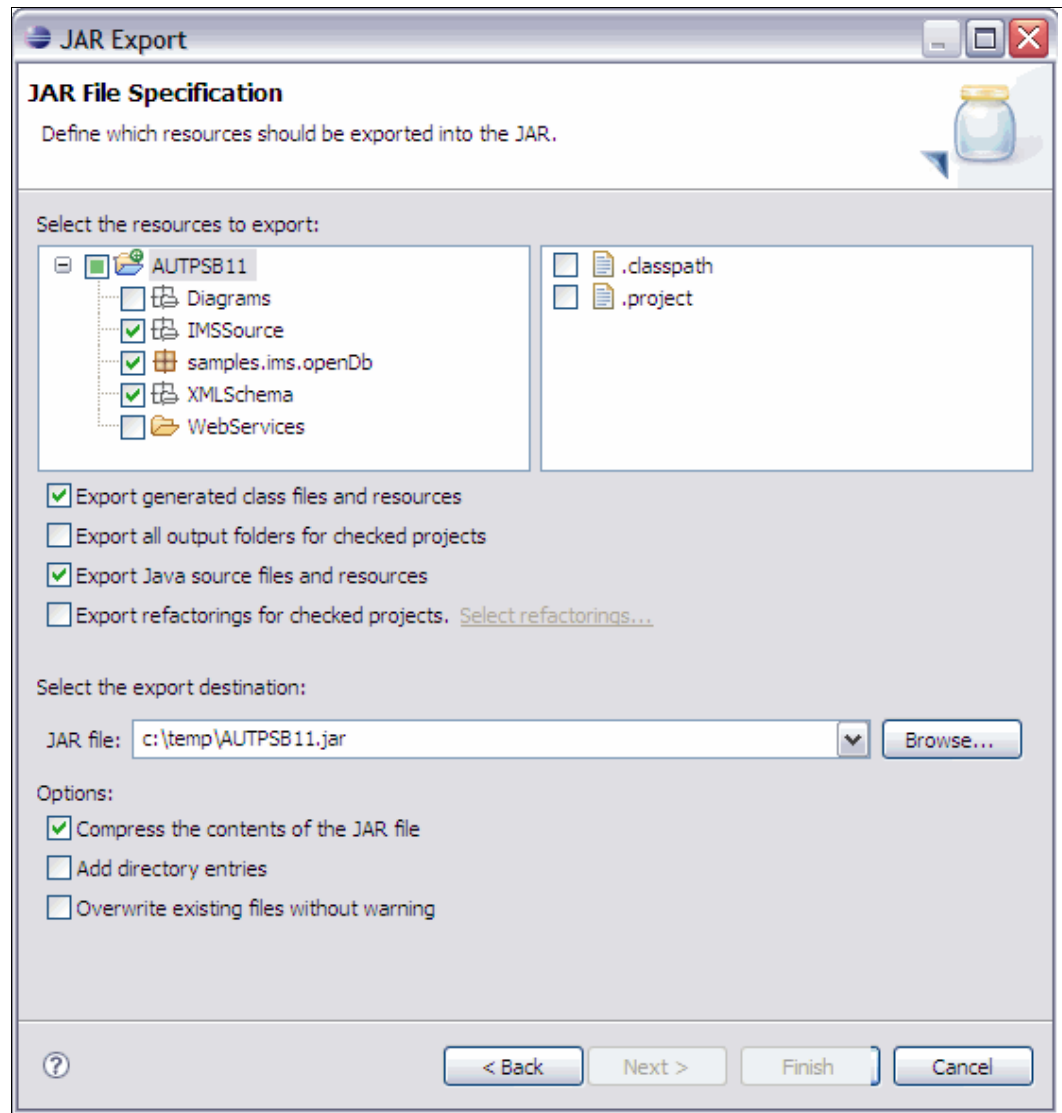


Figure 4-16 Export of AUTPSB11.jar

- ▶ Click **Finish** to complete the Export.

## 4.6 Additional considerations for the IMS Enterprise Suite DLI Model Utility

There are several other considerations for using the IMS Enterprise Suite DLI Model Utility, which we discuss here.

### 4.6.1 Ensuring consistency between generated class files and other JRE files

The current Eclipse version of the IMS Enterprise Suite DLIModel utility uses JDK 6.0 by default to generate the metadata class files. If you will be working with other files that are based on a different Java Runtime Environment (JRE) -- for example, version 1.5 -- you must change the version to match before compiling a java application. If you do not do this, you will receive a *Java class version* error when you compile. To change the version, follow these steps:

- ▶ Right-click on your project and select **Properties**.
- ▶ Select the **Java Compiler** page and select the **Enable Project specific settings** checkbox. Select your **Compiler compliance level** to the level you need. In Figure 4-17 it is specified as 1.5.

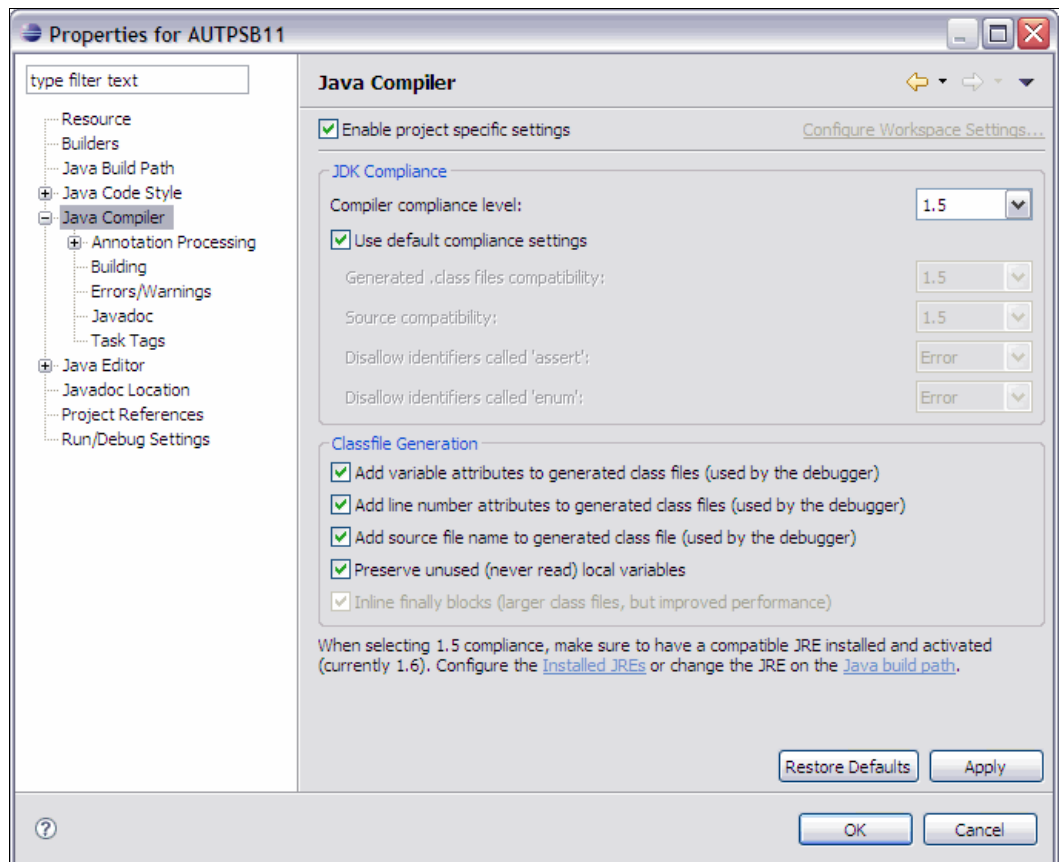


Figure 4-17 Compiler compliance level

- ▶ You will be asked to regenerate the project. Click **Yes** to rebuild it. Make sure that you export your Jar file again in order to be able to use it.

## 4.6.2 Track changes of IMS database reorganizations

If you change your IMS database design and do a reorganization of a database, you must also recreate the metadata files for this database. This should be considered in the change management process of your company when databases are being changed. If you do not have the source of your DBDs and PSBs, you can use IMS Tools such as the *IBM IMS Library Integrity Utilities for z/OS* to recover it from the IMS libraries. This also guarantees that you have the most current source for your database definitions and PSBs.

## 4.6.3 Integrate the DLIModel Utility with other Eclipse products

If you want to use the DLIModel Utility without installing it on every PC, one option is copying the plug-in to the Eclipse plug-in folder. As a reminder, the Eclipse installation must have the minimum requirements to run the DLIModel Utility plug-in. If you did not originally use the IBM Installation Manager to install Eclipse, we recommend that you manually check for updates associated with newer Eclipse versions. To copy the plug-in manually, follow these steps:

- ▶ Copy the **com.ibm.ims.dlimodel.ui\_2.0.3** directory from your installation directory (whose default is **C:\Program Files\IBM\IMS Enterprise Suite V1.1\DLIModel Utility Plug-in\plugins**) to your target Eclipse plug-ins directory. If you are using Rational Developer for System z, version 7.6 the default is **C:\Program Files\IBM\SDP\plugins**.

**Note:** With newer Eclipse versions you do not have to issue an Eclipse **-clean** command to load the plug-in. Instead, simply restart your Eclipse to begin using it.

## 4.6.4 Data type conversion table

Table 4-1 shows a mapping of COBOL copybook data definitions and how they can be mapped to the DLIModel Utility.

Table 4-1 Conversion table - Copybook format to data types

Copybook format	DLIType info data type	Java data types
PIC X(25)	CHAR	java.lang.String
PIC S9 (1-4 figures) COMP	SMALLINT (2 Bytes)	short
PIC S9 (5-9 figures) COMP-4	INTEGER (4 Bytes)	int
PIC S9 (10-18 figures) BINARY	BIGINT (8 Bytes)	long
COMP-1	FLOAT	float
COMP-2	DOUBLE	double
PIC S9(06)V99 COMP-3	PACKEDDECIMAL	java.math.BigDecimal
PIC S9(06)V99	ZONEDECIMAL	java.math.BigDecimal
PIC 9(06).99	ZONEDECIMAL	java.math.BigDecimal
PIC 9 DISPLAY	ZONEDECIMAL	java.math.BigDecimal



# IMS Open Database for application developers

In this chapter, we provide an overview about the IMS Version 11 Open Database functions and how to use them for developing applications.

This chapter is intended to be read by Java application programmers who want to get an understanding of what the IMS Version 11 Open Database feature can provide for developing applications, the components involved and their requisites.

In this chapter we discuss the following topics:

- ▶ Overview of IMS Open Database on the application side
- ▶ Architectural considerations
- ▶ IMS Universal Database resource adapter
- ▶ IMS Universal JDBC driver (Stand Alone)
- ▶ IMS Universal DL/I driver
- ▶ SQL syntax for the IMS Universal drivers

## 5.1 Overview of IMS Open Database on the application side

Since IMS Version 11 it is much easier to get to data stored in IMS Databases. The Open Database feature from IMS Version 11 enables you to leverage your existing valuable IMS data by reusing it easily in new applications.

IMS provides certain drivers for different environments and programming paradigms. The exciting about this drivers are, that it is for the first time possible to access IMS data out of the box without any additional Tool or Product. And it also doesn't matter whether the application is running on the z/OS Mainframe side or it is running on any other distributed platform. This helps your company to keep costs down for additional Tools and Products and get rid of additional steps like replicating the data to an accessible place.

For developing a Java application to access IMS data you will need, beside your Integrated Development Environment (IDE), one of the IMS Universal DB drivers as well as the Database metadata for the databases you want to access in IMS.

### 5.1.1 IMS Universal DB drivers

The IMS Universal DB drivers are a set of Java classes and resource adapters that enable the access to IMS databases. They are built on industry standards and open specifications. The IMS Universal DB drivers are able to communicate remotely by using TCP/IP (type-4 connectivity) and locally (type-2 connectivity).

The type-4 connectivity is based on the Distributed Relational Database Architecture (DRDA) protocol, but you do not need to know how to program with DRDA if you are using Java as programming language. If you are using the type-4 connectivity, it will need IMS Connect on the z/OS side, which is an integral part of IMS. IMS Connect interacts as the TCP/IP Endpoint for the IMS Universal drivers.

The IMS Universal DB drivers also have a local option (type-2 connectivity) included. You can use this type of connectivity for accessing IMS data when your application program is running on the same logical partition (LPAR) as your IMS. For the local connectivity you do not need IMS Connect as TCP/IP is not used for this direct communication.

The drivers have different implementations you can use based on your application environment and your application architecture. They provide an application programming framework that offers the greatest choice of options for accessing IMS data. These programming options include:

- ▶ IMS Universal DB resource adapter: A Java EE Connector Architecture (JCA) 1.5-compliant resource adapter for your managed JEE Environment. It provides access to IMS data using the Common Client Interface (CCI) and Java Database Connectivity (JDBC) interfaces
- ▶ IMS Universal JDBC driver: A stand-alone Java Database Connectivity (JDBC) driver that implements the JDBC 3.0 API for making SQL-based database calls against IMS Databases
- ▶ IMS Universal DL/I driver: A Java API for making calls with traditional DL/I programming semantics

The IMS Universal drivers are part of the IMS Installation Procedure (SMP/E) as an optional part (FMID JMK1106). They are maintained during the normal maintenance process of IMS.

The IMS Universal drivers are mounted in the z/OS OMVS file system and can be downloaded via FTP from your mainframe. Usually they are mounted to the following path:

```
/usr/lpp/ims/ims11/imsjava/
```

The exact path to download the files may vary depending on your system configuration. Ask your IMS System programmer for help on how to get the drivers.

**Note:** Make sure that, when you download the IMS Universal drivers via FTP, you download them in binary mode.

## 5.1.2 IMS database metadata

IMS has its own catalog-like library called ACBLIB. There are all information about the IMS application's view on the database called PSB (Program Specification Block) and the Physical Layout of the database called DBD (Database Definition) stored. The ACBLIB is currently only accessible by IMS itself and not by external applications. But the IMS Universal DB driver has to know how the database looks like when you want to access it from your application. Therefore you need metadata about the database you want to access in your application.

The metadata files are Java class representations of the IMS Database View generated out of the PSB and DBD sources from IMS. This step is done with the help of the IMS Enterprise Suite DLI Model Utility. This step should be done from your IMS System programmer, because he has access to the necessary resources in IMS.

The metadata Java class defines maps the hierarchy of the segments to Java arrays of the Type `DLTypeInfo`. These arrays contains the fields and defines the data type of each column, the column length and start offset within the segment. Example 5-1 shows an extract of the metadata Java class from our Car Dealer IVP Example.

*Example 5-1 Extract of the Java metadata class from the Car Dealer IVP Example*

---

```
import com.ibm.ims.db.*;
import com.ibm.ims.base.*;

public class AUTPSB11DatabaseView extends DLIDatabaseView {
// This class describes the data view of PSB: AUTPSB11
// PSB AUTPSB11 has database PCBs with 8-char PCBNAME or label:
// AUTOLPCB,AUTS1PCB,AUTS2PCB,AUSI2PCB,EMPLPCB

// Constructor
public AUTPSB11DatabaseView() {
    super("2.0.3","AUTPSB11", "AUTOLPCB", "AUTOLPCB", AUTOLPCBArray, "AP");
    addDatabase("AUTS1PCB", "AUTS1PCB", AUTS1PCBArray, "GRP");
    addDatabase("AUTS2PCB", "AUTS2PCB", AUTS2PCBArray, "GRP");
    addDatabase("AUSI2PCB", "AUSI2PCB", AUSI2PCBArray, "GRDP");
    addDatabase("EMPLPCB", "EMPLPCB", EMPLPCBArray, "AP");
} // end AUTPSB11DatabaseView constructor

// The following describes Segment: DEALER ("DEALER") in PCB: AUTOLPCB ("AUTOLPCB")
static DLTypeInfo[] AUTOLPCBDEALERArray= {
    new DLTypeInfo("DLRNO", DLTypeInfo.CHAR, 1, 4, "DLRNO", DLTypeInfo.UNIQUE_KEY),
    new DLTypeInfo("DLRNAME", DLTypeInfo.CHAR, 5, 30, "DLRNAME"),
    new DLTypeInfo("CITY", DLTypeInfo.CHAR, 35, 10, "CITY"),
    new DLTypeInfo("ZIP", DLTypeInfo.CHAR, 45, 10, "ZIP"),
    new DLTypeInfo("PHONE", DLTypeInfo.CHAR, 55, 7, "PHONE")
};
static DLISegment AUTOLPCBDEALERSegment= new DLISegment
```

```

        ("DEALER", "DEALER", AUTOLPCBDEALERArray, 61);

// The following describes Segment: MODEL ("MODEL") in PCB: AUTOLPCB ("AUTOLPCB")
static DLTypeInfo[] AUTOLPCBMODELArray= {
    new DLTypeInfo("MODTYPE", DLTypeInfo.CHAR, 1, 2, "MODTYPE"),
    new DLTypeInfo("MODKEY", DLTypeInfo.CHAR, 3, 24, "MODKEY",
DLTypeInfo.UNIQUE_KEY),
    new DLTypeInfo("MAKE", DLTypeInfo.CHAR, 3, 10, "MAKE"),
    new DLTypeInfo("MODEL", DLTypeInfo.CHAR, 13, 10, "MODEL"),
    new DLTypeInfo("YEAR", DLTypeInfo.CHAR, 23, 4, "YEAR"),
    new DLTypeInfo("MSRP", DLTypeInfo.CHAR, 27, 5, "MSRP"),
    new DLTypeInfo("COUNT1", DLTypeInfo.CHAR, 32, 2, "COUNT")
};
static DLISegment AUTOLPCBMODELSegment= new DLISegment
    ("MODEL", "MODEL", AUTOLPCBMODELArray, 37);
...

```

---

This example shows that the Java class contains all segments with its fields as Java objects. The metadata class defines the type, the length and the name of the columns. These attributes can be changed manually if you have special needs.

**Note:** You can also see that the column “COUNT” in the “MODEL” segment has been assigned automatically the alias “COUNT1” as Count is a restricted SQL keyword.

### 5.1.3 Java version requirements

Java application programs that use the IMS Universal drivers require at least the Java Development Kit 5.0 (JDK 5.0). Java programs that run in IMS Java dependent regions require the Java Development Kit 6.0 (JDK 6.0) or later.

## 5.2 Architectural considerations

The IMS Universal drivers have several implementations you can choose from. Which driver you should take for application development and which options you have to set is depending on several factors:

### 5.2.1 Transactional support

You have to decide which level of transactional support you need in your application. JCA differentiates between drivers with Global Transaction support and Local Transaction support.

#### Global transaction support (XA)

The XA standard is an X/Open specification for distributed transaction processing (DTP). It describes the interface between the global transaction manager and the local resource manager. The XA specification describes what a resource manager must do to support transactional access. In the J2EE platform, the XA specification is driven via the transactional system contract. The interface the contract uses is the XAResource interface. A resource adapter that is XA compliant must implement this interface. At transaction commit time, the resource managers are informed by the transaction manager to prepare, commit, or rollback a transaction according to the two-phase commit protocol. There are also one-phase optimizations built into the XAResource system contract.



The X/Open XA Protocol ensures that all components in a transaction whether commit or rollback their changes for one transaction. This is done by using the 2 Phase Commit Protocol with all used resources. A higher instance, usually an application server, interacts as Sync Point Manager. It controls each connection and executes the necessary 2 Phase Commit functions for each resource in the responsible driver. The driver has therefore to implement the necessary functions for the two-phase commit handling.

If you choose this option with the IMS Universal drivers this means that Recovery Resource Services (RRS) will be used on the z/OS side and the JEE Container will take place as Sync point Manager across all connections to z/OS and distributed. IMS Connect builds the necessary RRS structure to support the two-phase commit protocol. The XA supporting drivers have a detection for one-phase commit processing, if no two-phase processing is necessary.

### **Local transaction support**

The Local Transaction support means, that each connection to a resource runs in an own transactional unit of work. This gives you the ability to decide for your own where you want set a commit point or rollback the transaction. But this means also that you have to take care for the consistency of your data for your own. This can be very complex, especially when you have an application which connects to IMS and other resources like DB2 and you do not have global transaction support in place across the different drivers and connections. You have to commit and rollback each unit of work individually and take care for data integrity in case of application failures.

Local Transaction support can be useful if you have an application which only connects to one resource. It can be faster because the driver doesn't have to handle the two-phase commit functions. It can be also useful if you do not want to use RRS on the z/OS side or if you have a need for individual handling of the commit-points

## **5.2.2 Access types**

Two major types of connectivity are supported by the IMS Universal drivers. Local connectivity to IMS databases on the same Logical Partition as IMS (type-2 connectivity) and distributed connectivity through TCP/IP from anywhere (type-4 connectivity).

### **Distributed access (type-4 connectivity)**

Distributed Access means that the IMS Universal drivers can run on any platform that supports TCP/IP and a Java Virtual Machine (JVM), including z/OS. This is referenced as type-4 Connectivity. Type-4 drivers are pure Java and implement the network protocol for a specific data source. The client connects directly to the data source.

The IMS Universal drivers first establish a TCP/IP-based socket connection to IMS Connect on a certain Port. IMS Connect is responsible for routing the request to the IMS databases using the Open Database Manager (ODBM), and sending the response back to the client application.

The IMS Universal drivers support connection pooling with type-4 connectivity, which limits the time that is needed for allocation and deallocation of TCP/IP socket connections. To maximize connection reuse, only the socket attributes of a connection are pooled. These attributes include the IP address and port number that the host IMS Connect is listening on. As a result, the physical socket connection can be reused and additional attributes can be sent on this socket in order to connect to an IMS database.

Figure 5-1 shows how the IMS Universal drivers route communications between your Java client applications running in a distributed environment and an IMS subsystem, using type-4 connectivity.

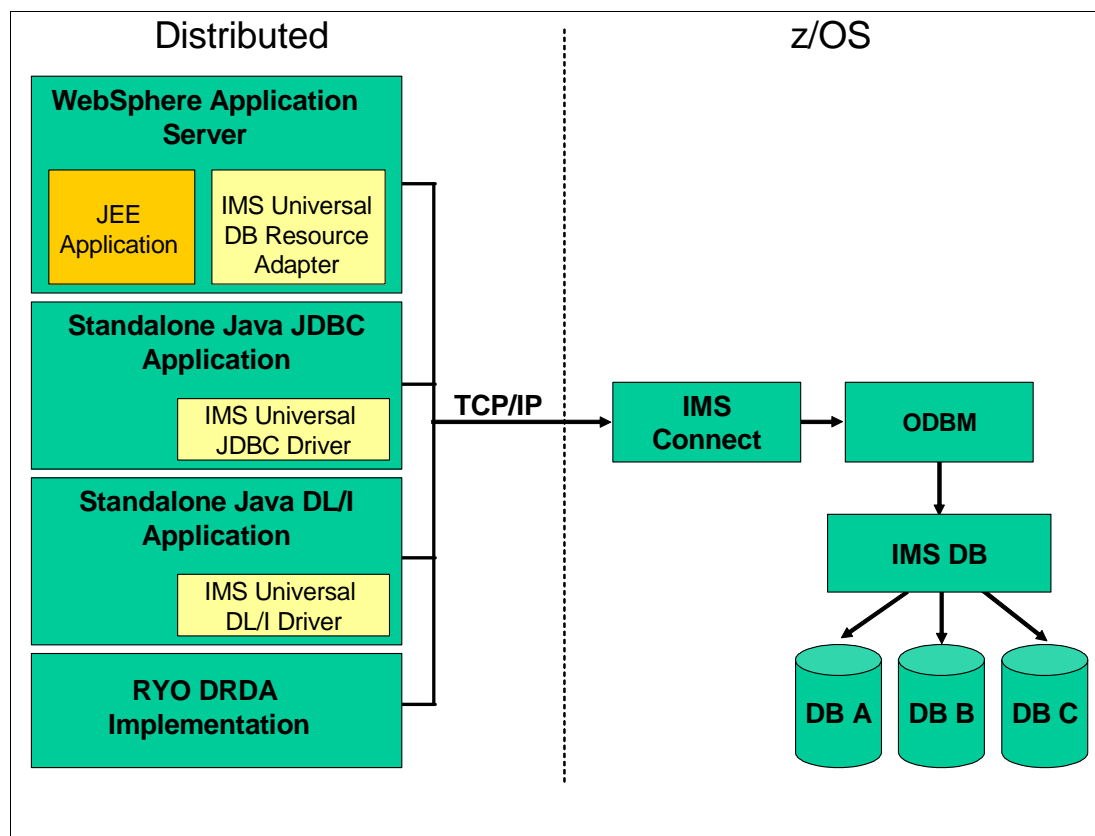


Figure 5-1 Distributed access - Type 4 connectivity

You can also use the IMS Universal drivers with type-4 connectivity if your Java clients are running in a z/OS environment but are located on a separate logical partition from the IMS subsystem. Use type-4 connectivity in a z/OS environment if you want to isolate the application runtime environment from the IMS subsystem environment.

### Local access (type-2 connectivity)

Local Access means that the application with the IMS Universal drivers has to be on the same logical Partition (LPAR) as IMS. This access is referenced as type-2 connectivity. This type of access doesn't need IMS Connect because it communicates directly local instead via TCP/IP.

Type-2 drivers are written partly in the Java programming language and partly in native code. The drivers use a native client library specific to the data source to which they connect. Because of the native code, their portability is limited.

Table 5-1 shows the z/OS runtime environments that support client applications of the IMS Universal DB drivers using type-2 connectivity.

Table 5-1 z/OS runtime environment support for IMS Universal drivers with type-2 connectivity

z/OS runtime environment	Recommended IMS Universal type-2 driver	Used component
WebSphere Application Server for z/OS	► IMS Universal DB resource adapter	ODBA, DRA

<b>z/OS runtime environment</b>	<b>Recommended IMS Universal type-2 driver</b>	<b>Used component</b>
CICS Java Applications	<ul style="list-style-type: none"> <li>► IMS Universal JDBC driver</li> <li>► IMS Universal DL/I driver</li> </ul>	DRA
DB2 for z/OS Stored Procedures in Java	<ul style="list-style-type: none"> <li>► IMS Universal JDBC driver</li> <li>► IMS Universal DL/I driver</li> </ul>	ODBA, DRA
IMS Java dependent regions (JMPs & JBP's)	<ul style="list-style-type: none"> <li>► IMS Universal JDBC driver</li> <li>► IMS Universal DL/I driver</li> </ul>	IMS native access

Each of the Products has already a type-2 interface integrated which will be used by the IMS Universal drivers.

### **RRSLocalOption connectivity type**

In addition to type-4 and type-2 connectivity, the RRSLocalOption connectivity type is supported by the IMS Universal DB resource adapter running on WebSphere Application Server for z/OS. With RRSLocalOption connectivity, applications using the IMS Universal DB resource adapter do not issue commit or rollback calls. Instead, transaction processing is managed by WebSphere Application Server for z/OS.

**Tip:** Set the driverType setting of the connection settings to the value "RRSLocalOption"

### **IMS Universal driver settings**

The settings in Table 5-2 can be used with the IMS Universal drivers depending on your Access type.

*Table 5-2 IMS Universal drivers settings*

<b>Setting and Description</b>	<b>Type-4</b>	<b>Type-2 / RRSLocalOption</b>
<b>driverType</b> Defines the type of access whether TCP/IP or within LPAR	4 or IMSManagedConnectionFactory.DRIVER_TYPE_4	2 / RRSLocalOption
<b>datastoreName</b> Defines to which IMS should be connected	IMS ID (or Alias) configured in the ODBM Configuration or empty for IMSplex	IMS ID specified in the DRA table used for accessing IMS
<b>metadataURL</b> Describes the fully qualified classname of the metadata generated through the DLI Model Utility	class://samples.PSBDATABASE View	class://samples.PSBDATABASE View
<b>datastoreServer</b> Defines the endpoint for the IMS Universal driver	IP or Hostname of IMS Connect	-
<b>portNumber</b> Specifies the ODAccess Port of IMS Connect	5555 (Default)	-
<b>user</b> Specifies the UserID	RACF User ID	-
<b>password</b> Specifies the User password	RACF Password	-

### 5.2.3 Programming approach

There are different programming approaches for the IMS Open Database feature with different capabilities and access methods. The following overview should help you to find the right programming approach for your application's requirements:

#### **JCA/JDBC with SQL**

The JCA/JDBC programming model eases your life in a managed Application Server environment as it uses standard JDBC methods with standard SQL syntax and makes use of the management capabilities of the environment via the JCA. With the XA supported IMS Universal drivers, the Application Server takes care of the two-phase commit handling between different JCA drivers in your application. The big advantage of this approach is that JDBC and SQL is usually very well known by application developers as it is used by very much database drivers. JDBC is the industry standard for database-independent connectivity between the Java programming language and any database that has implemented the JDBC interface. The client uses the interface to query and update data in a database. It is the responsibility of the JDBC driver itself to implement the underlying access protocol for the specific database the driver is implemented for.

#### **JCA/CCI with SQL or DL/I**

The Common Client Interface (CCI) is the other approach for managed Application Server environments and is part of the JCA specifications. The advantage of this approach is that you are able to use SQL syntax and also DL/I commands within one unit of work to get the maximum out of your IMS database and at the same time using the management capabilities of the Application Server. The disadvantage of this approach is, that the CCI model is more difficult to understand and in the handling. Beside of the complexity of the CCI model this approach can help you to realize applications with special requests to your IMS database. Such requests could be for example batch processing of data in sequential hierarchical order and using SQL for the rest of the application.

#### **Standalone/JDBC with SQL**

The Standalone/JDBC programming model is basically the same as the JCA/JDBC programming model because it uses JDBC with SQL. The difference is that is intended to be used in non-managed environments. This means you have to keep track of your two-phase commit in the XA driver by yourself, because there is no higher management instance who manages it. You are also not able to easily predefine standard values in a common way for all your applications, like in it the JCA drivers. But the syntax and supported SQL functions are the same, so that it should be easily expandable to the JCA approach if necessary.

#### **Standalone/ DL/I**

The Standalone/DL/I programming model is for writing applications with DL/I syntax in standalone Java applications. It uses a DL/I programming API which also enables you programming in the same way you usually code non-Java IMS applications with data access. This gives you the ability to leverage all IMS database functions in Java if necessary.

#### **RYO/DRDA**

The Roll-Your-Own (RYO) approach can be used for all cases where you have the requirement to write non-Java applications which have to access IMS databases via TCP/IP directly. The IMS Open Database feature uses DRDA under the cover for communication. So you have to write your own connection socket handling and implement all features you want to have manually. This is not the recommended programming approach because it is the most complex one. You should consider if there are other options for your requirements like a wrapper or middle-layer possible which uses one of the other programming models.

For more informations about the possible DRDA commands see the Chapter 14 'DRDA DDM command architecture' of *IMS Version 11 Application Programming APIs*, SC19-2429

## 5.2.4 Comparison of the IMS Universal drivers

Depending on your IT infrastructure, solution architecture, and application design, choose the IMS Universal drivers programming approach that is best for your development scenario.

Table 5-3 lists the recommended IMS Universal drivers programming approach to use, based on the application programmer's choice of application platform, data access method, and transaction processing option.

Table 5-3 Comparison of IMS Universal driver approaches

Target Runtime	Programming approach	Transactionality support	Recommended IMS Universal driver
WebSphere Application Server	JCA/JDBC with SQL	XA and Local-like support	imsudbJXA.rar
		Local support only	imsudbJLocal.rar
	CCI with SQL or DL/I	XA and Local-like support	imsudbXA.rar
		Local support only	imsudbLocal.rar
Standalone Java Application	JDBC with SQL	XA (self managed) and local support	imsudb.jar
	DL/I API with DL/I	XA (self managed) and local support	imsudb.jar
Non-Java Application	DRDA with DDM	XA (self implemented) possible	-

**Restriction:** The XA support is only available with type-4 connectivity.

**Note:** The IMS Universal Database resource adapters have normally the file extension rar for Resource Adapter Archive. The IMS Universal Standalone drivers are Java jar files.

The IMS Universal DB resource adapters with XA support have XA support as well as local-like transaction support. Which one is used is decided by the transaction settings of the application, whether you use container-managed transactions or bean-managed transactions.

The IMS Universal JDBC and the IMS Universal DL/I driver for the Standalone Java applications have XA support and local support implemented. But as there is no higher instance who controls the 2 Phase Commit execution, you have to implement them in your application by yourself if you want to use the XA support.

For Non-Java applications you can create your own implementations by using the DRDA standard and sending DDM commands to IMS Connect. The application programmer is responsible for implementing the two-phase commit procedure for XA support.

## 5.3 IMS Universal Database resource adapter

The IMS Universal DB resource adapter is intended to be used in managed Java Enterprise Edition Application Servers (JEE Servers). It makes use of the Java Connector Architecture (JCA) to integrate easily into standard conform servers. JCA offers many advantages for the programmer as it provides globally managed services as security management, connection pooling and transaction management via system contracts without additional coding by the application programmer.

JCA is a Java-based technology solution for connecting application servers and enterprise information systems (EIS) as part of enterprise application integration (EAI) solutions. While JDBC is specifically used to connect Java EE applications to databases, JCA is a more generic architecture for connection to non-standard systems (including databases). JCA offers a standard interface between the J2EE application server and any EIS via a JCA resource adapter.

A resource adapter is a J2EE component that implements the J2EE Connector architecture for a specific EIS. It is through a resource adapter that a J2EE application communicates with the EIS itself. There are two types of contracts (interfaces) implemented by a resource adapter – the application contract and the system contract. The application contract defines the API through which a J2EE component such as an Enterprise bean accesses the EIS. This API is the only view an application has of the EIS. The system contracts and the resource adapter implementation are transparent to the application component. The EIS specific communication is handled by the resource adapter implementation itself. The system contracts define the interfaces that link the resource adapter to the services managed by the J2EE server itself. These services include connection, transaction, and security services.

The IMS Universal DB resource adapters with XA support are intended to be used in two-phase commit transaction processing but you can also use them in single-phase commit applications. The IMS Universal DB resource adapters with local transaction only support provides only single-phase commit functionality.

With the IMS Universal DB resource adapters you can group interactions in applications together to make sure that all interactions are committed or rolled back. This can be done using container-managed or bean-managed transaction demarcation.

In container-managed transactions, all work performed in an EJB method invocation is part of one unit of work, and no explicit demarcation by the application is required. Transactional integrity is managed by the Java EE application server.

Use a bean-managed EJB if you need to have multiple units of works within the same EJB method invocation. In bean-managed transactions, you must use the `javax.resource.cci.LocalTransaction` or `javax.transaction.UserTransaction` interface to programmatically demarcate units of work explicitly.

- ▶ If you use the `LocalTransaction` interface you can only group work performed through the used resource adapter
- ▶ Using the `UserTransaction` interface allows all transactional resources within the application to be grouped.

With the IMS Universal Database resource adapters you have the choice of two different programming styles: The Common Client Interface style and the JCA/JDBC style

Table 5-4 gives a brief overview about the major differences

Table 5-4 Comparison of JCA Models - CCI and JDBC

Attribute / Function	JDBC	CCI
Data Access Language	SQL	SQL and DL/I
Main Java classes	java.sql	javax.resource.cci
Connection creation	DataSource or DriverManager	ConnectionFactory with ConnectionSpec
Command creation	Connection	Connection
Execution commands	Statement	Interaction with SQLInteractionSpec or DLInteractionSpec
Result	ResultSet	RecordFactory and Records or ResultSet

### 5.3.1 JCA/Common Client Interface approach

The JCA specification defines a programming interface called the Common Client Interface (CCI). This interface is used to communicate with any Enterprise Information System. The IMS Universal DB resource adapter implements the CCI for interactions with IMS databases. The CCI interfaces for the IMS Universal DB resource adapter are in the `com.ibm.ims.db.cci` package. The CCI implementation provided by IMS allows applications to make either SQL or DL/I calls to access the IMS database.

The following two drivers are available for the IMS Universal DB resource adapter for CCI

- ▶ `imsudbXA.rar` with XA and local-like transaction support
- ▶ `imsudbLocal.rar` with local transaction support only

**Restriction:** XA transaction support is only available with type-4 connectivity.

**Note:** Local transaction with the XA driver means Single Phase Commit when possible but it still requires RRS.

#### Example for CCI application with SQL calls

Example 5-2 shows an EJB using the CCI programming model. It gets a connection through using JNDI to lookup the Connection Factory in the JEE Server and it uses SQL with the `SQLInteractionSpec` class to get a result from the Car Dealer IVP Database.

Example 5-2 CCI with SQL calls

```
import java.sql.SQLException;
import javax.naming.InitialContext;
import javax.resource.ResourceException;
import javax.resource.cci.Connection;
import javax.resource.cci.ConnectionFactory;
import javax.resource.cci.Interaction;
import javax.resource.cci.ResultSet;
import javax.transaction.UserTransaction;
import com.ibm.ims.db.cci.SQLInteractionSpec;

public class BeanManagedCCISQLBean implements javax.ejb.SessionBean{
```



```

private javax.ejb.SessionContext mySessionCtx;
public void execute() throws Exception {
    InitialContext ic = new InitialContext();
    ConnectionFactory cf = (ConnectionFactory) ic.lookup("java:comp/env/MyMCF");
    Connection conn = null;
    UserTransaction ut = null;
    try {
        ut = this.mySessionCtx.getUserTransaction();
        ut.begin();
        conn = cf.getConnection();
        Interaction ix = conn.createInteraction();
        SQLInteractionSpec iSpec = new SQLInteractionSpec();
        iSpec.setSQL("SELECT * FROM AUTOLPCB.DEALER WHERE ZIP='12345-6789'");
        ResultSet rs = (ResultSet) ix.execute(iSpec, null);
        while (rs.next()) {
            System.out.println(rs.getString("DLRNAME"));
        }
        rs.close();
        ix.close();
        ut.commit();
        conn.close();
    } catch (ResourceException e) {
        ut.rollback();
        conn.close();
    } catch (SQLException e) {
        ut.rollback();
        conn.close();
    }
}

public javax.ejb.SessionContext getSessionContext() {
    return mySessionCtx;
}

public void setSessionContext(javax.ejb.SessionContext ctx) {
    mySessionCtx = ctx;
}

public void ejbCreate() throws javax.ejb.CreateException {}
public void ejbActivate() {}
public void ejbPassivate() {}
public void ejbRemove() {}
}

```

---

### Example for CCI application with DL/I calls

The coding in Example 5-3 shows you an EJB using the CCI programming model. It gets a connection through using JNDI to lookup the Connection Factory in the JEE Server and it uses DL/I calls with the `DLIInteractionSpec` class to get a result from the Car Dealer IVP Database.

#### *Example 5-3 CCI with DL/I calls*

---

```

import java.sql.SQLException;
import javax.naming.InitialContext;
import javax.resource.ResourceException;
import javax.resource.cci.Connection;
import javax.resource.cci.ConnectionFactory;
import javax.resource.cci.Interaction;
import javax.resource.cci.MappedRecord;
import javax.resource.cci.RecordFactory;
import javax.resource.cci.ResultSet;
import javax.transaction.UserTransaction;

```



```

import com.ibm.ims.db.cci.DLIInteractionSpec;

public class BeanManagedCCIDLIBean implements javax.ejb.SessionBean{
    private javax.ejb.SessionContext mySessionCtx;
    public void execute() throws Exception {
        InitialContext ic = new InitialContext();
        ConnectionFactory cf = (ConnectionFactory) ic.lookup("java:comp/env/MyMCF");
        Connection conn = null;
        UserTransaction ut = null;
        try {
            ut = this.mySessionCtx.getUserTransaction();
            ut.begin();
            conn = cf.getConnection();
            Interaction ix = conn.createInteraction();
            DLIInteractionSpec iSpec = new DLIInteractionSpec();
            iSpec.setFunctionName("RETRIEVE");
            iSpec.setPCBName("AUTOLPCB");
            iSpec.setSSAList("DEALER (DLRNO = '1235')");
            RecordFactory rf = cf.getRecordFactory();
            MappedRecord input = rf.createMappedRecord("DEALER");
            input.put("DLRNAME", null);
            input.put("ZIP", null);
            ResultSet results = (ResultSet) ix.execute(iSpec, input);
            while (results.next()) {
                System.out.println(results.getString("DLRNAME"));
                System.out.println(results.getString("ZIP"));
            }
            results.close();
            ix.close();
            ut.commit();
            conn.close();
        } catch (ResourceException e) {
            ut.rollback();
            conn.close();
        } catch (SQLException e) {
            ut.rollback();
            conn.close();
        }
    }
    public javax.ejb.SessionContext getSessionContext() {
        return mySessionCtx;
    }
    public void setSessionContext(javax.ejb.SessionContext ctx) {
        mySessionCtx = ctx;
    }
    public void ejbCreate() throws javax.ejb.CreateException {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void ejbRemove() {}
}

```

---

### 5.3.2 JCA/JDBC approach

In addition to the CCI interface provided by the IMS Universal DB resource adapter, you can also write JDBC applications to access your IMS data from a managed environment, while leveraging the Java EE services provided by the application server. This capability is provided by the IMS Universal JCA/JDBC driver version of the IMS Universal DB resource adapter.

The IMS Universal JCA/JDBC driver is based on the Java Platform, Enterprise Edition (Java EE) Connector Architecture (JCA) 1.5 and Java Database Connectivity (JDBC) 3.0 standard.

The following two drivers are available for the IMS Universal JCA adapter with JDBC

- ▶ `imsudbjXA.rar` with XA and local-like transaction support
- ▶ `imsudbjLocal.rar` with local transaction support only

**Restriction:** XA transaction support is only available with type-4 connectivity.

**Note:** Local transaction with the XA driver means Single Phase Commit when possible but it still requires RRS

### Example for JCA/JDBC application with SQL calls

Example 5-4 shows you an EJB using the JDBC programming model for the IMS Universal DB resource adapter with JDBC. It gets a DataSource Object through using JNDI to lookup the Data Source in the JEE Server and it uses SQL calls through the JDBC API to get a result from the Car Dealer IVP Database.

*Example 5-4 JCA/JDBC with SQL calls*

---

```
import java.sql.SQLException;
import javax.naming.InitialContext;
import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.Statement;
import java.sql.ResultSet;
import javax.transaction.UserTransaction;

public class BeanManagedJDBCSQLBean {
    private javax.ejb.SessionContext mySessionCtx;
    public void execute() throws Exception {
        InitialContext ic = new InitialContext();
        DataSource ds = (DataSource) ic.lookup("java:comp/env/MyMCF");
        Connection conn = null;
        UserTransaction ut = null;
        try {
            ut = this.mySessionCtx.getUserTransaction();
            ut.begin();
            conn = ds.getConnection();
            Statement st = conn.createStatement();
            ResultSet rs = st.executeQuery("SELECT DLRNAME, ZIP FROM AUTOLPCB.DEALER");
            while (rs.next()) {
                System.out.println(rs.getString("DLRNAME"));
                System.out.println(rs.getString("ZIP"));
            }
            rs.close();
            ut.commit();
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
            ut.rollback();
            conn.close();
        }
    }

    public javax.ejb.SessionContext getSessionContext() {
        return mySessionCtx;
    }
}
```

```
    }  
    public void setSessionContext(javax.ejb.SessionContext ctx) {  
        mySessionCtx = ctx;  
    }  
    public void ejbCreate() throws javax.ejb.CreateException {}  
    public void ejbActivate() {}  
    public void ejbPassivate() {}  
    public void ejbRemove() {}  
}
```

---

## 5.4 IMS Universal JDBC driver (Stand Alone)

The IMS Universal JDBC driver is intended to be used in Stand Alone Java Applications on the client side. It provides the capability of issuing SQL calls to IMS databases. The IMS Universal JDBC driver is based on the JDBC 3.0 standard.

JDBC is an application programming interface (API) that Java applications use to access relational databases or tabular data sources. The JDBC API is the industry standard for database-independent connectivity between the Java programming language and any database that has implemented the JDBC interface. The client uses the interface to query and update data in a database. It is the responsibility of the JDBC driver itself to implement the underlying (specific) access protocol for the specific database the driver is implemented for. Drivers convert requests from Java programs to a protocol that the database management system (DBMS) can understand.

IMS support for JDBC lets you write Java applications that can issue dynamic SQL calls to access IMS data and process the result set that is returned in tabular format. The IMS Universal JDBC driver is designed to support a subset of the SQL syntax with functionality that is limited to what the IMS database management system can process natively. Its DBMS-centric design allows the IMS Universal JDBC driver to fully leverage the high performance capabilities of IMS. The IMS Universal JDBC driver also provides aggregate function support, and ORDER BY and GROUP BY support.

There are three ways to establish a connection to an IMS with the IMS Universal JDBC driver by using

- ▶ JDBC DataSource Interface and providing the necessary values directly
- ▶ JDBC DataSource Interface and using JNDI to lookup the Connection
- ▶ JDBC DriverManager Interface

### 5.4.1 Connecting to an IMS database using the JDBC DataSource interface

JDBC versions starting with version 2.0 provide the DataSource interface for connecting to a data source. Using the DataSource interface is the preferred way to connect to IMS from your IMS Universal JDBC driver application. This model has the advantage that it allows you to keep your application dynamically by using JNDI to look up the Connection or specifying it manually by using the setter methods belonging to the specific DataSource.

#### Application managed connection parameters

You can provide all necessary information to access the IMS database directly in your application. The code in Example 5-5 shows how to do this using the JDBC DataSource Interface.

*Example 5-5 JDBC DataSource Connection with Application Managed approach*


---

```

import java.sql.*;
import com.ibm.ims.jdbc.*;

public class DataSourceAppManagedApp {
    public static void main(String[] args) {
        Connection conn = null;
        // Create an instance of DataSource
        IMSDataSource ds = new com.ibm.ims.jdbc.IMSDataSource();
        // Set the URL of the fully qualified name of the Java metadata class
        ds.setMetadataURL("class://samples.ims.openDb.AUTPSB11DatabaseView");
        // Set the data store name
        ds.setDatastoreName("IMS2");
        // Set the data store server
        ds.setDatastoreServer("myhost.itso.ibm.com");
        // Set the port number
        ds.setPortNumber(5555);
        // Set the JDBC connectivity driver type
        ds.setdriverType(IMSDatasource.DRIVER_TYPE_4);
        // Disable SSL for connection
        ds.setSSLConnection(false);
        // Set timeout for connection
        ds.setLoginTimeout(10);
        // Set user ID for connection
        ds.setUser("IMSUSR");
        // Set password for connection
        ds.setPassword("myPW");
        // Create JDBC connection
        try {
            conn = ds.getConnection();
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```

---

**JNDI managed connection parameters**

The better alternative is to specify the values once in Java Naming and Directory Interface (JNDI) and create a connection through obtain the object from the there. You can anyway override values like username, password or the metadata classes. Through specifying the MetadataURL in your application you can define only one JNDI connection per IMS and define the used database metadata in you application. The code in Example 5-6 shows you how to do this by using the JDBC DataSource Interface and JNDI in a managed environment.

*Example 5-6 JDBC DataSource Connection with JNDI Managed approach*


---

```

import java.sql.*;
import com.ibm.ims.jdbc.*;
import javax.naming.*;

public class DataSourceJNDIManagedApp {
    public static void main(String[] args) {
        try {
            Context ctx = new InitialContext();
            IMSDataSource ds = (IMSDataSource)ctx.lookup("jdbc/imsopendb");
            // Optional Overrides
            ds.setMetadataURL("class://samples.ims.openDb.AUTPSB11DatabaseView");
        }
    }
}

```

---

```

        ds.setUser("IMSUSR");
        ds.setPassword("myPW");
        // Create Connection
        Connection conn = ds.getConnection();
        conn.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

---

## 5.4.2 Connecting to an IMS database using the JDBC DriverManager interface

A JDBC application can also establish a connection to a data source using the JDBC DriverManager interface, which is part of the java.sql package. See Example 5-7.

### *Example 5-7 Connecting with the JDBC DriverManager Interface*

---

```

import java.sql.*;
import java.util.Properties;

public class DriverManagerJDBCApp {
    public static void main(String[] args) {
        try {
            Connection conn = null;
            // Create Properties object
            Properties props = new Properties();
            // Disable SSL for connection
            props.put("sslConnection", "false");
            // Set driverstoreName for connection
            props.put( "driverstoreName", "IMS2" );
            // Set timeout for connection
            props.put("loginTimeout", "10");
            // Set user ID for connection
            props.put( "user", "IMSUSR" );
            // Set password for connection
            props.put( "password", "myPW" );
            // Set URL for the data source
            Class.forName("com.ibm.ims.jdbc.IMSDriver");
            // Create connection
            conn = DriverManager.getConnection("jdbc:ims://myhost.ibm.com:5555/" +
                "class://samples.ims.openDb.AUTPSB11DatabaseView",props);
            // Close Connection
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

---

Instead of using the Java.Util.Properties you can also provide the properties directly in the DriverManager.getConnection(url) function. Here is an example of this approach:

```

conn = DriverManager.getConnection("jdbc:ims://myhost.ibm.com:5555/" +
    "class://samples.ims.openDb.AUTPSB11DatabaseView:datastoreName=IMS2;" +
    "loginTimeout=10;sslConnection=false;user=IMSUSR;password=myPW;");

```

The necessary values are specified directly in the DriverManagerURL in this case. The disadvantage of using this is that your application defines a static connection. When you change for example the IP Address or Port you have to recompile your application. The DriverManager Interface is used very often in Tools to create a Connection. The syntax of the DriverManagerURL is

```
jdbc:ims://<IP>:<PORT>/class://<Package.PSBDatabaseView>:<Parameter1>=<Value1>;  
<Parameter2>=<Value2>;
```

## 5.5 IMS Universal DL/I driver

Use the IMS Universal DL/I driver when you need to write granular queries to access IMS databases directly from a Java client in a non-managed environment.

The IMS databases are hierarchically organized instead of relational. Because of this, the JDBC API cannot give you the full capability of all IMS functions. So sometimes it can be useful to use the natural way of accessing IMS resources, which is Data Language /I (DL/I). The IMS Universal DL/I driver is closely related to the traditional IMS DL/I database call interface that is used with other programming languages for writing applications in IMS.

By using the IMS Universal DL/I driver, you can build segment search arguments (SSAs) and use the methods of the program communication block (PCB) object to read, insert, update, delete, or perform batch operations on segments. You can gain full navigation control in the segment hierarchy.

### 5.5.1 Basic steps in writing a IMS Universal DL/I driver application

If you are familiar with DL/I processing, then the usage of the DL/I API is very similar. It tries to give you the possibilities to work with the traditional programming model in a new environment like Java. This book concentrates on leveraging industry standards which in principle is JDBC and SQL programming. To explain DL/I in detail would be too much for the purpose of this book. Therefore the following steps explain briefly the general steps of writing an application program with the IMS Universal DL/I driver:

1. Import the `com.ibm.ims.dli` and `com.ibm.ims.base` packages that contain the IMS Universal DL/I driver classes, interfaces, methods and exceptions.
2. Create an `IMSConnectionSpec` instance by calling the `create IMSConnectionSpec` method in the `IMSConnectionSpecFactory` class.
3. Set the connection properties for the `IMSConnectionSpec` instance.
4. Obtain a program specification block (PSB), which contains one or more PCBs by passing the connection request properties to the `PSBFactory` class to create the PSB instance. When the PSB instance is created successfully, a connection is established to the database.
5. Obtain a PCB handle, which defines an application's view of an IMS database and provides the ability to issue database calls to retrieve, insert, update, and delete database information.
6. Obtain an unqualified segment search argument list (SSAList) of one or more segments in the database hierarchy.
7. Add qualification statements to specify the segments targeted by DL/I calls.
8. If retrieving data, mark the segment fields to be returned.
9. Execute DL/I calls to the IMS database.

10. Handle errors that are returned from the DL/I programming interface.

11. Disconnect from the IMS database subsystem.

For more Informations about DL/I programming with the IMS Universal DL/I driver see Chapter 36 “Programming with the IMS Universal drivers” of *IMS V11 Application Programming Guide*, SC19-2428 and *IMS Version 11 Application Programming APIs*, SC19-2429.

### 5.5.2 Example code using IMS Universal DL/I driver

For a detailed explanations and code samples using the IMS Universal DL/I driver see Chapter 8, “Scenario 3 - Writing DL/I and mixed applications” on page 179.

## 5.6 SQL syntax for the IMS Universal drivers

SQL stands for structured query language and is used to query and manipulate data in relational database management systems. IMS Databases are hierarchically organized instead of relational. This organization form has a lot of advantages for many application areas like XML processing or sequential processing of data usually called as batch processing.

The data access language of IMS for the hierarchical databases is traditionally DL/I. But today SQL is de-facto industry standard for data access language in applications. But this isn't a problem for IMS as it can take advantage of both models relational and hierarchical. It can provide a relational view of your hierarchical IMS Databases and make it accessible via SQL and you can use traditional programming languages for hierarchical database access in new environments. At the same time you do not have to change your existing DL/I using applications.

The Java database metadata class contains information about the IMS database, including segments, segment names, the segment hierarchy, fields, field types, field names, fields offsets, and field lengths. The metadata is used by the IMS Universal JDBC drivers to allocate program specification blocks (PSBs), issue DL/I calls, perform data transformation, and translate SQL queries to DL/I calls.

Table 5-5 shows the mapping between hierarchical database terms and relational database terms.

*Table 5-5 Mapping between IMS terms and relational terms*

Hierarchical IMS DB term	Relational DB equivalent
Segment name	Table name
Segment instance	Table row
Segment field name	Column name
Segment unique key	Table primary key
Virtual foreign key field	Table foreign key
PCB	Database View
PSB	Collection of Database View

The following sections give an overview about the most important SQL statements and how to use them with the IMS Universal drivers supporting JDBC:

### 5.6.1 SQL keywords

If you use a SQL keyword as a name for a PCB, segment, or field, your JDBC application program will return an error when it attempts an SQL query. The keywords are not case-sensitive. The SQL keywords are listed in Table 5-6.

Table 5-6 SQL keywords

ALL	DISTINCT	SELECT
AND	FROM	SET
AS	GROUP BY	SUM
ASC	INSERT	UPDATE
AVG	MAX	VALUES
COUNT	MIN	WHERE
DELETE	OR	
DESC	ORDER BY	

In addition to the supported SQL keywords, there are several more keywords known by the SQL syntax. These are restricted keywords by the IMS Universal drivers as they might be used in future releases of the IMS Universal drivers. Therefore is our recommendation not to use these keywords as column names.

**Note:** The IMS Enterprise Suite DLIModel Utility should detect if these keywords are used as a field or segment name and should rename them. In our Car Dealer Database this is for example the case for the COUNT field in the MODEL Segment which is renamed to COUNT1

Table 5-7 lists the restricted SQL keywords.



Table 5-7 Restricted AQL keywords

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

## 5.6.2 Primary key and virtual foreign key handling

In relational databases the relations are build by using foreign key relationships between tables. In IMS the relations are part of the hierarchy itself. The IMS Universal JDBC driver introduces the concept of virtual foreign keys to capture these explicit hierarchies in a relational sense.

Figure 5-2 shows an extract of the Car Dealer IVP Database Diagram created by the IMS Enterprise Suite DLIModel Utiltiy using the AUTOLPCB.

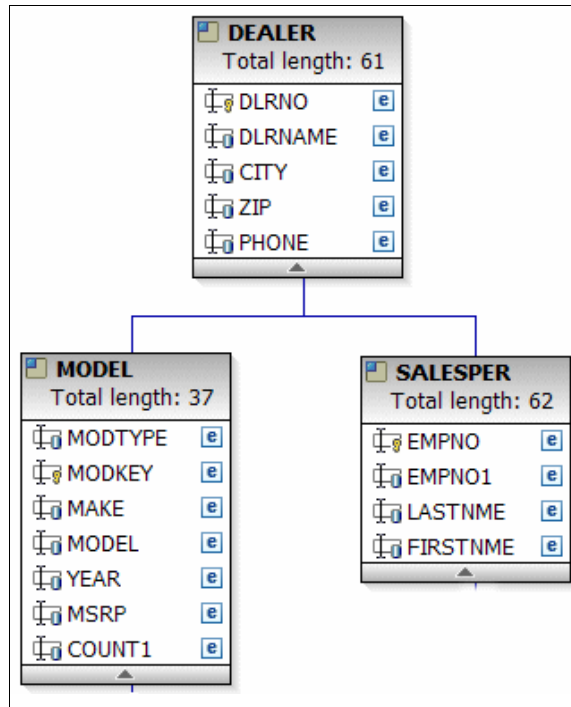


Figure 5-2 Extract from overview diagram from Car Dealer IVP Example

The DEALER segment is the root segment in this view and has two child segments MODEL and SALESPER. The DLRNO field is the primary key for the DEALER segment. As well as MODKEY is the primary key in the MODEL segment.

Every table that is not the root table in a hierarchic path will virtually contain the unique keys of all of its parent segments up to the root of the database. These keys are called virtual foreign key fields.

The IMS Universal JDBC drivers automatically generates a Virtual Foreign Key Column based on the Segment Names. The following SELECT query shows an example.

```
SELECT * FROM AUTOLPCB.MODEL
```

This statement returns all columns from the MODEL segment which are physically stored in the database as well as one more column called DEALER\_DLRNO which contains the reference to the parent's segment primary key field. This field is not stored physically in the database. Figure 5-3 shows the result of this query.

DEALER_DLRNO ▼	MODTYPE	MODKEY	MAKE	MODEL	YEAR	MSRP	COUNT1
1234	S	FORD FO...	FORD	FOCUS	2002	17995	03
1235	S	Volvo S40 ...	Volvo	S40	2002	21000	01
1236	S	TRABANT S...	TRABANT	SUPERIOR	1988		03

Figure 5-3 Query result from MODEL table

The purpose of the virtual foreign key fields is to maintain referential integrity, similar to foreign keys in relational databases. This allows SQL SELECT, INSERT, UPDATE, and DELETE queries to be written against specific tables and columns located in a hierarchic path.

### 5.6.3 Usage of SELECT statement

The SELECT statement is used to retrieve data from one or more tables. The result is returned in a tabular result set.

When using the SELECT statement with the IMS Universal JDBC driver:

- ▶ If you are selecting from multiple tables and the same column name exists in one or more of these tables, you must table-qualify the column or an ambiguity error will occur.
- ▶ The FROM clause must list all the tables you are selecting data from. The tables listed in the FROM clause must be in the same hierarchic path in the IMS database.
- ▶ In Java applications using the IMS JDBC drivers, connections are made to PSBs. Because there are multiple database PCBs in a PSB, queries must specify which PCB in a PSB to use. 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.

Here are some examples of valid SELECT statements against the Car Dealer IVP Database:

```
SELECT * FROM AUTOLPCB.DEALER
```

Returns all rows and all columns from the DEALER Segment in the AUTOLPCB.

```
SELECT DLRNAME FROM AUTOLPCB.DEALER WHERE PHONE=6667777
```

Returns the DLRNAME column from all rows where the PHONE equals the search string from the DEALER Segment in the AUTOLPCB.

```
SELECT DISTINCT MODEL FROM AUTOLPCB.MODEL
```

Returns the MODEL column from all distinct rows from the MODEL Segment in the AUTOLPCB.

```
SELECT ZIP,CITY,DLRNAME AS NAME FROM AUTOLPCB.DEALER ORDER BY ZIP
```

Returns all rows from ZIP, CITY and DLRNAME column. The DLRNAME column is referenced as NAME. The Resultset is ordered by ZIP ascendingly by default.

```
SELECT MAKE AS BRAND,MODEL FROM AUTOLPCB.MODEL GROUP BY BRAND,MODEL ORDER BY MAKE
```

Returns all rows from the columns MAKE and MODEL grouped by MAKE (AS BRAND) and MODEL. It is also ordered by the column MAKE.

```
SELECT MAX(YEAR) FROM AUTOLPCB.MODEL
```

Returns the highest YEAR from the MODEL column. The column name is a combination of the aggregate function name and the field name separated by an underscore character (\_). In this case you would get the result by using resultSet.getInt("MAX\_YEAR"). If the aggregate function argument field is table-qualified, the ResultSet column name is the combination of the aggregate function name, the table name, and the column name, separated by underscore characters (\_). For example, SELECT MAX(Model.year) results in a column name MAX\_Model\_year.

```
SELECT * FROM AUTOLPCB.DEALER, AUTOLPCB.MODEL WHERE MODEL.DEALER_DLRNO = DEALER.DLRNO
```

This select statement returns data from the DEALER root segment as well as from the MODEL segment. It uses the virtual foreign key DEALER\_DLRNO field in the DEALER table.

So it will give you all entries from the MODEL database and the referred parent entries in the DEALER table.

### 5.6.4 Usage of INSERT statement

The INSERT statement is used to insert new rows into a table.

The following sample SQL shows the insert of a record at root level:

```
INSERT INTO AUTOLPCB.DEALER (DLRNO, ZIP, DLRNAME, CITY, PHONE) VALUES ('8888',  
'71139', 'Thilo', 'Stuttgart', '555-888')
```

When inserting a record in a table at a non-root level, you must specify values for all the virtual foreign key fields of the table:

```
INSERT INTO AUTOLPCB.MODEL (DEALER_DLRNO, MODTYPE, MAKE, MODEL, YEAR, MSRP,  
COUNT1) VALUES ('8888', 'S', 'LIDLA', 'SuperABC', '2010', '66000', '05')
```

This insert statement creates a MODEL record which refers to the DEALER segment with the primary key DLRNO=8888.

**Attention:** The MODKEY field should not be inserted because it is a primary key field and is automatically build by combining the MAKE, MODEL and YEAR values. Alternatively you can insert the MODKEY in the proper format and therefore do not specify the MAKE, MODEL and YEAR.

### 5.6.5 Usage of UPDATE statement

The UPDATE statement is used to update existing records in a table.

The following sample SQL shows an update in a dependent table:

```
UPDATE AUTOLPCB.MODEL SET MSRP='50000' WHERE DEALER_DLRNO = '8888' AND MSRP >=  
'60000'
```

This UPDATE statement sets the MSRP value to 50000 for all MODEL records that refer to DEALER with the DLRNO = 8888 and where the MSRP value is higher or equal to 60000.

**Restriction:** Updates on a virtual foreign key field will fail, because they doesn't exist in the database. To change a dependent segment you have to delete the segment and reinsert it referring to the segment you want to put it under.

### 5.6.6 Usage of DELETE statement

The DELETE statement is used to delete rows in a table. DELETE operations are cascaded to all child segments.

The following sample SQL shows the usage of the DELETE statement.

Deleting the root Segment DEALER without a where clause, as in the following statement, deletes the whole database with all its records in it:

```
DELETE FROM AUTOLPCB.DEALER
```

The following statement deletes all entries in the MODEL table and referenced segments under it in the whole database.

```
DELETE FROM AUTOLPCB.MODEL
```

The following SQL statement deletes the root record of DLRNO= 8888 and all referenced children segments connected to it.

```
DELETE FROM AUTOLPCB.DEALER WHERE DLRNO = '8888'
```

The following SQL statement deletes the MODEL segments and all referenced segments under it for the DEALER with DLRNO= 8888:

```
DELETE FROM AUTOLPCB.MODEL WHERE DEALER_DLRNO = '8888'
```

### 5.6.7 Usage of the WHERE statement

The WHERE statement can be used in combination with the SELECT, UPDATE and DELETE statements. It is used to specify an exact record or filter results in a query.

You can use the following operators in the WHERE clause to select data conditionally:

=	Equals
!=	Not equal
>	Greater than
>=	Greater than or equals
<	Less than
<=	Less than or equals

**Note:** Our recommendation is to compare columns to values and not to other columns. This is better for the performance of the queries and helps to reduce errors in your SQL statements where you get the wrong results back. However, it is legal to compare the virtual foreign key column with another primary key column.

The WHERE statement has the following usage rules:

- ▶ Do not use parentheses. Qualification statements are evaluated from left to right. The order of evaluation for operators is the IMS evaluation order for segment search arguments.
- ▶ List all qualification statements for a table adjacently. For example, in the following valid WHERE clause, the qualified columns from the same DEALER table are listed adjacently:

```
SELECT * FROM AUTOLPCB.DEALER, AUTOLPCB.MODEL WHERE AUTOLPCB.DEALER.ZIP = '88888' OR AUTOLPCB.DEALER.ZIP = '88888' AND AUTOLPCB.MODEL.MODEL = 'B Plus'
```

The following statement would be an invalid WHERE clause as they are not grouped together:

```
SELECT * FROM AUTOLPCB.DEALER, AUTOLPCB.MODEL WHERE AUTOLPCB.DEALER.ZIP = '88888' AND AUTOLPCB.MODEL.MODEL = 'B Plus' OR AUTOLPCB.DEALER.ZIP = '88888'
```

- ▶ The OR operator can be used only between qualification statements that contain columns from the same table. To combine qualification statements for different tables, use an AND operator.
- ▶ The columns in the WHERE clause must be DBD-defined fields. The only exception to this is when the columns in the WHERE clause are subfields that make up a DBD-defined field.

**Note:** The columns that are in the DBD are marked in the DLIModel IMS Java report as being either primary key fields or search fields.

For example, a DBD-defined field is named ADDRESS and is 30 bytes long. In a COBOL copybook, this field is broken down into CITY, STATE, and ZIP subfields as illustrated by the following code:

```
01 ADDRESS
  02 CITY PIC X(10)
  02 STATE PIC X(10)
  02 ZIP PIC X(10)
```

Without the subfield support, the ADDRESS value in the WHERE clause would have to be padded manually, and entered like this:

```
WHERE ADDRESS = 'Stuttgart GER          70565      '
```

With the subfield support, you can enter the WHERE clause like this:

```
WHERE CITY = 'Stuttgart' AND STATE = 'GER' AND ZIP = '70565'
```

The following rules apply for subfields:

- Subfields must always be fully defined, as it is only possible to search for the whole DBD-defined search-field.
- The columns in the WHERE clauses are not allowed to be separated and must be combined by an AND operator
- The EQUAL (=) is the only allowed operator on subfields.

## 5.6.8 Usage of AGGREGATE functions

The following aggregate functions are supported by the IMS Universal drivers:

- AVG
- COUNT
- MAX
- MIN
- SUM

The following supported keywords can be also used to aggregate the results or influence the above aggregate functions

- AS
- DISTINCT
- GROUP BY
- ORDER BY ASC/DESC

The ResultSet column name from an aggregate function is a combination of the aggregate function name and the field name separated by an underscore character (\_). The examples in Table 5-8 show the query and the example method of getting the column results from the ResultSet.

Table 5-8 Aggregate functions examples

SQL Select statement	ResultSet column name
SELECT MAX(year)	rs.getInt("MAX_year")
SELECT MAX(MODEL.year)	rs.getLong("MAX_MODEL_year")
SELECT MAX(MODEL.year) AS 'oldest'	rs.getInt("oldest")

SQL Select statement	ResultSet column name
SELECT COUNT(DEALER.DLRno)	rs.getInt("COUNT_DEALER_DLRno")
SELECT COUNT(DISTINCT DEALER.DLRNAME)	rs.getInt("COUNT_DISTINCT_DEALER_DLRNAME")
SELECT COUNT(DISTINCT AUTOLPCB.DEALER.DLRNAME)	rs.getInt("COUNT_DISTINCT_DEALER_DLRNAME")

**Note:** The schema name (PCB name) is not added to the column name.

**Attention:** Only the COUNT aggregate function allows the DISTINCT keyword in it.

Table 5-9 shows the aggregate functions allowed arguments and the corresponding result types.

Table 5-9 Aggregate Functions and result types

Aggregate Function	Argument Type	Result Type
SUM and AVG	Byte	Long
	Short	Long
	Integer	Long
	Long	Long
	BigDecimal	BigDecimal
	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

A type conversion can be done by using the `ResultSet.getXXX()` function, as long as it is valid conversion.

## 5.7 Data transformation support

The IMS Universal drivers allow you to transform easily data from a in the metadata defined format to an other supported Java format. This is done by using the `get<type>` function of the `ResultSet` in JDBC applications or by using these functions on the path object in DL/I applications.

### 5.7.1 JDBC data types to Java data types mapping

Table 5-10 shows the mapping between JDBC data types, which are used in the IMS Metadata class file to define the data type of the fields in the IMS database, and the Java data types.

Table 5-10 JDBC data types to Java data types mapping

JDBC data type	Java data type	Length
BIGINT	long	8 bytes
BINARY	byte[]	1-32 KB
BIT	Boolean	1 byte
CHAR	java.lang.String	1-32 KB
DATE	java.sql.Date	Application-defined
DOUBLE	double	8 bytes
FLOAT	float	4 bytes
INTEGER	int	4 bytes
PACKEDDECIMAL <sup>a</sup>	java.math.BigDecimal	1-10 bytes
SMALLINT	short	2 bytes
TIME	java.sql.Time	Application-defined
TIMESTAMP	java.sql.Timestamp	Application-defined
TINYINT	byte	1 byte
VARCHAR	java.lang.String	1-32 KB
ZONEDECIMAL <sup>a</sup>	java.math.BigDecimal	1-19 bytes

a. Data types of the IMS drivers (not in JDBC Standard)

## 5.7.2 Compatible data transformation functions

Table 5-11 shows the available get methods in the ResultSet or Path Interface for accessing data of a certain JDBC type.

Table 5-11 Available get methods for data types

JDBC Type	TINYINT	SMALLINT	INTEGER	BIGINT	FLOAT	DOUBLE	BIT	CHAR	VARCHAR	PACKEDDECIMAL	ZONEDECIMAL	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



JDBC Type	TINYINT	SMALLINT	INTEGER	BIGINT	FLOAT	DOUBLE	BIT	CHAR	VARCHAR	PACKEDDECIMAL	ZONEDECIMAL	BINARY	DATE	TIME	TIMESTAMP
getBigDecimal	O	O	O	O	O	O	O	O	O	X	X				
getBytes												X			
getDate									O	O			X		O
getTime									O	O				X	O
getTimestamp									O	O			O	O	X

Those marked with “X” indicate methods designed for accessing the given data type. No truncation or data loss will occur using those methods. Those marked with “O” indicate all other legal calls; however, data integrity cannot be ensured using those methods. If the box is empty (it neither contains an “X” or an “O”), using that method for a data type will result in an exception.

**Note:** At this time PackedDecimal and ZoneDecimal data types DO NOT support the Sign Leading or Sign Separate modes. Sign information is always stored with the Sign Trailing method.

**Note:** PackedDecimal and ZoneDecimal are no data types of the JDBC specification



## 6



## Scenario 1 - JDBC data access through tooling

This chapter shows you the possibilities of the IMS Open Database feature for leveraging your existing programs and tools. What is shown in this chapter should also work for other products as the IMS Universal DB drivers are building on open standards like JDBC. They should be compatible with all tools supporting that standard as long as the IMS Universal JDBC driver and the metadata class file are in the application class path available. This solution opens access to IMS data for many available products as well as open source projects.

In this chapter, we demonstrate the following tools and products which you probably have in your shop already or want to try it out for this purpose:

- ▶ IBM Data Perspective in Data Studio and Rational products
- ▶ Accessing IMS Data in Cognos
- ▶ Accessing IMS Data Using the IBM Mashup Center

## 6.1 IBM Data Perspective in Data Studio and Rational products

A Perspective is an arrangement of views in Eclipse. IBM has in some of their Eclipse based Products a Data Perspective integrated. In this case it is an arrangement of useful views concerning databases access, maintenance in combination with application development. It is very much focused on databases with SQL access. With IMS Version 11 and the IMS Universal DB Drivers, it now becomes also easily usable for IMS databases access and application development.

### 6.1.1 Download and Install IBM Data Studio or Rational products

You should be able to use what is shown here with Rational Application Developer, Rational Developer for System z and IBM Data Studio depending on which product you already have your company. Beside this products there are several open source projects and other vendor products available with similar capabilities, which should be configurable and usable in a similar way.

If you do not have one of these products you can try them out by downloading and installing them on your workstation.

- ▶ A special Rational Developer for System z Version for IMS can be downloaded from the IBM IMS Enterprise Suite web site from  
<http://www.ibm.com/software/data/ims/soa-integration-suite/enterprise-suite/>
- ▶ A trial version of Rational Application Developer can be downloaded from  
<http://www.ibm.com/developerworks/downloads/r/rad/>
- ▶ IBM Data Studio is also as a no-charge product available as download  
<http://www.ibm.com/software/data/optim/data-studio/features.html>

In this example we are showing the capabilities of this Perspective along with the IBM Data Studio as it is available as a no-cost product via the IBM web site. For the no-cost Version of IBM Data Studio there are currently two packages available the stand-alone package and the IDE package:

- ▶ The **stand-alone package** is a lighter weight offering ideal for administrators to get up and running quickly and easily. This package is sufficient for most users who do standard database administration or database routine development and includes the new query formatting and pureScale administration support. The stand-alone package does not include SQLJ, Data Web Services, or XML development capability – users who need that capability should download the Integrated Development Environment (IDE) package described below.
- ▶ The **integrated development environment (IDE) package** includes most administrative capabilities as well as an integrated Eclipse development environment for Java, XML (for DB2 for Linux, UNIX, and Windows), and Web services development. You can install this package with compatible Eclipse-based products such as InfoSphere™ Data Architect, Optim™ Development Studio, Optim Database Administrator, and Optim Query Tuner within the same Eclipse instance (shell share), providing seamless movement between data-centric roles, or to share objects across geographically distributed teams.

In our case we have chosen the IDE package for download. The following steps describe shortly the way of installing the product:

1. Go to the IBM Data Studio web site and select the IDE Package for download

2. You will need an IBM ID for the download, which can be also created on the web site (if you do not have one yet).
3. Select the correct Download for your Workstation Operating System as shown in Figure 6-1 and click Continue.

Offering	Platform	Format
<input type="radio"/> <b>IBM Data Studio</b> Version 2.2  Languages: Chinese Simplified, Chinese Traditional, Czech, English, French, German, Hungarian, Italian, Japanese, Korean, Polish, Portuguese Brazilian, Russian, Spanish	Red Hat Linux SUSE Linux Enterprise Desktop(SLED) SUSE Linux Enterprise Server (SLES)	download
<input type="radio"/> <b>IBM Data Studio</b> Version 2.2  Languages: Chinese Simplified, Chinese Traditional, Czech, English, French, German, Hungarian, Italian, Japanese, Korean, Polish, Portuguese Brazilian, Russian, Spanish	Windows Vista Business Windows Vista Enterprise Windows Vista Ultimate Windows XP Professional	download


 **Continue**

Figure 6-1 Data Studio Installation - Operating System Choice

4. Download the Data Studio Install zip File.
5. After Downloading the zip file, extract it to a temporary Directory
6. To Install it you have two options
  - a. Using the Launchpad which is part of the Installation zip File. Therefore change to the extracted directory and start setup.exe.
  - b. If you have IBM Installation Manager already installed, you can also use the Installation Manager and add the disk1 directory of the extracted files to the Repository of the IBM Installation Manager. This approach is necessary if you want to do shell-sharing with other Eclipse Products or if you want to do all Installations with one product.
7. In our case we choose option a) in the previous step and continue the Installation steps on the panel until it is finished.

If you need more Information or help with the Installation see the IBM Data Studio Information Center

[http://publib.boulder.ibm.com/infocenter/idm/v2r2/topic/com.ibm.datatools.ds.install.doc/topics/t\\_install\\_over.html](http://publib.boulder.ibm.com/infocenter/idm/v2r2/topic/com.ibm.datatools.ds.install.doc/topics/t_install_over.html)

## 6.1.2 Configuring IBM Data Studio for use with the IMS Universal JDBC Driver

After Installing IBM Data Studio you now have to do the necessary steps to configure it for the use with the IMS Universal JDBC Driver.

**Tip:** The following steps are similar in other Eclipse based products which has the IBM Data Perspective.

- You can start IBM Data Studio now from the Windows Start Menu or a Desktop Shortcut. It will ask you for a directory for its workspace. The Workspace is a folder where all Eclipse based settings as well as your projects are stored in. If the Directory is not there it will be automatically created. Figure 6-2 shows an example of our Workspace selection:

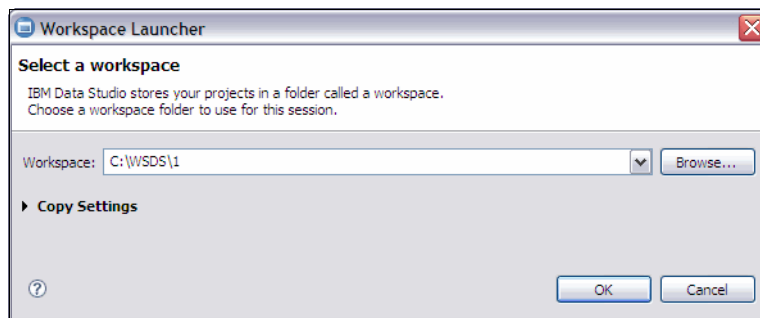


Figure 6-2 Data Studio - Workspace selection

- When you start IBM Data Studio the first time or with a new workspace it will show you the welcome panel shown in Figure 6-3.

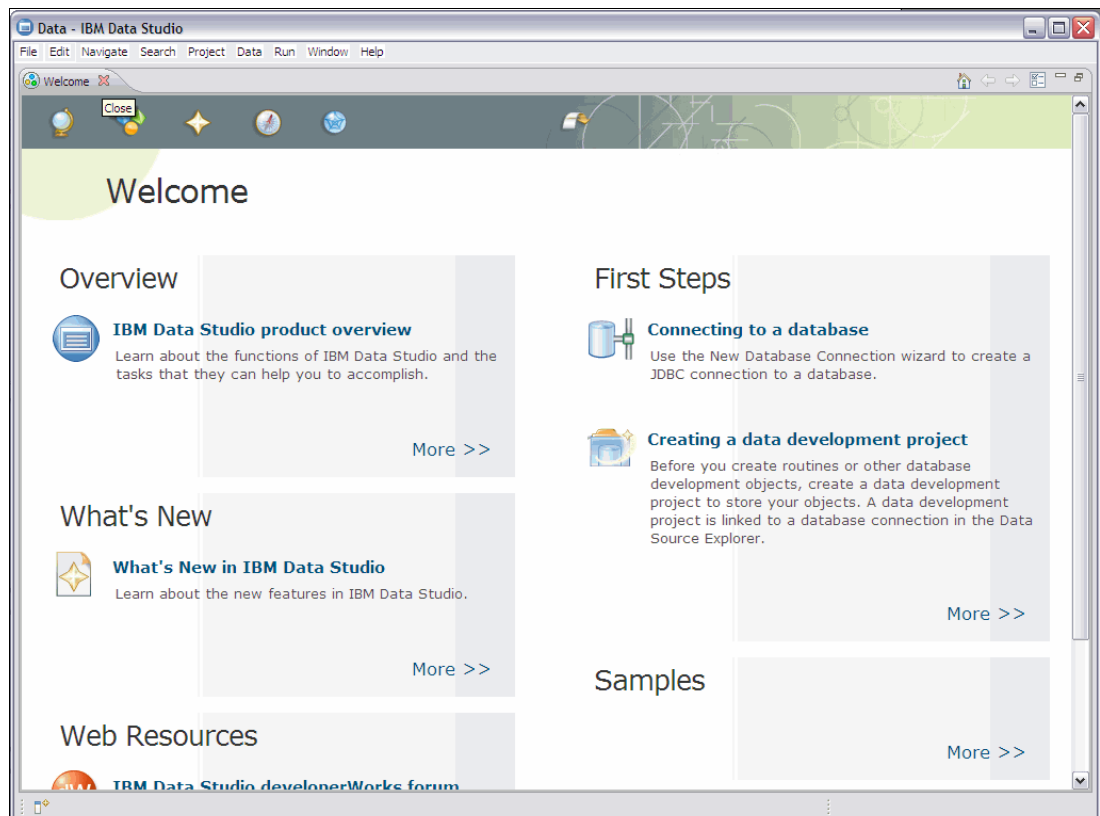


Figure 6-3 Data Studio - Welcome panel

- You can close the Welcome panel by clicking on the x on the top tab bar and it will automatically show you the Data Perspective.

- From the Menu select **Window -> Preferences**. In the Menu drill down to **Data Management -> Connectivity -> Driver Definitions** and select the predefined **Generic JDBC 1.0 Driver** as shown in Figure 6-4.

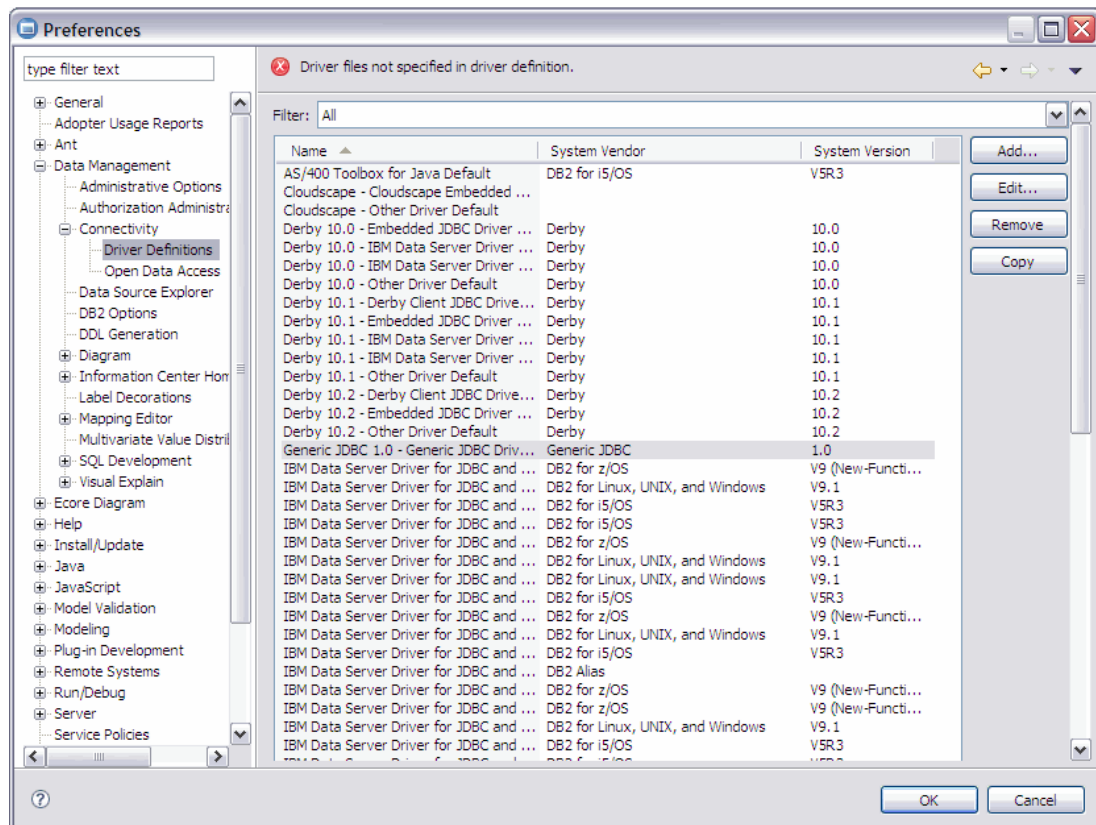


Figure 6-4 Data Studio - Configuring IMS Universal Drivers Step 1

- Click on **Edit...** and switch in the appearing panel to the **Jar List** tab.
- Click the **Add Jar/Zip** button and select the **imsudb.jar** as the IMS Universal Driver on your hard disk.
- Repeat the previous step for adding the Database Metadata Jar Files you want to access in our example this is **AUTPSB11.jar** which is generated in the IMS Enterprise Suite DLIModel Utility chapter. In our example the Jar List tab looks like Figure 6-5.

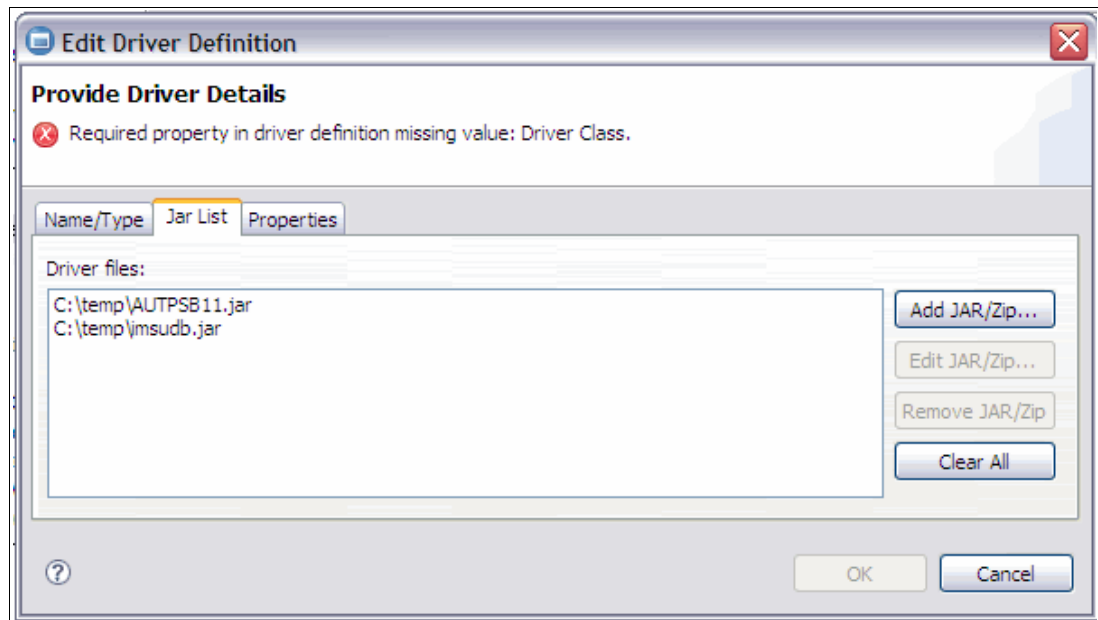


Figure 6-5 Data Studio - Configuring IMS Universal Drivers Step 2

- Switch to the Properties tab and click on the ... in the Driver Class Value field.
- Select **Browse for class** and select **com.ibm.ims.jdbc.IMSDriver** as shown in Figure 6-6 and Click **OK**.

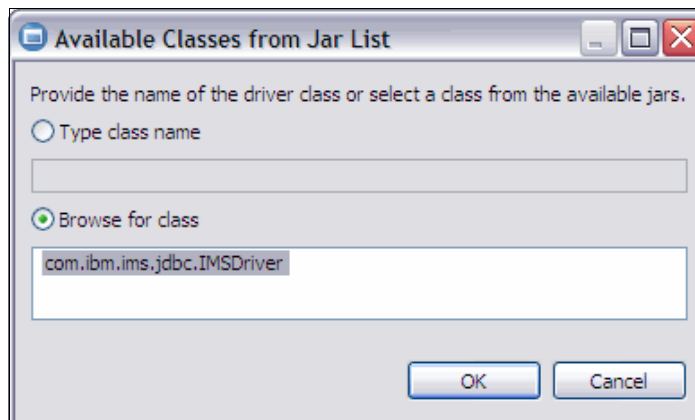


Figure 6-6 Data Studio - Configuring IMS Universal Drivers Step 3

- You can now optionally specify the Connection URL template which is then the default for new connections. We set it to **jdbc:ims://host:5555/class://samples.xDatabaseView:datastoreName=IMS2** as well as the Database Name and the Default User ID as shown in Figure 6-7 and Click **OK**.



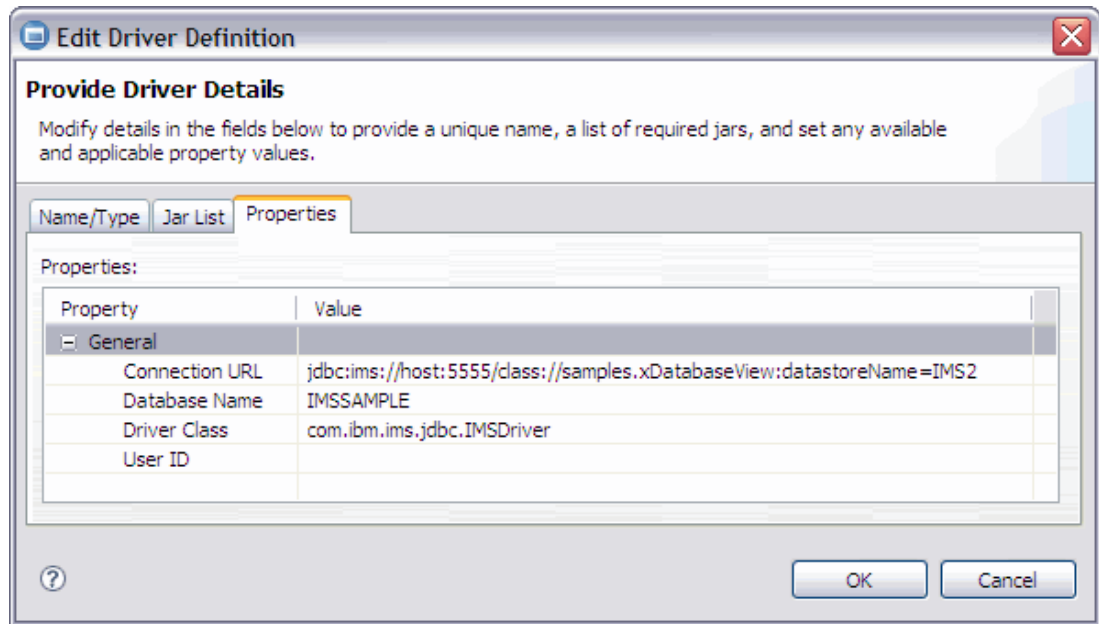


Figure 6-7 Data Studio - Configuring IMS Universal Drivers Step 4

- After Clicking Finish you are done with setup of the workstation and can now use the Data Perspective in your product.

### 6.1.3 Using the Data Perspective with the IMS Universal Drivers

This chapter should give you a brief overview about some interesting features of the Data Perspective and how it works with the IMS Universal Drivers.

#### Creating a Connection to IMS

- At first you have to create a new connection to an IMS Database. Right-click on **Database Connections in the Data Source Explorer** and Select **New...** as shown in Figure 6-8.

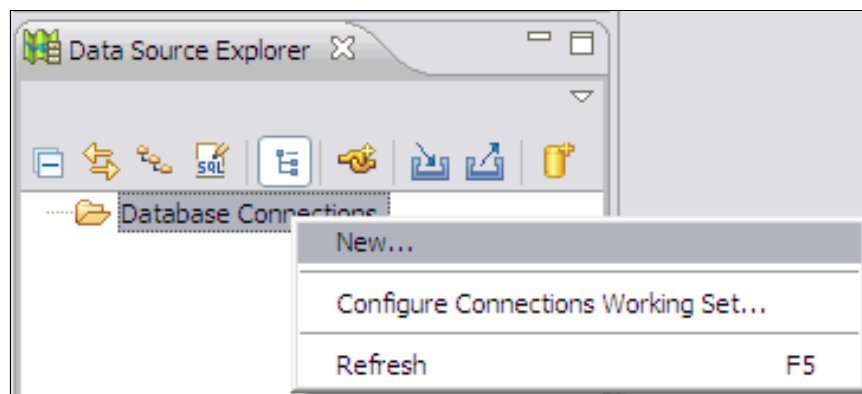


Figure 6-8 Data Studio - Creating a new Connection Step 1

- Select Generic JDBC in the left list and fill in the necessary informations for the correct URL, User name and Password. In our case the URL would look like this `jdbc:ims://myhost:5555/class://samples.ims.openDb.AUTPSB11DatabaseView:datastoreName=IMS2;dpsbOnCommit=true;` as shown in Figure 6-9.

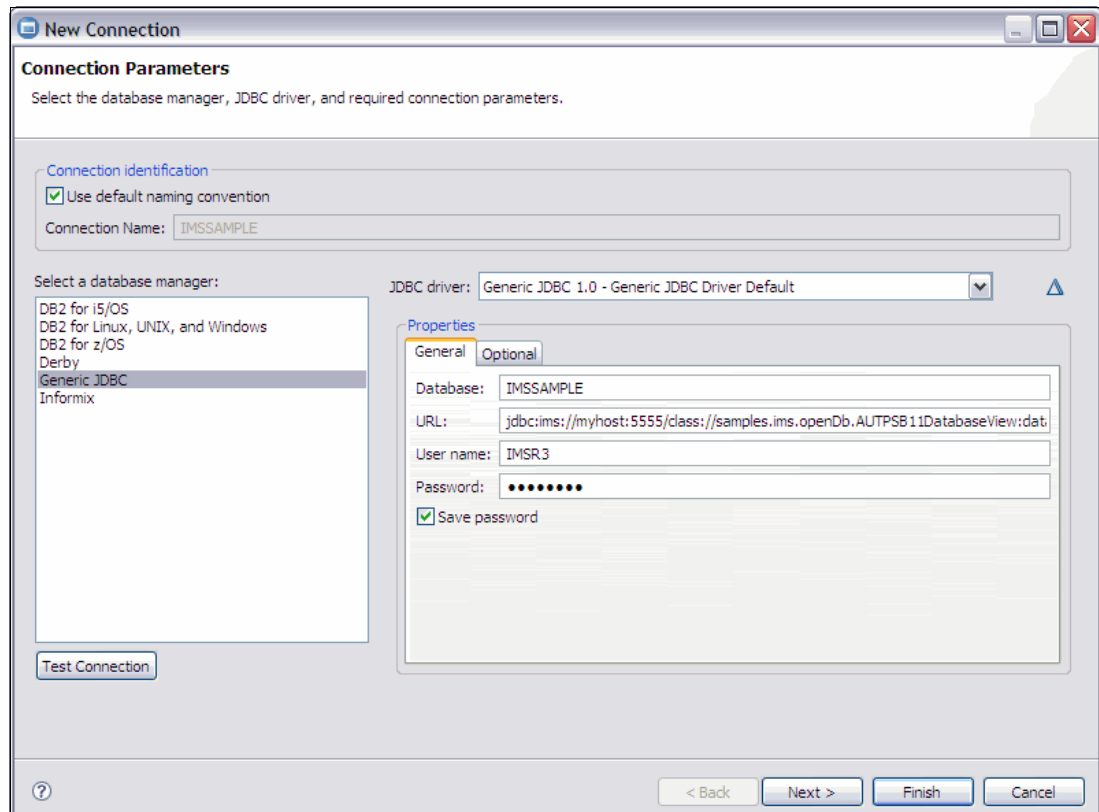


Figure 6-9 Data Studio - Creating a new Connection Step 2

**Note:** The `dpsbOnCommit=true` parameter is recommended in environments which are doing an internal pooling of the connections without notifying the Driver.

- Click **Finish** and you should see your new connection open in the Data Source Explorer.

### Returning Sample Data through Data Source Explorer

After we have now created the connection we can now use it in our further steps

- You can drill down and expand the connection until you see the schemas (PCBs), the tables (segments) and columns (fields) as shown in Figure 6-10.

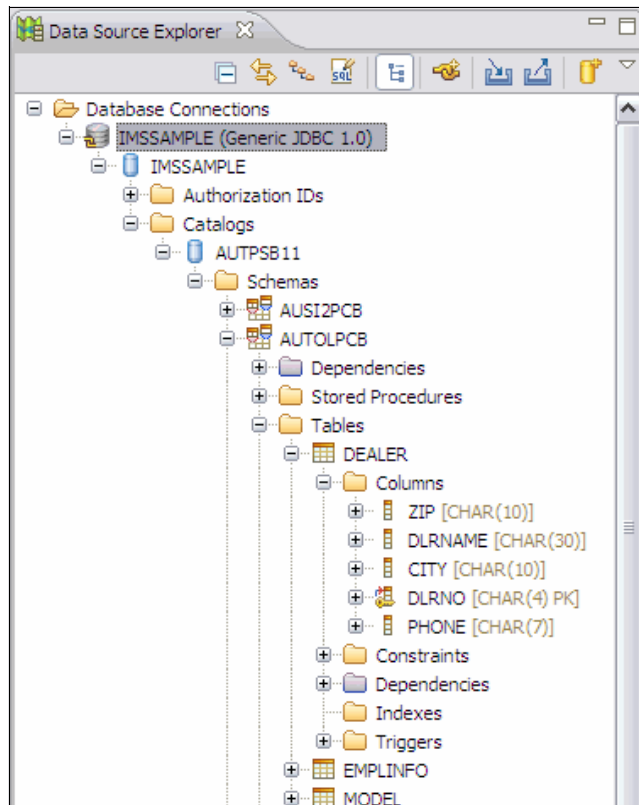


Figure 6-10 Data Studio - AUTPSB11 expanded view

- You can right-click on a table and select **Data -> Return All Rows** as shown in Figure 6-11.

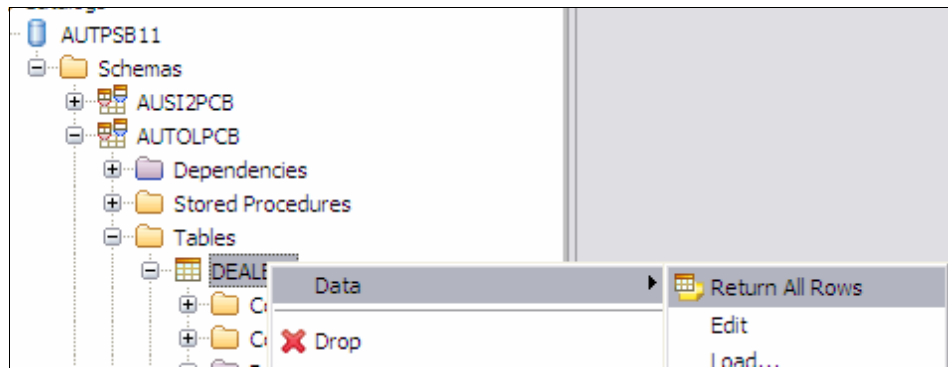
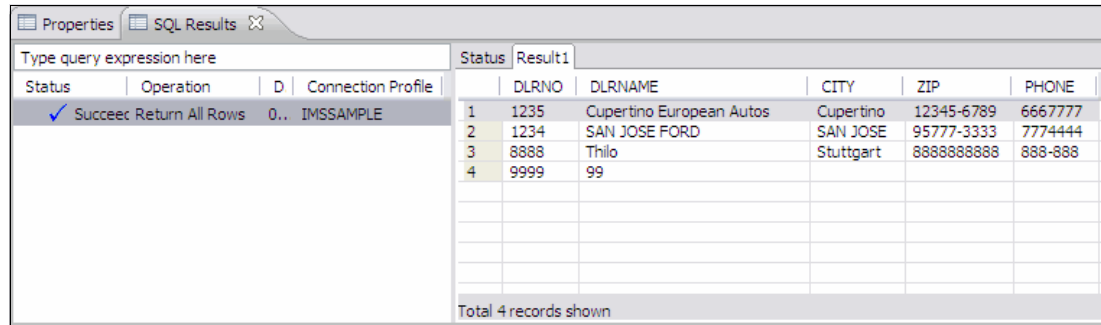


Figure 6-11 Data Studio - Return all Rows

The results back from the IMS Database are shown in Figure 6-12.



Status	Operation	D	Connection Profile	Status	Result1
✓	Succeed	Return All Rows	0.. IMSSAMPLE		
1	1235	Cupertino European Autos	Cupertino	12345-6789	6667777
2	1234	SAN JOSE FORD	SAN JOSE	95777-3333	7774444
3	8888	Thilo	Stuttgart	8888888888	888-888
4	9999	99			

Total 4 records shown

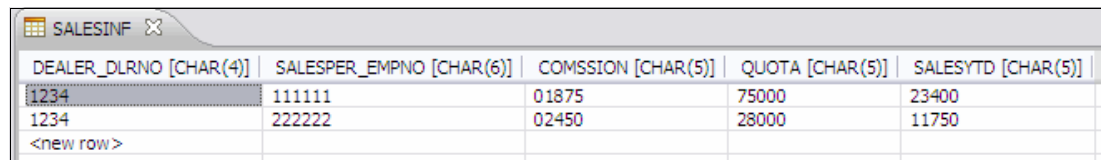
Figure 6-12 Data Studio - Return all Rows - Results

This is a way to easily return some sample Data of a specific segment in a IMS Database

## Editing Data via Data Source Explorer

It is also possible to edit the data in a tabular form with the Data Source Explorer.

- Therefore right-click on a table (e.g. AUTOLPCB.SALESINF) in the Data Source Explorer and select **Data -> Edit**
- You will see the contents of the data in the editor as shown in Figure 6-13.



DEALER_DLRNO [CHAR(4)]	SALESPER_EMPNO [CHAR(6)]	COMSSION [CHAR(5)]	QUOTA [CHAR(5)]	SALESYTD [CHAR(5)]
1234	111111	01875	75000	23400
1234	222222	02450	28000	11750
<new row>				

Figure 6-13 Data Studio - Data Edit

When you have edited some values and save the table it will do automatically all updates, inserts and deletes to the IMS Database

**Restriction:** Be aware that in this example the first two columns are virtual foreign key segments and they cannot be updated, because they do not exist in the database.

## Extracting Data via Data Source Explorer

You can also easily extract some data in a specific format like a comma separated value (CSV) or a XML file.

- Therefore right click on a table-> Select Data->Extract... as shown in Figure 6-14

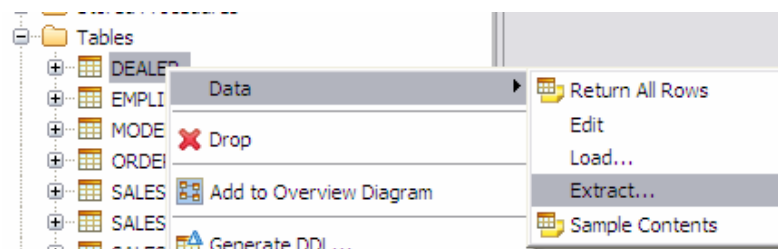


Figure 6-14 Data Studio - Extract Data

- We specify a name Dealer.csv and select comma as separator and click finish. The result is a file which contains the values of the table as shown in Example 6-1.

*Example 6-1 Contents of Dealer.csv*

"1235", "Cupertino European Autos	", "Cupertino "	", "12345-6789", "6667777"
"1236", "IBM Bristol Cars	", "Bristol "	", "BS1 6DG ", "3338888"
"8892", "Thilo	", "Stuttgart "	", "8888888888", "888-888"
"1234", "SAN JOSE FORD	", "SAN JOSE "	", "95777-3333", "7774444"
"8888", "Test1234	", "Stuttgart "	", "8888888888", "888-888"
"9999", "99	", "	", " ", " "

## Creating an Overview Diagram

You can also create some overview diagrams for your data with the Data Source Explorer.

- Therefore select multiple tables (by holding <CTRL>) and right-click and select **Add to Overview Diagram** as shown in Figure 6-15.

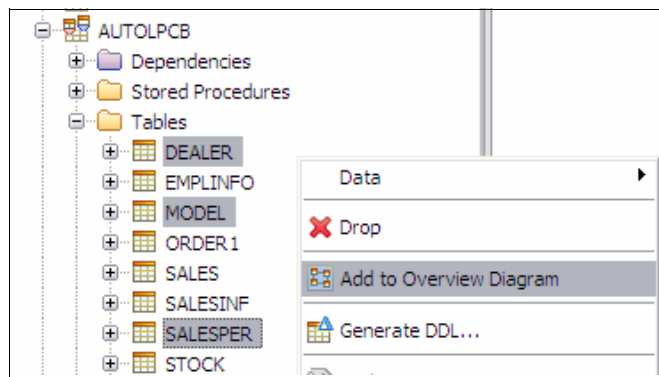


Figure 6-15 Data Studio - Add to Overview Diagram

The generated Overview Diagram is shown in Figure 6-16.

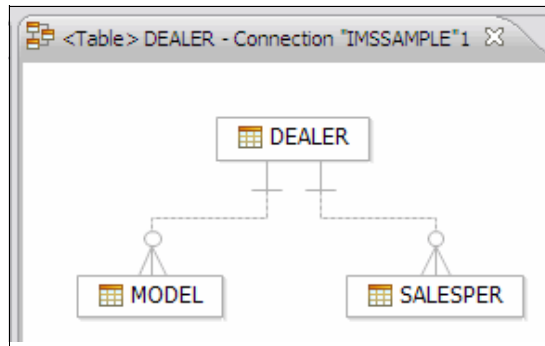


Figure 6-16 Data Studio - Overview Diagram

## Working with Data Development Projects

To write own SQL Queries you have first to create a new Data Development Project.

- Therefore Click on **File -> New... -> Data Development Project**
- Specify the Project Name and click **Next**.
- Select your created **IMS Data Source** and click **Finish**.

Now you will see a new project in the Data Project Explorer.

- Expand the Project and right click on the **SQL Scripts** Folder and select **New -> SQL or XQuery Script** as shown in Figure 6-17.

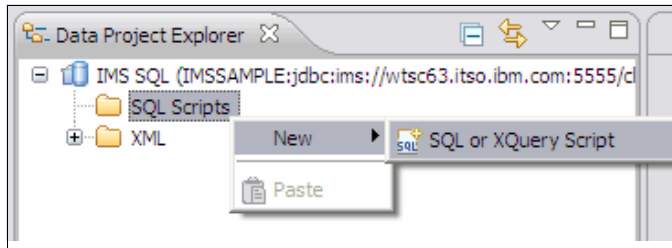


Figure 6-17 Data Studio - New SQL Script

- ▶ You will be asked what type of SQL Script you want to write. Select a single SELECT statement as shown in Figure 6-18 and Click Finish.

**Tip:** The Other Option will give you the ability to write several SQL statements separated through a semicolon like a batch job.

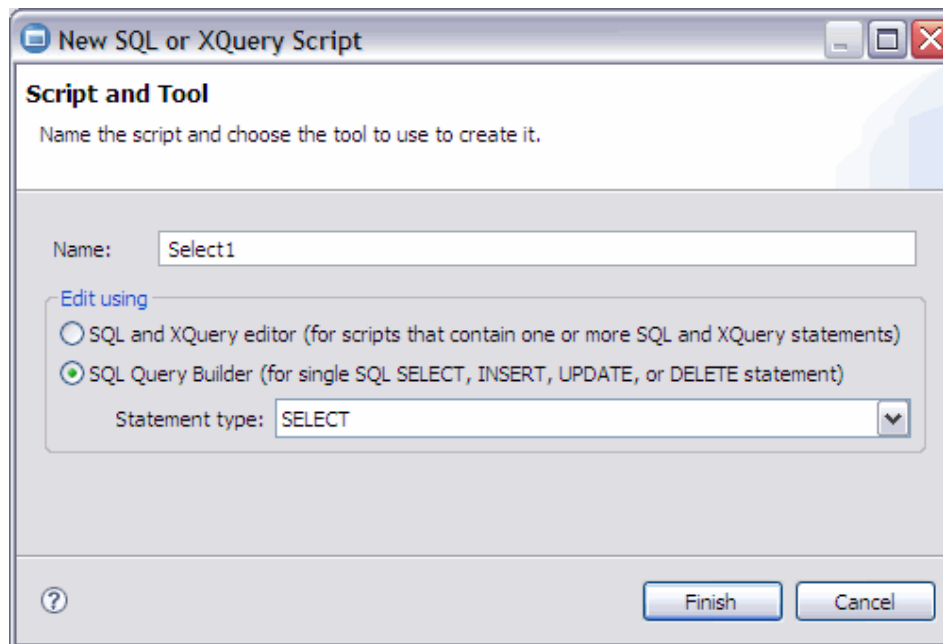


Figure 6-18 Data Studio - New SQL Script Step 2

You can now adding tables by right-clicking on the middle pane and selecting some tables which are in the same schema and hierarchical path for your select statement. You can also specify the Where clauses with the context help of the tool. Figure 6-19 shows an example for a more complex SELECT statement:

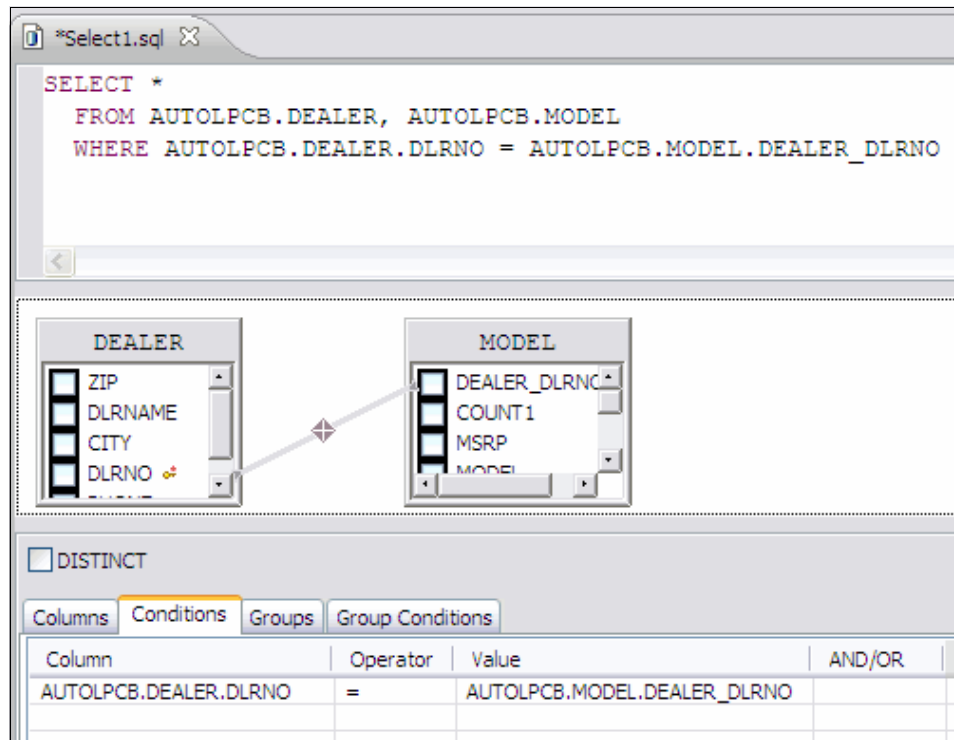


Figure 6-19 Data Studio - SQL Script Step 3

You can now save the SQL and right-click it and say Run SQL and you will get the results back.

There are a lot of more features you can do with the Data Perspective. There are of course also some functions that right now cannot work with the IMS Universal Drivers like a CREATE DB statement. If you want to learn more about IBM Data Studio and its features you will find many samples in the help feature of the product.

## 6.2 Accessing IMS Data in Cognos

Business needs drive the need for information. Sometimes not all information can fit into a data warehouse; moreover, some online data is often indispensable. These reasons are why we want to retain the option to merge different data sources in a BI deployment.

IBM Cognos® is a Business Intelligence and Data Warehouse solution from IBM. You can collect Data from various Data Sources in the Cognos solution. Currently IMS Data access is not supported as a direct DataSource. Up to now to be able to do this you had to either replicate the IMS data to an other Data Source like DB2 to access it or you needed a Federation Server to access IMS databases directly. Starting with IMS Version 11 and the new IMS Universal JDBC Drivers there is an easy way of getting data from your IMS. This section explains what you have to do to get it working and which components are used.

IBM Cognos 8 BI can deliver data from various sources by accessing data sources through IBM Cognos direct access and federating data sources through InfoSphere Federation Server or by using IBM Cognos Virtual View Manager. For accessing IMS data we will use the Cognos Virtual View Manager.

## 6.2.1 IBM Cognos 8 Virtual View Manager

IBM Cognos Virtual View Manager enables data source federation inside the BI environment. As an Enterprise Information Integrator, it provides benefits in terms of connectivity (by way of JDBC) and improves performance in multiproduct joins.

Virtual View Manager is the primary tool used to access multiple data sources and define, publish, and manage resources. Virtual View Manager allows you to:

- ▶ Create, edit, and manage data sources, transformations, views, SQL scripts, parameterized queries, packaged queries, definition sets, triggers, Virtual View Manager databases, and Web services.
- ▶ Publish data sources, transformations, views, SQL scripts, parameterized queries, packaged queries, definition sets, triggers, Virtual View Manager databases, and Web services.
- ▶ Archive Virtual View Manager resources and deploy them back to a desired location with the export/import options. This function is based on Composite Information Server technology, and it delivers standard JDBC drivers to deliver high performance connectivity.



More information about Virtual View Manager software environments can be found at the following address:

<http://www.ibm.com/support/docview.wss?rs=3442&uid=swg27014427>

Virtual View Manager creates views of the database, optimized for IBM Cognos 8, and Framework Manager is then used to model the database view and create a single business view. IBM Cognos 8 components, including Framework Manager, use an Open DataBase Connectivity (ODBC) interface to access a Virtual View Manager data service. The Virtual View Manager Server accesses the data sources through Java Database Connectivity (JDBC), a Java API, ODBC, the OS File System, or SOAP (see Figure 6-20).

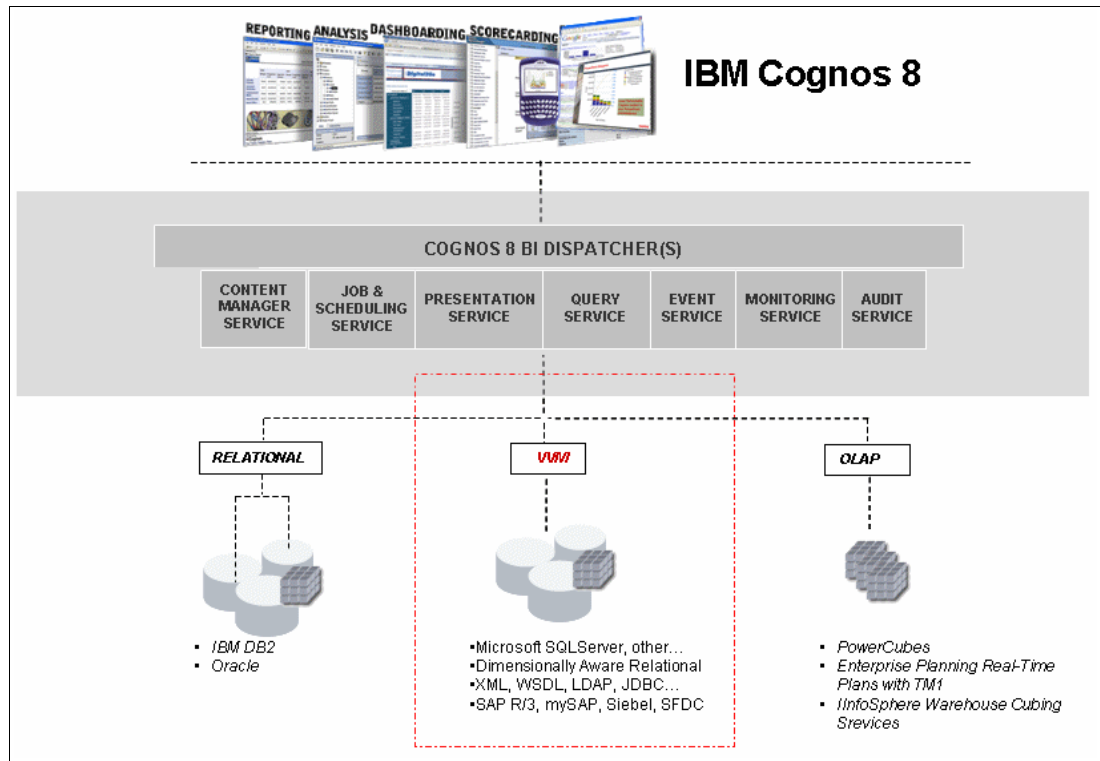


Figure 6-20 Virtual View Manager in the IBM Cognos 8 architecture

Virtual View Manager is composed of two main components: Virtual View Manager Server, which is the main engine that runs processes, and Virtual View Manager Studio, which is a client console that is used to connect, model, and expose data sources.

### Basic concepts of Virtual View Manager

The workflow for using Virtual View Manager is separated into three processes:

- ▶ Setting up the environment by installing and configuring the appropriate software and drivers. The installation also installs IBM Informix® and creates a repository to contain your Virtual View Manager content.
- ▶ Creating a data source using Virtual View Manager Studio, which includes accessing and simplifying the metadata using Virtual View Manager Server.
- ▶ Accessing Virtual View Manager views using IBM Cognos 8 and preparing metadata for reporting in IBM Cognos 8.

Virtual View Manager Server can be installed in the same environment as Cognos 8, but an installation on a separate computer gives better performance and availability. The installation

creates a Virtual View Manager repository and starts both the Virtual View Manager process and the Virtual View Manager Server.

Because IBM Cognos 8 uses the Virtual View Manager ODBC driver to access Virtual View Manager data sources, the ODBC driver and driver manager must be installed on each instance of the IBM Cognos 8 report server (refer to Figure 6-21) and the Framework Manager. The driver manager routes all IBM Cognos 8 requests to the appropriate ODBC driver to access the data sources. When you add an ODBC Data Source Name (DSN) using the ODBC Data Source Administrator, you identify an ODBC driver for the driver manager. The driver manager then knows that the data source associated with this DSN is accessed through a particular ODBC driver.

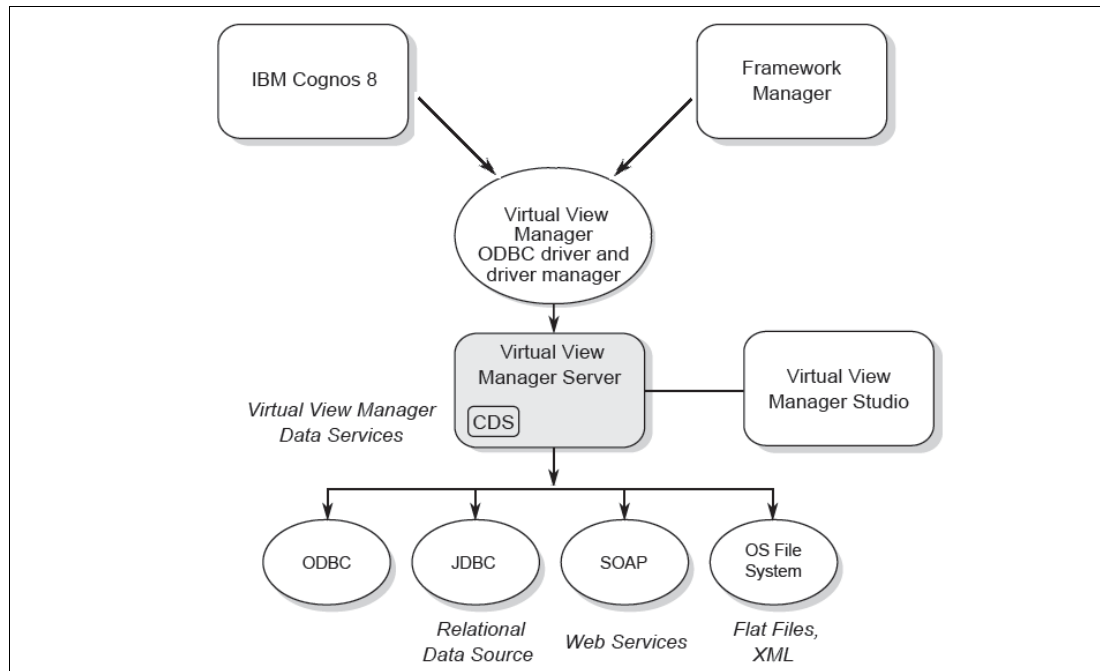


Figure 6-21 IBM Cognos Virtual View Manager architecture

## Virtual View Manager Studio

Virtual View Manager Studio serves as a data view design and development area. It provides three important functions: resource management, modeling, and publishing. Figure 6-22 shows the Virtual View Manager Studio interface.

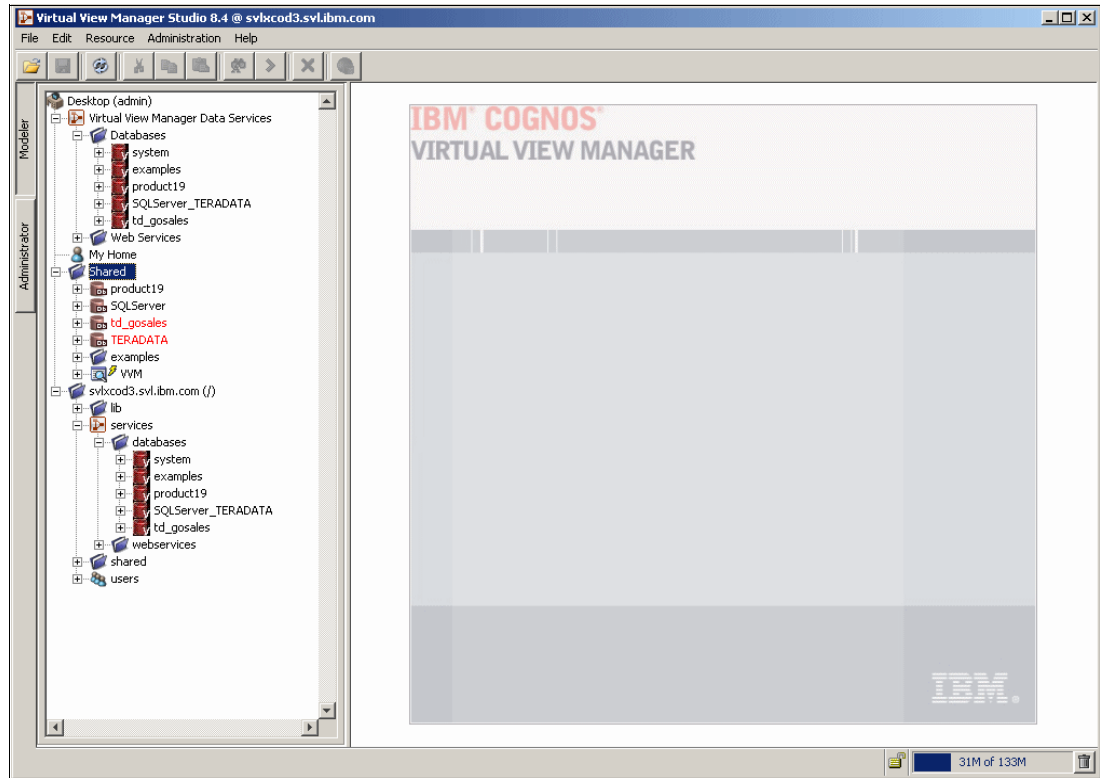


Figure 6-22 Virtual View Manager Studio interface

### Modeling a Virtual View Manager data source

Use ODBC to access the Virtual View Manager data source from IBM Cognos 8 using the command-line utility named `driverConfig` to add an ODBC DSN. The data source associates a particular ODBC driver with the data you want to access through that driver. In order to be able to create an ODBC DSN, the user publishing the views must have the appropriate permissions for the Virtual View Manager configuration files and libraries. The ODBC DSN must be configured with the same parameters on the client in which Cognos 8 Framework Manager is used to model and publish BI contents.

In IBM Cognos 8 BI, a data source is a named set of connections to a physical database or other data source. Cognos 8 BI connects to Virtual View Manager data sources using an ODBC data source connection. Data source authentication is needed when adding a data source.

## 6.2.2 Configuring Virtual View Manager for IMS Data access

In this example we are showing how to configure IBM Virtual View Manager (VVM) 8.4.1 for Windows with the Virtual View Manager Studio on your workstation, but the steps should be very similar in other runtime environments.

**Note:** As VVM is currently using JRE Version 1.5 the Java Metadata files have to be also JRE 1.5 compatible exported, otherwise you will get a Java `unsupportedClassVersion` Exception.

First you have to copy the IMS Universal JDBC Driver and the IMS DB Metadata Jar files to the correct path that VVM can make use of it. Therefore you should create a directory called

**ims\_universal\_jdbc\_driver** in the **conf\adapters\custom** directory of your vvm installation directory. In our case this would look like:

C:\Program Files\cognos\vvm\conf\adapters\custom\ims\_universal\_jdbc\_driver

Copy the IMS Universal JDBC Driver (imsudb.jar) and the IMS DB Metadata Jar files (in this case: AUTPSB11.jar) to this directory.

**Note:** Alternatively, there is always the option to put the driver and metadata files in the JRE lib/ext folder, in this case this would be C:\Program Files\cognos\vvm\jre\lib\ext. But you should avoid this when possible, as it is not a good usage paradigm.

Now you can start the VVM Server via the **Windows start menu -> Programs -> IBM Cognos Virtual View Manager -> Server -> Server start**.

When the Server is started you can start the VVM Console via **Windows start menu -> Programs -> IBM Cognos Virtual View Manager -> Studio -> Studio**

You can now login with the userid and password (default is admin/admin) as shown in Figure 6-23.



Figure 6-23 VVM - Login panel

In the VVM Studio you can right-click on the **Shared** Directory and select **New Data Source** as shown in Figure 6-24.

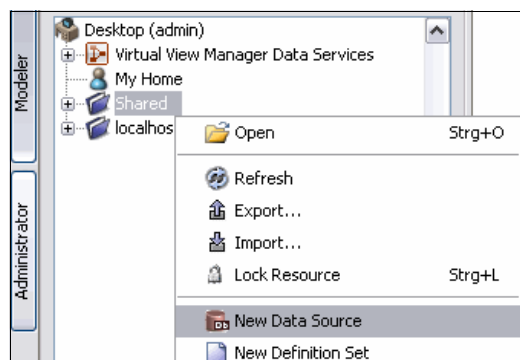


Figure 6-24 VVM - New Data Source - Step 1

In the appearing window (see Figure 6-25) click on the **New Adapter** button.

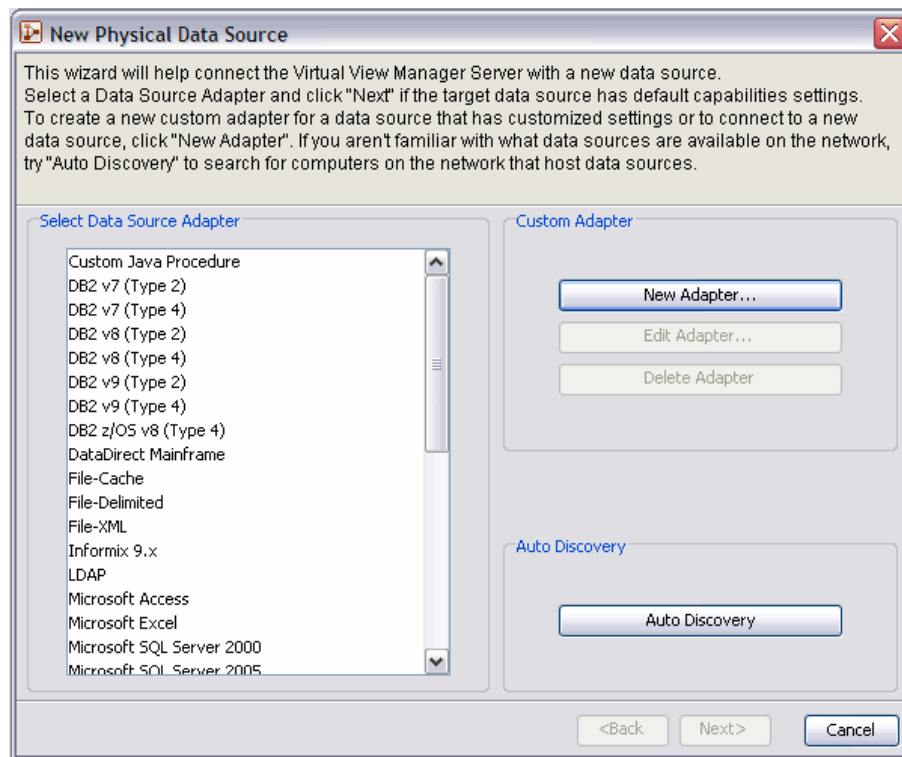


Figure 6-25 VVM - New Data Source - Step 2

Insert the values of Table 6-1 as shown in Figure 6-26.

Table 6-1 New Adapter Values

Attributes	Value
Name	IMS Universal JDBC Driver
Parent Adapter	Generic JDBC
Adapter Class Name	com.ibm.ims.jdbc.IMSDriver
Connection URL Pattern	jdbc:ims://<HOST>:<PORT>/<DATABASE_NAME>

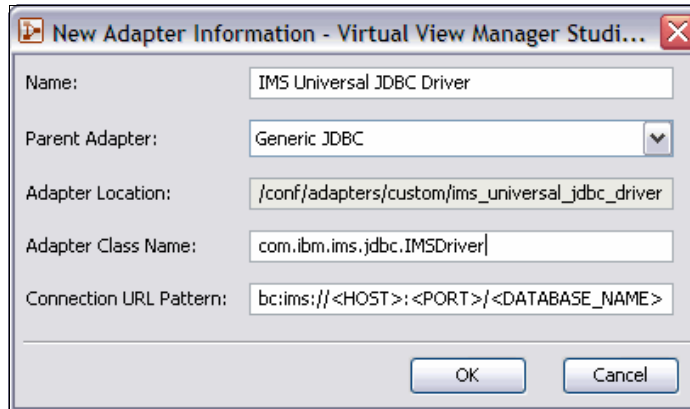


Figure 6-26 New Adapter values

Click **OK** and select the new created Driver from the list and click **Next**. The following window should appear (see Figure 6-27). Specify your IMS Connect hostname or IP Address, Port number, the full qualified name of the Database and User ID and Password if security in IMS is enabled.

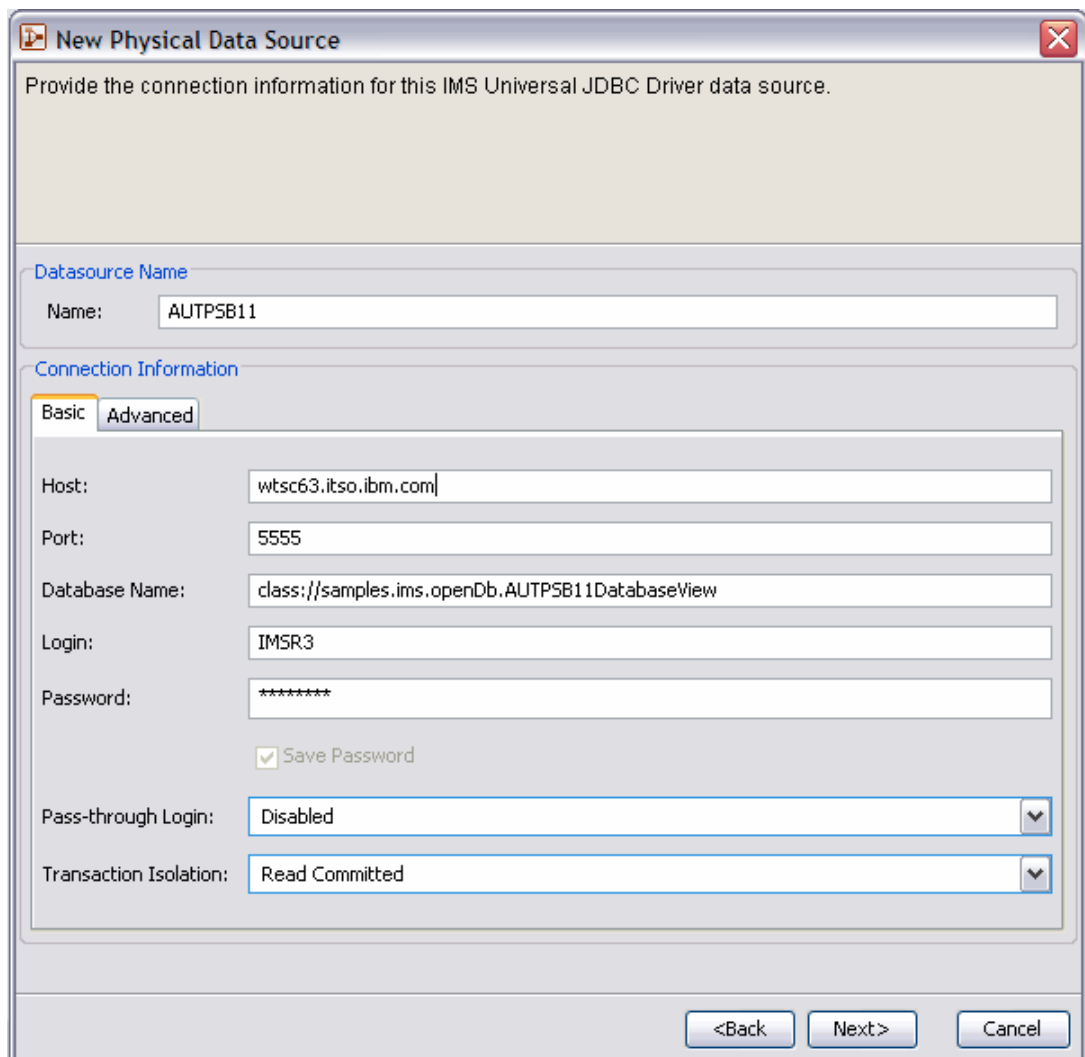


Figure 6-27 VVM - New Data Source Step 3

Switch to the Advanced tab as shown in Figure 6-28.

Figure 6-28 VVM - New Data Source Step 4

Select **Execute SELECTs Independently** and click on the JDBC Connection Properties button. Specify the two connection properties of Table 6-2 as shown in Figure 6-29.

Table 6-2 JDBC Connection Properties

Property	Value
dpsbOnCommit	true
fetchSize	100000

Figure 6-29 JDBC Connection Properties

The dpsbOnCommit value helps the IMS Universal JDBC Driver to keep the connection alive in environments where pooled connections are used without notifying the Driver.

The fetchSize value specifies the number of rows which are collected during one network call. Increasing this parameter influences performance if you do queries with much results.

Click **OK** and click **Next**. In the following panels you can specify which tables you want to select.

Finally you can query some results to test the access to the database by right-clicking on a table and select **Show Contents** as shown in Figure 6-30.

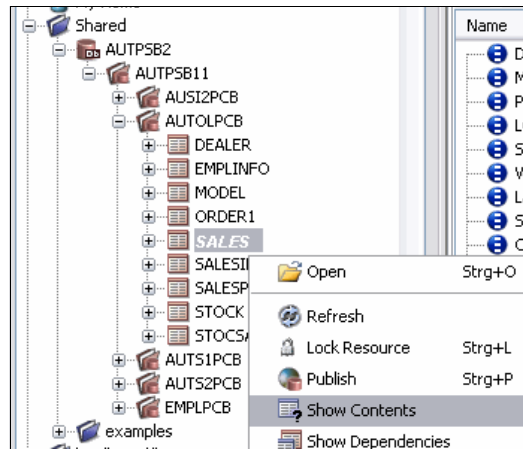


Figure 6-30 VVM - Show Contents of a table

When your connection works you can build an ODBC Connection for Cognos by following the steps on the web site:

[http://publib.boulder.ibm.com/infocenter/c8bi/v8r4m0/index.jsp?topic=/com.ibm.swg.im.cognos.vvm\\_installation\\_guide.8.4.0.doc/vvm\\_installation\\_guide\\_id1191AddSystemDataSource.html](http://publib.boulder.ibm.com/infocenter/c8bi/v8r4m0/index.jsp?topic=/com.ibm.swg.im.cognos.vvm_installation_guide.8.4.0.doc/vvm_installation_guide_id1191AddSystemDataSource.html).

## 6.3 Accessing IMS Data Using the IBM Mashup Center

IMS Web 2.0 support of IBM Mashup Center V2.0 enables customers to create RSS, Atom, or XML feeds from IMS databases by using the IMS Universal DB resource adapter in IMS Version 11.

With IBM Mashup Center V2.0, you are able to extend IMS data into Web 2.0 solutions through standard SQL queries. IBM Mashup Center has a graphical user interface for creating, customizing, and transforming feeds.

IBM Mashup Center also includes a mashup builder to enable Web-savvy business users to mash information from various sources into a single view of disparate sets of information, and create a flexible and dynamic application.

If you do not already have the IBM Mashup Center installed, a 60 day trial Windows-based version, which includes all product features, can be downloaded from

<http://www.ibm.com/software/info/mashup-center/>

Included with the IBM Mashup Center is IBM WebSphere Application Server, Version 7.0.0.5.

Also required is the IMS Enterprise Suite DLIModel utility plug-in, which is needed to generate a metadata file that describes the database view.

We are using the Java metadata class file AUTPSB11.jar generated in the IMS Enterprise Suite DLIModel Utility chapter.

First you need to decide on the database connection approach.



If you plan on using the JNDI (managed) connection type, where WebSphere Application Server manages the connection to the IMS database:

- ▶ Copy the IMS Universal JCA/JDBC resource adapter `imsudbjLocal.rar` to a location that is accessible to WebSphere Application Server. To install the IMS Universal DB resource adapter:
  - Select and open the WebSphere Application Server administrative console from the IBM Mashup Centre v2.0 start menu.
  - In the WebSphere Application Server administrative console, click Resources > Resource Adapters, as shown in Figure 6-31.

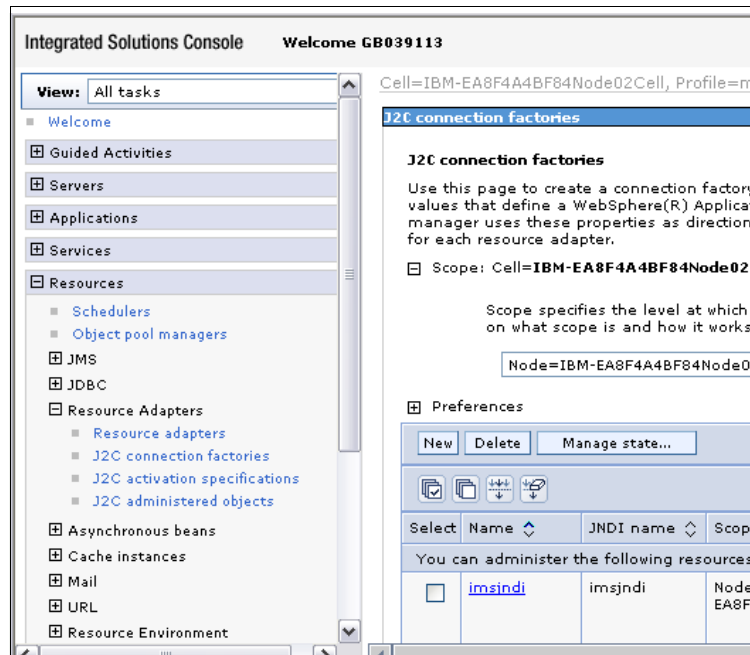


Figure 6-31 Install RAR

- Click Install RAR. The "Install RAR File" page displays.
- Type the path to the `imsudbjLocal.rar` file, or use the Browse button to navigate to and select this RAR file. The path can be to a local location or to a location on the server.
- Click Next. The configuration page opens.
- Click OK. The IMS Universal DB resource adapter - JDBC Local Transaction resource adapter is listed.
- In the messages box, click Save
- ▶ Configure a connection factory and specify a JNDI name in the WebSphere Application Server.
  - In the WebSphere Application Server administrative console, click Resources > Resource Adapters > J2C connection factories. See Figure 6-32.

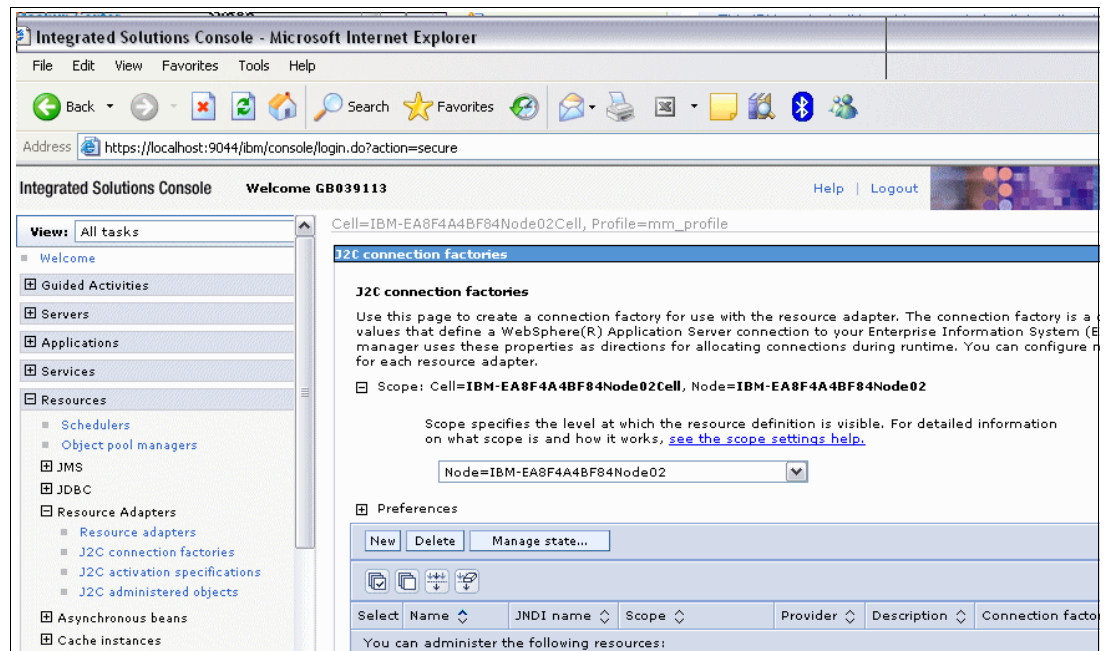


Figure 6-32 Selection of J2C connection factories

- Click New.
  - From the list of providers, select the name of the IMS Universal DB resource adapter that you configured
  - Type a descriptive name for this connection factory.
  - Type the JNDI name for this connection factory.
  - Click Custom properties under Additional Properties.
  - Type the information for the following properties:
    - DatastoreName
    - DatastoreServer (the IMS host system)
    - MetadataURL
    - Password
    - PortNumber
    - User
  - Click Apply.
  - In the messages box, click Save to save this configuration information.
- ▶ Copy the compiled metadata package that you generated into the <install-dir>\AppServer\profiles\profileName\installedConnectors\imsudbjLocal.rar\ directory.
  - ▶ Stop and restart the WebSphere Application Server.

If you plan on using the Driver Manager (non-managed) connection type:

- ▶ Provide the IMS data store name, IMS Connect port number, username and password for connecting to the database, and the fully qualified name of the Java metadata file (metadata URL) that is generated by the IMS Enterprise Suite DLIModel utility plug-in.
- ▶ Copy the metadata package that you generated into <install-dir>\AppServer\lib directory.
- ▶ Stop and restart the WebSphere Application Server.

You can now follow the following actions to create an IMS feed:

- ▶ Select and start the InfoSphere MashupHub from IBM Mashup Centre v2.0 menu.
- ▶ On the Home:Catalog tab, click Create and select New Feed, as shown in Figure 6-33.

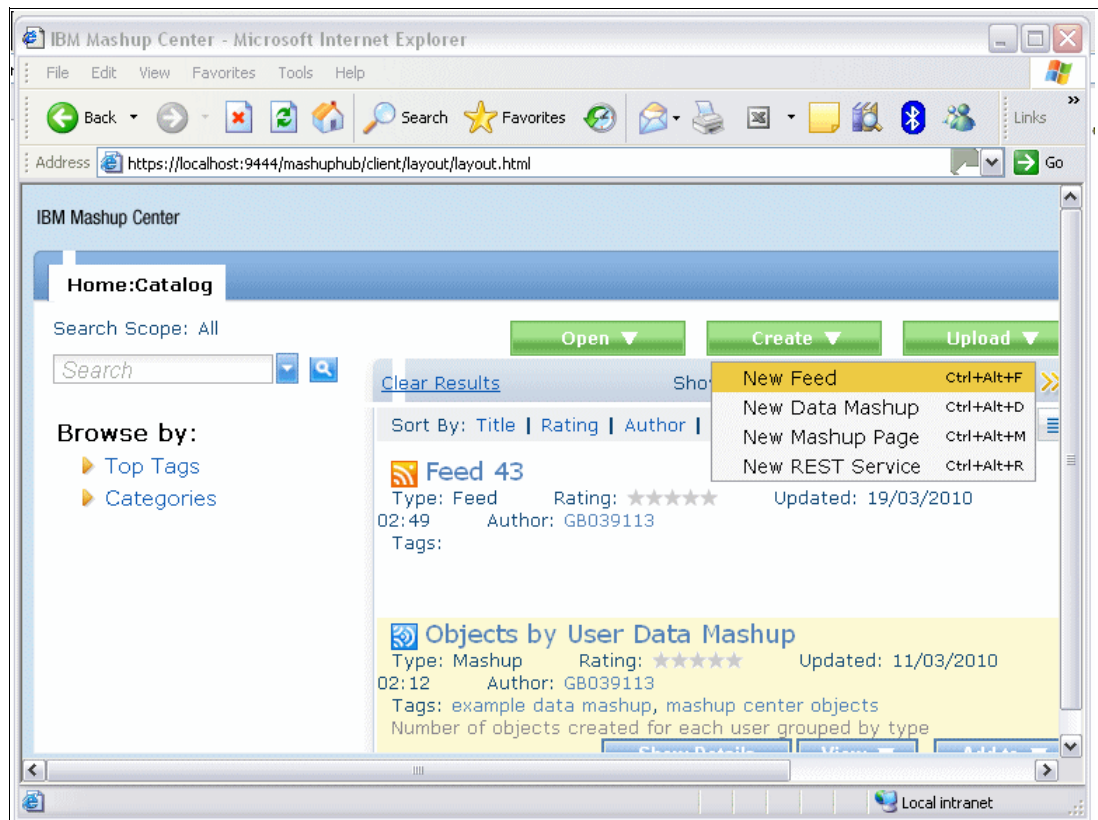


Figure 6-33 Creating a new Feed

- ▶ Select Enterprise database (JDBC) and click Next. See Figure 6-34.

Figure 6-34 Database connection panel

- ▶ In the Select field, select New to create a new database connection profile.
- ▶ In the Profile Name field, type the new profile name.
- ▶ In the Database Type field, select IMS.
- ▶ In the Connection Type field, select whichever of Driver Manager (Non-managed Connection) or JNDI (Managed Connection) you decided to use and fill in the rest of the information required. If you are using the JNDI connection the JNDI Name required is the name of the JNDI name created above.
- ▶ Click Next. The SQL Query Builder opens. See Figure 6-35.

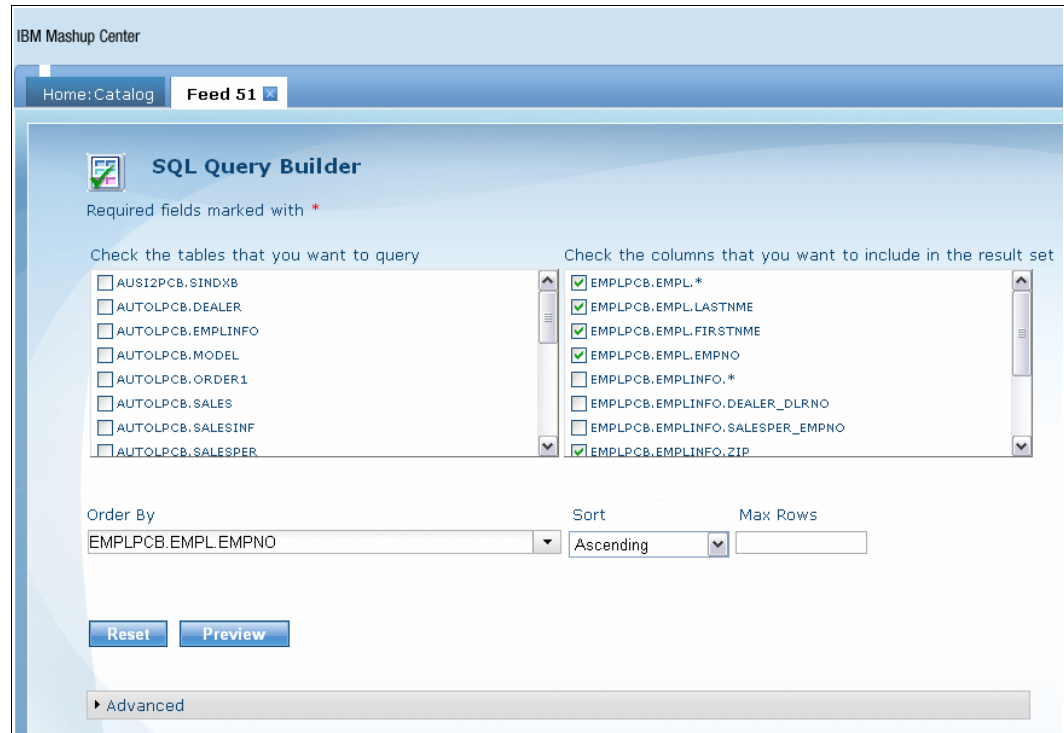


Figure 6-35 The SQL query builder panel

- ▶ In the SQL Query Builder, create the SQL query by selecting the tables and columns to construct a basic query. If customization or more advanced SQL query is required:
  - Use the Generate SQL button to get the query that is associated with your selection of tables and columns, if you want to further refine the query.
  - Click to expand the Advanced field to type or customize your SQL statement.
  - You can see the results of your query in a new window by clicking the Preview button.
  - See the online help for assistance and guidance information about how to formulate your SQL statement.

**Note:** Because IMS data segments are physically stored in a hierarchical database, you cannot alter, create, or drop any table.

- ▶ When you are done with the SQL query, click Next. See Figure 6-36.

IBM Mashup Center

Home: Catalog **Feed 51**

Required fields marked with \*

\* Title:

Description:

\* Version:

Tags:

Permissions:

☒ Public (All users can view the Feed)

☐ Private (Feed is invisible to all other users)

☐ Custom (Custom permission settings)

Advanced:

- Categories
- Related entries already in the catalog
- Technical Documentation
- Caching data
- Access Methods
- Active Content Filtering

Figure 6-36 The final feed panel

- Specify the required information to identify this feed. All fields with an asterisk (\*) are required.
- When you are done specifying the information for this feed, click Finish. You are informed that the information has been saved.

Now you can use the feed in the Mashup Hub for example to combine an IMS feed which displays the address of a Car Dealer and combine this feed with Map to display the dealer on the map when you select him. With this approach it is very easy to generate valuable web sites by using already available information and combining them.





## Scenario 2 - Developing JDBC applications

This chapter focuses on application development with the IMS Universal JDBC Drivers in managed and non-managed environments. It explains the steps very detailed by using an eclipse-based Product. We are using Rational Developer for System z with Java because it gives us across all three scenarios the most support. The following applications will be developed:

- ▶ Developing a stand alone Java application using the IMS Universal JDBC Driver
- ▶ Developing a managed Java application using the IMS Universal Database Resource Adapter (XA) and DB2 Data Server Drivers (XA)
- ▶ Developing an IMS Java Transaction using the IMS Universal JDBC driver
- ▶ To set up the transactions, you have to follow the usual steps for a new IMS Java transaction. The JMP region must be configured correctly for the use with the IMS Universal Drivers in IMS Java transactions. The region startup procedure contains the ENVIRON and JVMOPMAS parameter which specifies the members in the IMS PROCLIB. These configuration files must contain the IMS Universal JDBC Driver jar file in the class path and also the metadata class file (here this would be the generated AUTPSB11.jar). If you use type-2 connectivity you also have to specify the location of the native Drivers for type 2 connectivity.

## 7.1 Developing a stand alone Java application using the IMS Universal JDBC Driver

This scenario shows you how to develop a Java application step by step for a non-managed environment accessing the IMS Car Dealer IVP Database using the IMS Universal JDBC driver.

### 7.1.1 Prerequisites

As prerequisite to be able to follow this scenario you need the following products:

- IDE with Java Development Kit 1.5

In this scenario we use IBM Rational Application Developer (RAD) as integrated development environment (IDE) for Java. Since RAD is an eclipse based product, the same steps are applicable for any other eclipse based Java development product.

If you want to try Rational Application Developer a Download Trial Version is available from the IBM web site <http://www.ibm.com/developerworks/downloads/r/rad/>.

- IMS Universal JDBC Driver

You need the IMS Universal driver called `imsudb.jar` which is the standalone JDBC and DL/1 Driver for IMS.

- Metadata Java class

In our example we use the Metadata for an IMS database which is normally installed as part of the Installation Verification Procedure (IVP) of IMS. It is a Car Dealer Database and uses a PSB called AUTPSB11. The IMS Enterprise Suite DLI Model Utility has generated a class file out of this PSB. It is exported to Jar File called AUTPSB11.jar. The full qualification of the metadata class file is:

`class://samples.ims.openDb.AUTPSB11DatabaseView`

- IMS with IMS Connect

IMS and IMS Connect and the Open Database Manager address space should be set-up correctly for use of the IMS Open Database feature. You need the Hostname or IP address the Portnumber where IMS Connect is listening for Open Database Connections, as well as the DataStore Name or the Alias Name (if specified) of the IMS you want to access. If Security is enabled in IMS Connect you need also a valid User ID and Password.

### 7.1.2 Creating and configuring a new Java Project

Follow the steps:

- Start the Rational Application Developer and select your workspace.
- Open the Java Perspective by clicking on the switch Perspective icon and selecting Java as shown in Figure 7-1.



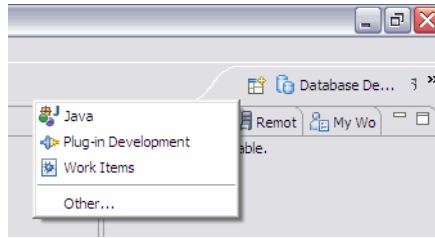


Figure 7-1 Switch to Java perspective

- ▶ Select **File -> New -> Java Project**
- ▶ Specify a name of the Java Project and leave the default settings as they are and click **Finish**.
- ▶ Now you have to add the IMS Universal JDBC Driver as well as the Java Metadata Class File to your Project Build Path that your application will find the referenced Java packages. Therefore right-click on your Project in the Package Explorer View and select **Build Path -> Configure Build Path** as shown in Figure 7-2.

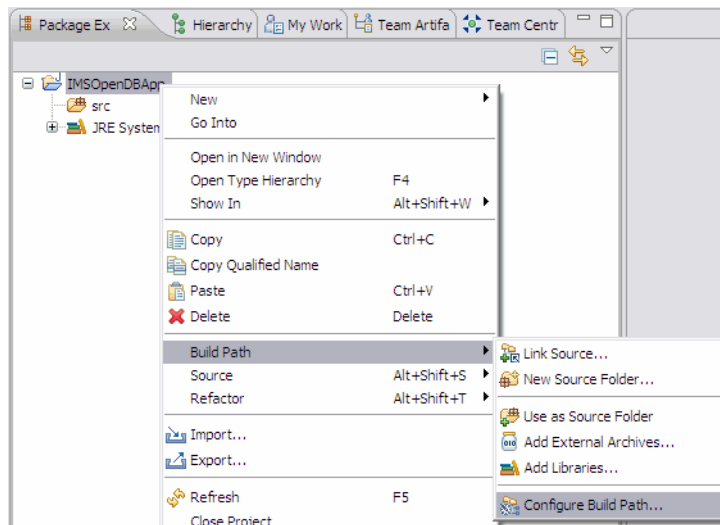


Figure 7-2 Configure Build Path

- ▶ Click on the **Libraries** tab and click on **Add external JARs** and select the **imsudb.jar** on your hard disk.
- ▶ Repeat the last step and add also the Java Metadata jar file containing the IMS Database Metadata class file called **AUTPSB11.jar** to the Build Path libraries.

**Note:** As an alternative you could also add the Metadata source file to the project in the right package. But it is better to have it separated. So you don't have to change the application if you have changes on the database and you just have to replace the referenced Metadata Jar file containing the new Metadata class in the applications Classpath.

- ▶ The result should look like Figure 7-3, then Click **OK**.

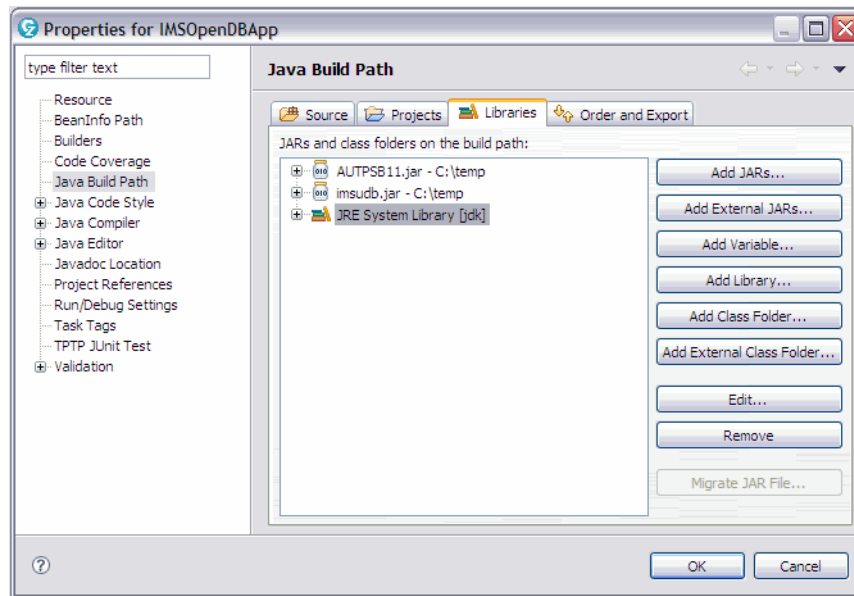


Figure 7-3 Java Build Path Properties

- Now right-click on your project's source folder and select **New -> Package** as shown in Figure 7-4.

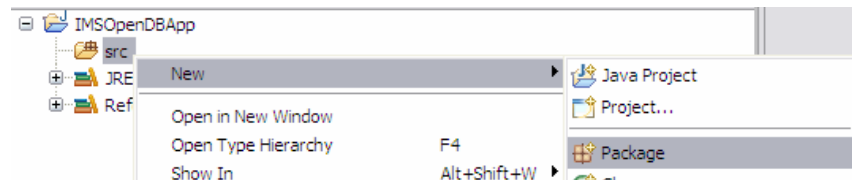


Figure 7-4 New Java Package

- Type in a name for the package like **samples.ims.applications** and click **Finish**.
- Right-click on the new created package and select **New -> Class** as shown in Figure 7-5

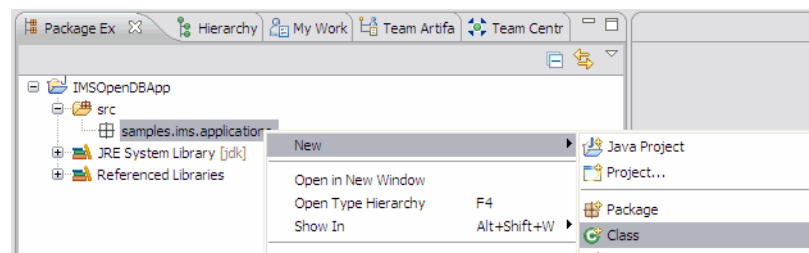


Figure 7-5 New Java Class

- Write a name for the Java class like **IMSJDBCStandalone** and select **public static void main (String[] args)** to generate a Java file with some skeleton methods to define that it is a runnable application. The other values should be automatically filled in like the package name as shown in Figure 7-6 and Click **Finish**.

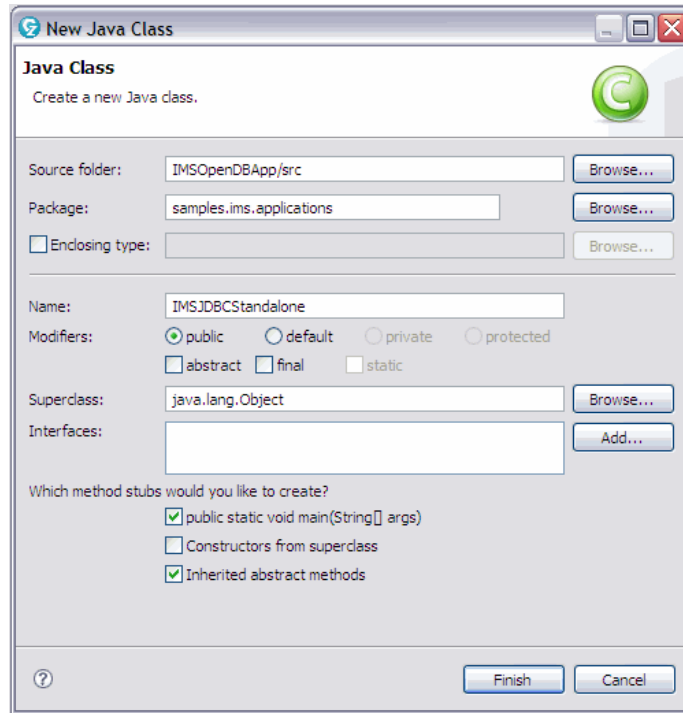


Figure 7-6 New Java Class Properties

### 7.1.3 Writing the application

As you have now a Java application file you can start developing your Java application using the IMS Universal JDBC Driver. The code in Example 7-1 shows a Standalone JDBC Java Application using the IMS Universal JDBC Driver to do SELECT, INSERT, UPDATE and DELETE calls.

*Example 7-1 Code of IMSJDBCStandalone Application*

```
package samples.ims.applications;
import java.sql.*;
import com.ibm.ims.jdbc.*;

public class IMSJDBCStandalone {
    public static Connection conn;
    public static IMSDataSource ds;
    public static Statement st;

    public static void main(String[] args) {
        ds = new com.ibm.ims.jdbc.IMSDataSource();
        ds.setMetaDataURL("class://samples.ims.openDb.AUTPSB11DatabaseView");
        ds.setDatastoreName("IMS2");
        ds.setDatastoreServer("host.itso.ibm.com");
        ds.setPortNumber(5555);
        ds.setDriverType(IMSDatasource.DRIVER_TYPE_4);
        ds.setUser("IMSUSR");
        ds.setPassword("myPw");
        try {
            conn = ds.getConnection();
            conn.setAutoCommit(false);
            st = conn.createStatement();
            displayDealer();
        }
    }
}
```

```

// 9
String sql="INSERT INTO AUTOLPCB.DEALER (DLRNO,DLRNAME,ZIP,CITY,PHONE) "+
"VALUES('8889', 'Thilo Sample Inc', '71139', 'Ehningen','555-123')";
System.out.println("\nSQL Command: "+sql);
int insertCount = st.executeUpdate(sql);
System.out.println("The Number of Rows Inserted: "+insertCount);
displayDealer();

//10
sql="UPDATE AUTOLPCB.DEALER SET PHONE='555-789'" +
" WHERE DLRNO='8889' or PHONE='555-123'";
System.out.println("\nSQL Command: "+sql);
int updateCount = st.executeUpdate(sql);
System.out.println("The Number of Rows Updated: "+updateCount);
displayDealer();

//11
sql="DELETE FROM AUTOLPCB.DEALER WHERE DLRNO='8889'";
System.out.println("\nSQL Command: "+sql);
int deleteCount = st.executeUpdate(sql);
System.out.println("The Number of Rows Deleted: "+deleteCount);
displayDealer();

//12
st.close();
conn.rollback();
conn.close();
System.out.println("\nChanges Successfully committed");
} catch (SQLException e) {
e.printStackTrace();
try {
//13
conn.commit();
conn.close();
System.out.println("Changes rolled back because of an error");
} catch (SQLException e1) {
//14
System.out.println("Error in Connection");
e1.printStackTrace();
}
}
}

public static void displayDealer() throws SQLException{
//15
String sql="SELECT * FROM AUTOLPCB.DEALER";
System.out.println("\nSQL Command: "+sql+"\n");
ResultSet rs = st.executeQuery(sql);
//16
ResultSetMetaData rsmd = rs.getMetaData();
//17
int numColumns = rsmd.getColumnCount();
for (int i=1; i<=numColumns; i++) {
System.out.print(rsmd.getTableName(i)+"."+rsmd.getColumnName(i)+" | ");
}
System.out.println("\n-----"+
"-----");
while (rs.next()) {
//18
for (int i=1; i<=numColumns; i++) {
//19
System.out.print(rs.getString(i) + " | ");
}
System.out.println();
}
rs.close();
//20
}
}

```

---

Here are the explanations for the commented numbers in the code:

1. These are the two import statements you will need for using JDBC and the IMS Universal JDBC Driver
2. The application uses a function which needs access to the Connection, IMSDataSource and the Statement variable. Therefore these variables are globally defined.
3. When the application starts, a new IMSDataSource object is created. The alternative would be to obtain one from a managed container via JNDI.
4. The IMSDataSource object is populated with the required parameters via its setter methods.
5. The Connection is initiated from the IMSDataSource.getConnection() function. This and some of the following functions can throw SQLExceptions which must be caught and handled.
6. As the application does several interactions in one connection and to be sure that all are committed or rolled back disables this function the autocommit for the connection.
7. This command generates a new Statement object for this specific connection.
8. Because this function is called before and after each change in the database it is separated in a function which is called.
9. The statements between comment 9 and 10 are inserting some values in the DEALER segment of the Database. For Insert, Updates and Deletes the Statement.executeUpdate(sql) command is used. The return value of the method is the number of rows affected.
10. The statements between 10 and 11 are updating the just inserted entry and change the PHONE entry with the WHERE clause matches whether the specified DLRNO or PHONE.
11. The statements between 11 and 12 are deleting the just inserted entry again.
12. After finishing sending the queries the Statement is closed and the connection is explicitly committed and closed. You have to ensure to commit the connection before you close it, otherwise the changes will be rolled back.
13. If any Exception is thrown in the Try{...} construct the catch(){...} construct will rollback the changes made and close the Connection.
14. If this also fails there were an error with the connection itself and usually your connection times out and get rolled back on the IMS side, depending on your settings in IMS.
15. The displayDealer() function can throw an SQLException as it contains methods which throw this Exception. It is handled in the main program try{...}catch(){...} construct.
16. To issue a SELECT statement the Statement.executeQuery(sql) function is used. It returns a ResultSet object which pulls the results from IMS depending on the fetchSize settings of the IMSDataSource object.
17. The statements between 17 and 18 demonstrating how to obtain the table and column names from the resultsets.
18. The While(){ } loop will run until there is no more data in the ResultSet.
19. In the DEALER segment every field is as CHAR defined and can be pulled out as String with the ResultSet.getString(columnName or columnIndex) method.
20. The ResultSet is closed when it is not needed anymore.

## 7.2 Developing a managed Java application using the IMS Universal Database Resource Adapter (XA) and DB2 Data Server Drivers (XA)

This scenario shows the important steps to setup the IMS Universal Resource Adapter in WebSphere Application Server, and how to develop a managed application that access IMS and DB2 resources by using the XA in both drivers. This sample also shows that the syntax for programming against IMS and DB2 is very similar.

### 7.2.1 Prerequisites

As prerequisite to be able to follow this scenario you need the following products:

1. Rational Application Developer with Java Development Kit 6

For this scenario we are using IBM Rational Application Developer as integrated development environment (IDE) for Java, because it offers a good integration with WebSphere Application Server and offers some capabilities for an easier development of JEE applications.

If you want to try Rational Application Developer a Download Trial Version is available from the IBM web site <http://www.ibm.com/developerworks/downloads/r/rad/>.

2. WebSphere Application Server 7.0

WebSphere Application Server 7.0 can be used as standalone or can be used as integrated Test Environment Installation of Rational Application Developer.

You can download a Trial Version of WebSphere Application Server from the IBM web site at <http://www.ibm.com/developerworks/downloads/ws/was/>.

3. IMS Universal Database Resource Adapter

You need the IMS Universal Database Resource Adapter called `imsudbJXA.rar` which is the managed JDBC Resource Adapter with XA support.

4. IMS DB Metadata Java class

In our example we use the Metadata for an IMS database which is normally installed as part of the Installation Verification Procedure (IVP) of IMS. It is a Car Dealer Database and uses a PSB called `AUTPSB11`. The IMS Enterprise Suite DLI Model Utility has generated a class file out of this PSB. It is exported to Jar File called `AUTPSB11.jar`. The full qualification of the metadata class file is:

`class://samples.ims.openDb.AUTPSB11DatabaseView`

5. DB2 Data Server Drivers for JDBC and SQLJ

For accessing DB2 tables you will need the Data Server Drivers for JDBC and SQLJ for DB2. Note that this driver is also called the JAVA Common Client (JCC) driver and was formerly known as the IBM DB2 Universal Database™ Driver. Include `db2cc.jar` for JDBC 3.0 and earlier functions and `db2cc4.jar` for JDBC 4.0 and later functions

If you are using DB2 on System z you will also need the correct Driver license jar file. To connect to DB2 z/OS you also need the license file (`db2jcc_license_cisuz.jar`).

For more information about the DB2 drivers see Appendix A.1, "IBM Data Server Drivers and Clients" on page 228.

6. DB2

In this scenario we are also accessing DB2 tables which are stored in DB2 for z/OS, but for the success of this tutorial it can be a DB2 on any platform.

## 7. IMS with IMS Connect and RRS enabled

IMS and IMS Connect and the Open Database Manager address space should be set-up correctly for use of the IMS Open Database feature. As we use the XA Drivers, IMS, IMS Connect and ODBM all need to be configured to use RRS, or you will receive HWSK2880E RRS COMMAND FAILED ... messages from IMS Connect when running the application.

You need the Hostname or IP address the port number where IMS Connect is listening for Open Database connections, as well as the DataStore Name or the Alias Name (if specified) of the IMS you want to access. If Security is enabled in IMS Connect you need also a valid User ID and Password.

## 7.2.2 Installing the products

For the installation steps of the different products see the IBM Information Center for the corresponding product.

For installing WebSphere Application Server 7.0 see

<http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.installation.base.doc/info/aes/ae/welc6topinstalling.html>

For installing Rational Application Developer see

[http://publib.boulder.ibm.com/infocenter/radhelp/v7r5/topic/com.ibm.rad.install.doc/topics/c\\_intro\\_product.html](http://publib.boulder.ibm.com/infocenter/radhelp/v7r5/topic/com.ibm.rad.install.doc/topics/c_intro_product.html)

## 7.2.3 Creating the Projects in Rational Application Developer

This paragraph explains step by step how to create and setup the necessary files and Projects which are needed later in Rational Application Developer:

- ▶ Start Rational Application Developer.
- ▶ Select **File->New->Other** and select **Enterprise Application Project** and click **Next**. Name it **IMSandDB2EAR** and select WebSphere Application Server 7.0 as Runtime and EAR Version 5.0 as shown in Figure 7-7 and click **Finish**.

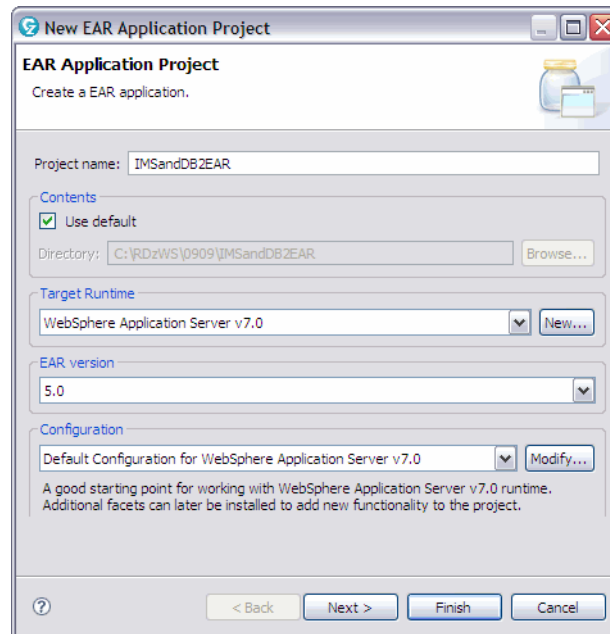


Figure 7-7 New EAR Project

- Select **File->New->Other** and select **EJB Project**. Name it **IMSandDB2EJB** and select **Add Project to an EAR** and select the created EAR Project. Make sure you have selected EJB Module version **3.0** and click **Next**. **Deselect Generate a Client Jar** and select **Generate Deployment Descriptor** as shown in Figure 7-8 and Click **Finish**.

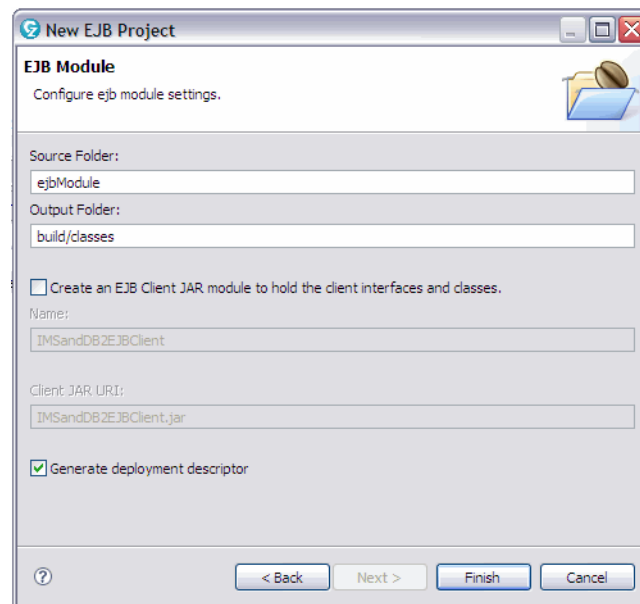


Figure 7-8 New EJB Project

- Select **File->Import** and select RAR File and click **Next**. Select the **imsudbJXA.rar** on your hard disk and click **Finish**
- Select **Window->Open Perspective->Other** and select **Java EE** and click **OK**.
- Right-click the EJB Project and select **Build Path->Configure Build Path**. Switch to the **Projects** tab and add the imported **imsudbJXA** and click **OK**.



- Select **File->New->Other** and select **Session Bean** and click **Next**. Select your created EJB Project and name the Bean **XASessionBean** in the Java Package **samples.ims** as shown in Figure 7-9 and click **Finish**

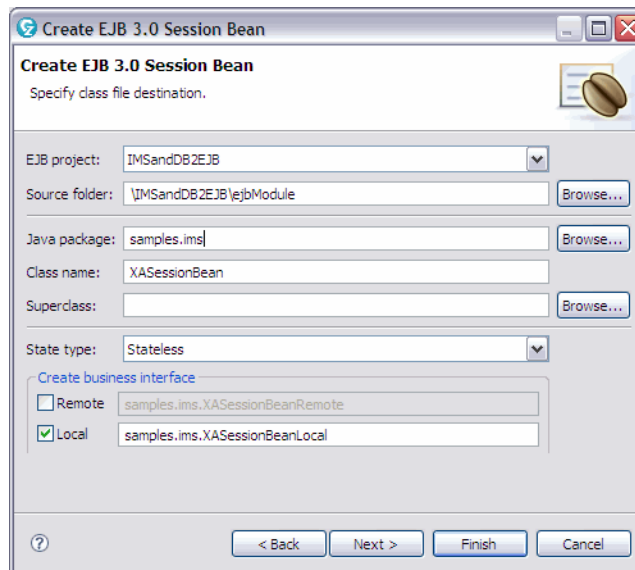


Figure 7-9 New EJB 3.0 Sessionbean

- Select **File->New->Other** and select **Dynamic Web Project** and click **Next**. Name it **IMSandDB2Web** and select **Add Project to an EAR** and select the created EAR Project. Make sure you have selected Dynamic Web Module version **2.5** and click **Finish**.
- Right-click the Web Project and select **Build Path->Configure Build Path**. Switch to the **Projects** tab and add the **IMSandDB2EJB** project and click **OK**.
- Select **File->New->Other** and select **Servlet** and click **Next**. Select your created Web Project and name the Servlet **XAServlet** in the Java Package **samples.imsservlet** and click **Finish**
- Select **File->New->Other** and select **Web page** and click **Next**. Select your created Web Project and name the web page **XATest.jsp** and click **Finish**

## 7.2.4 Sample code for a managed environment

After setting up the required Projects and Build Paths you can now start to write the application. This paragraph explains an example coding which allows to issue SQL commands against IMS and DB2 by using the XA Drivers for both components and the managed JDBC programming approach. The EAR Project doesn't need to be changed. You have to modify the EJB and the EJB Interface in the EJB project and the Servlet and the JSP Web Site in the Web Project.

### XASessionBeanLocal Interface

Example 7-2 shows the sample code for the Java EJB files in the EJB Project.

*Example 7-2 XASessionBeanLocal.java*

```
package samples.ims;
import javax.ejb.Local;

@Local
public interface XASessionBeanLocal {
```

```

        String execute(String SQL,String imsusr,String imspw,
                        String SQL2,String db2usr,String db2pw);
    }

```

---

The `XASessionBeanLocal` is the Interface for the EJB and defines it's functions. In this case the only function is `execute()` and expects as parameters the SQL commands, the Userids and passwords for IMS and DB2.

## XASessionBean implementation

Then you have to implement the `XASessionBean` itself. Example 7-3 shows the sample code.

### Example 7-3 *XASessionBean.java*

---

```

package samples.ims;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import javax.annotation.Resource;
import javax.annotation.Resource.AuthenticationType;
import javax.ejb.Stateless;
import javax.sql.DataSource;

@Stateless //1
public class XASessionBean implements XASessionBeanLocal {
    public XASessionBean() {}

    @Resource(name="imsjndixa",authenticationType=AuthenticationType.APPLICATION,shareable=true
    ,type=javax.sql.DataSource.class,mappedName="imsjndixa") private DataSource imsDS; //2
    @Resource(name="db2jndixa",authenticationType=AuthenticationType.APPLICATION,shareable=true
    ,type=javax.sql.DataSource.class, mappedName="db2jndixa") private DataSource db2DS;

    public String execute(String SQL, String imsusr, String imspw,
                        String SQL2, String db2usr, String db2pw) {
        String result=""; // 3
        try{
            if(!(SQL.isEmpty())){ //4
                Connection conn = imsDS.getConnection(imsusr,imspw); //5
                Statement st =conn.createStatement();
                result+="

### 


```

```

        result+=st.getUpdateCount() + "Rows Updated <br>"; //10
    }
    st.close(); //11
    conn.close();
}
} catch(Exception ex){ //12
    ex.printStackTrace();
    result+="" + ex.getMessage() + "</b><br>";
}
try{
    if(!(SQL2.isEmpty())){ //4
        Connection conn2 = db2DS.getConnection(db2usr,db2pw); //5
        Statement st2 =conn2.createStatement();
        result+="

### 


```

1. This is a EJB 3.0 and uses annotations for certain declarations. This annotation defines this as an Stateless EJB, which means that it doesn't save any information in it.
2. These are the two Resource annotations for the IMS DataSource and the DB2 DataSource. Each annotation specifies the JNDI name, if the Connection is shareable and the authentication type. Here it is AuthenticationType.APPLICATION as the application specifies the userid and the password. The alternative would be AuthenticationType.CONTAINER which would use a JAAS alias in WebSphere for authentication.
3. In this example the return value is a String, which is the result String. Every Output is appended to the string. This application mixes the business logic layer with the

presentation layer, what you shouldn't do in reality. The alternative would be to generate an object which holds the data in it and do the formatting in the presentation layer.

The DB2 code part are very similar to the IMS code part, as it uses also the `javax.sql.DataSource` interface and supports the same functions, and is programmed in the same approach. Therefore the numbers 4-12 appear two times in the code.

4. That you can also use the application if you do not have access to IMS or DB2 this check verifies if the SQL String is empty. If it is empty it won't open a connection to the resource.
5. This statement gets a Connection from via JNDI referenced `javax.sql.DataSource` Object by specifying the userid and the password for the authentication on the target system.
6. With this command the Statement executes the SQL. The return type indicates if your SQL is manipulating or just querying data. If it is a query than it the return value is true and will go in the if branch, otherwise it will go in the else branch.
7. In the query branch you have to get to the ResultSet with the `st.getResultSet()` function. After that it will generate the header with the format `<tablename.columnname>` and put the results in a HTML table.
8. This scenario assumes that all return types are Strings. If not it will try anyway to cast the result as String, if this doesn't work it will fail with an exception.
9. After finishing reading the results, the ResultSet Object can be closed and the data can be released.
10. In the manipulate data branch, if you do an INSERT, UPDATE or DELETE you will get the number of the affected rows back as result.
11. After finishing all work the Statement and Connection Objects can be marked as ready for closing. This doesn't mean that the Connection is really closed, because in a Global transaction the Container is deciding when to close the Connection object. Normally this is done after committing or rolling back all changes in the two phase-commit processing involved resources.
12. By calling the Connection, several methods can throw Java Exceptions for example if the user authentication fails. This Exceptions are caught and appended to the result to make them visible on the result web page.

## XAServlet implementation

Example 7-4 shows the sample code for the files in the Web Project.

*Example 7-4 XAServlet.java*

---

```
package samples.imsservlet;
import java.io.IOException;
import javax.ejb.EJB;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import samples.ims.*;

public class XAServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public XAServlet() {
        super();
    }
}
```

```

protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    doPost(request,response);
}

@EJB(beanName="XASessionBean") private XASessionBeanLocal bean;
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    String result=" ";
    HttpSession session = request.getSession(true);
    String query =request.getParameter("query").trim();
    String imsusr=request.getParameter("imsusr").trim();
    String imspw =request.getParameter("imspw").trim();
    String query2=request.getParameter("query2").trim();
    String db2usr=request.getParameter("db2usr").trim();
    String db2pw =request.getParameter("db2pw").trim();
    result = bean.execute(query,imsusr,imspw,query2,db2usr,db2pw);
    session.setAttribute("result", result);
    RequestDispatcher disp=getServletContext().getRequestDispatcher("/XATest.jsp");
    disp.forward(request, response);
}
}

```

The coding of the XAServlet is kept very simple. It uses the usual methods with doGet and doPost of a HttpServlet. The doGet method forwards the request to the doPost method. It catches the parameters from the web site form element and passes it to the EJB.execute() method. The response result is placed in the session object. The RequestDispatcher forwards the response to the jsp web site again.

## XATest JSP Web Site

Example 7-5 shows the source code of the XATest.jsp web site.

### Example 7-5 XATest.jsp

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"><%@page
    language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<html><head>
<title>IMS and DB2 Access via JDBC XA</title>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
</head><body>
<h1>IMS and DB2 Access via JDBC XA DataSource</h1>
<form method="post" action="/IMSandDB2Web/XAServlet">
<table>
<tr><td><b>IMS</b></td></tr>
<tr>
<td><input type="text" name="query" value="SELECT * FROM AUTOLPCB.DEALER" size="50"></td>
<td>User</td><td><input type="text" name="imsusr" size="9"></td>
<td>PW</td><td><input type="password" name="imspw" ></td>
</tr>
<tr><td><b>DB2</b></td></tr>
<tr>
<td><input type="text" name="query2" value="SELECT * FROM DSN8910.EMP" size="50"></td>
<td>User</td><td><input type="text" name="db2usr" size="9"></td>
<td>PW</td><td><input type="password" name="db2pw"></td>
</tr>
<tr><td><input type="submit" name="sub" value="Execute"></td></tr>
</table>

```

```
</form>
<BR>
<%= (String)session.getAttribute("result") %>
</body>
</html>
```

---

The XATest.jsp file contains only usual HTML elements. By pressing the submit button it will forward the values to the XAServlet, and after getting the result it will read the result String from the session context.

## 7.2.5 Exporting the application

Now the application is ready for deployment in WebSphere Application Server. Rational Application Developer offers you an easy way to publish your application to a WebSphere Application Server, but very often you have restrictions in accessing the administration port directly. For this reason, in this scenario we show the way that should always work to export the application and install it via the administration web console of the WebSphere Application Server.

- ▶ Right-click on your EAR Project and select **Export->EAR file**
- ▶ Choose a destination like **c:\temp\DB2andIMS.ear** and click **Finish**.

The EAR Project automatically contains all referenced Projects like the Web Project and the EJB Project and are part of the ear file.

## 7.2.6 Setting up the IMS Universal DB Resource Adapters in WebSphere Application Server 7.0

You have to install the correct IMS Universal DB Resource Adapter and configure it afterwards. This is done with the help of the WebSphere Application Server administrative console. When you have WebSphere Application Server running you can access the administrative console via opening it in your web browser. The default URL is:

<http://localhost:9061/ibm/console>

Whether you have Security enabled or not you have to enter your User ID and Password or leave it just blank to login.

### Installing IMS DB Resource Adapter

When the WebSphere configuration web site is open, then expand **Resources -> Resource Adapters -> Resource Adapters**. Select the scope where you want to install the Resource Adapter, normally you deploy it to the whole node as shown in Figure .



Figure 7-10 WAS - Console

Click on **Install RAR**. Specify the location of the IMS Universal DB Resource Adapter on your local file system and click **Next**.

**Note:** This scenario uses the `imsudbJXA.rar` Resource Adapter, because the application uses JDBC and requires Global XA Transaction support.

In the next step you should specify the Class path. The Class path should contain any additional resources which are needed. In the case of IMS the IMS Universal Resource Adapters need the Metadata class files to access an IMS database.

**Note:** The Metadata Class files can be part of the Class path of the IMS Universal DB Resource Adapter or can be part of the application. To be more flexible it is easier to specify it in the Class path of the Resource Adapter. So if you change your database layout, you do not have to rebuild all applications which contain the metadata file.

In this case we are specifying it as part of the Resource Adapter Class path. Therefore refer the Class path to the jar file with the Metadata class file in it for the Car Dealer IVP Database. In this example the path points to `c:\temp\AUTPSB11.jar` as shown in Figure 7-11. If you specify more than one jar file, then do not use separators but use a new line instead. Click **OK** and save the changes to the Master configuration.

Name  
S Universal DB Resource Adapter - JDBC XA Transaction

Description  
IMS Universal DB Resource Adapter - JDBC with XA Transaction support.

Archive path  
\${CONNECTOR\_INSTALL\_ROOT}

Class path  
c:\temp\AUTPSB11.jar  
c:\temp\DFSIVP1.jar

Native library path

☐ Isolate this resource provider

OK Reset Cancel

Figure 7-11 WAS Console - Specify Class path

## Creating J2C Connection factories

After the installation of the Driver itself, you have to make it usable to the application by specifying a certain connection to a system. This is done by creating a J2C Connection factory.

In the administration console click on **Resources->Resource Adapters-> J2C connection factories**. Select the same scope as where you have installed the Resource Adapter and click on **New**.

Select the installed Resource Adapter from the Provider list and specify the name and JNDI name as **imsjndixa** as shown in Figure 7-12 and click **Apply**.

General Properties

\* Scope  
cells:W500-4061BL5Node01Cell:nodes:W500-4061BL5Node01

Provider  
IMS Universal DB Resource Adapter - JDBC XA Transaction  
Create New Provider

\* Name  
imsjndixa

JNDI name  
imsjndixa

Description  
IMS Universal JDBC Resource Adapter with XA and AUTPSB11 in Class path against IMS2 on myhost.itso.ibm.com.

\* Connection factory interface  
javax.sql.DataSource

Figure 7-12 WAS Console - New J2C connection factory

After clicking Apply you should be able to click on the **custom properties** link and edit the values accordingly to your system settings.

- The **SSLConnection** parameter indicates if SSL encryption is enabled for the connection to IMS Connect or not. Set this value to **false** as this scenario doesn't use SSL.



- ▶ The **DatastoreName** is the name of the target IMS datastore. If you have specified an alias name in your ODBM configuration member on the mainframe side, than you have to use the alias name instead of the IMSID. In this book this value would be **IMS2**.
- ▶ The **DatastoreServer** is the name or IP address of the target IMS Connect. In this book the value would be **myhost.itso.ibm.com**
- ▶ The **PortNumber** specifies the port number of the target IMS Connect. In this book the value would be **5555**.
- ▶ The **DriverType** specifies the connectivity type. In the case of the XA Resource Adapter this value must be set to **4**.
- ▶ The **LoginTimeout** specifies the number of seconds to wait for a TCP/IP response from IMS Connect. A value of 0 indicates an unlimited wait time. Change this value to **10** that your application do not wait forever in an error case.
- ▶ The **MetadataURL** is the URL of the database metadata representing the target IMS database. It is build by the pattern `class://package.PSBDatabaseView`. In our case the value is **class://samples.ims.openDb.AUTPSB11DatabaseView**
- ▶ The **User** and **Password** specifies the userid and the password for the default user. This user is used if you do not specify a user explicitly and no JAAS alias is provided to the Resource Adapter. As we specify this values explicitly in the application by invoking the connection you do not have to specify them, but you can if you want.

After you have finished editing your settings you can **save your changes to the master configuration**.

## 7.2.7 Setting up DB2 Data Server Drivers in WebSphere Application Server 7.0

The installation of the DB2 Data Server Drivers has a similar pattern as the installation of the IMS Universal Drivers. The difference is, that we are not using a DB2 Resource Adapter but using the Drivers directly as JDBC Provider. If you want to know more about the IBM DataServer Drivers refer to Appendix A.1, "IBM Data Server Drivers and Clients" on page 228.

### JDBC Provider installation

At first you have to specify the Driver location which is made available for using in a specific scope.

Click on **Resources->JDBC->JDBC Providers**. Select the Scope for the installation and click **New**. Select **DB2** as Database type, and select the **JCC Driver** with the **XA data source** implementation as shown in Figure 7-13 and click **Next**.

Figure 7-13 WAS Console - New JDBC Provider

Specify the Class path where your DB2 Data Server Driver jar files are located. For example **c:\temp\db2** means that WebSphere will automatically search for **db2jcc4.jar**, **db2jcc\_license\_cu.jar** and **db2jcc\_license\_cisuz.jar** in that directory. Click **Next** and **Finish** in the Summary page and **save the changes to the master configuration**.

## DataSource setup

Second you have to create a DataSource for a specific Database connection which can be referenced by applications with its JNDI name.

Click on **Resources->JDBC->Data sources**. Select the same scope as for the Driver installation and click **New**. Specify **db2jndixa** as name and JNDI name and Click **Next**.

Select an **existing JDBC Provider** and choose **DB2 Using IBM JCC Driver (XA)** and click **Next**.

Specify the following attributes:

- ▶ **Driver type** as **4**
- ▶ **Database name** as your Sub System ID for your DB2 on System z database (i.e. **DB9A**)
- ▶ **Server name** which is the host or IP address of the target DB2 (i.e. **myhost.itso.ibm.com**)
- ▶ **Port number** (default is **50000**)

Click **Next** and **Next** again and finally **Finish** on the Summary page. **Save the changes to the master configuration**.

## 7.2.8 Installing and starting the application

After setting up the required DataSources for IMS and DB2 you are now ready to deploy the application in WebSphere Application Server.

- ▶ Go in the WebSphere Application Server administration console and expand **Applications->New Application**. Click **New Enterprise Applications**.
- ▶ Select the exported **ear file** from your local file system and click **Next**
- ▶ Select the Fast Path option for a quick installation and click **Next**

- ▶ Leave the defaults on the following pages and click **Next** and **Next** again until you are coming to the Map Resource References.
- ▶ Click on **Browse** for each DataSource and **select the matching JNDI DataSource** as shown in Figure 7-14.

Select	Module	EJB	URI	Resource Reference	Target Resource JNDI Name	Login configuration
<input type="checkbox"/>	IMSandDB2EJB	XASessionBean	IMSandDB2EJB.jar,META-INF/ejb-jar.xml	imsjndixa	imsjndixa Browse...	Resource authorization: Per application
<input type="checkbox"/>	IMSandDB2EJB	XASessionBean	IMSandDB2EJB.jar,META-INF/ejb-jar.xml	db2jndixa	db2jndixa Browse...	Resource authorization: Per application

Figure 7-14 WAS Console - JNDI Mapping

- ▶ Click **Next** and then **Finish** and check if the application is installed successfully. **Save the changes to the master configuration.**

Now the application is installed but not yet started. To start the application follow these steps:

- ▶ Click on **Applications->Application Types->WebSphere enterprise applications**
- ▶ Select your application and click **Start**. Check that the status icon turns green.

## 7.2.9 Running the application

Now the application should work. Open a Web Browser and your application should be reachable by the following link:

<http://localhost:9081/IMSandDB2Web/XATest.jsp>

The IP address and port number may be different. The IP address is the IP address or hostname of the server running WebSphere Application Server. The port number can vary on the settings of the Application Server. To determine which port number to use, from the WebSphere administrative console, click **Servers>Server Types>WebSphere Application servers**. Then click on your **server, Ports** (under the Communications heading on the right-hand side, and see what the value is for "**WC defaulthost**". In this example it is 9081.

The web page you will see is very simple. It contains fields for SQL commands against IMS and DB2 as well as for the userid and password for each connection. You can now play around with issuing SQL commands. If you want just use one connection leave the SQL for the other connection empty.

The result of a query against IMS and DB2 will look like Figure 7-15.

### IMS and DB2 Access via JDBC XA DataSource

**IMS**  
 SELECT \* FROM AUTOLPCB.DEALER User  PW

**DB2**  
 SELECT \* FROM DSN8910.EMP User  PW

**IMS Results**

for SELECT \* FROM AUTOLPCB.DEALER

DEALER.DLRNO	DEALER.DLRNAME	DEALER.CITY	DEALER.ZIP	DEALER.PHONE
1235	Cupertino European Autos	Cupertino	12345-6789	6667777
8892	Thilo	Stuttgart	888888888	888-888
1234	SAN JOSE FORD	SAN JOSE	95777-3333	7774444
8888	Test1234	Stuttgart	888888888	888-888
9999	99			

**DB2 Results**

for SELECT \* FROM DSN8910.EMP

EMP.EMPNO	EMP.FIRSTNME	EMP.MIDINIT	EMP.LASTNAME	EMP.WORKDEPT	EMP.PHONENO
000010	CHRISTINE	I	HAAS	A00	3978
000020	MICHAEL	L	THOMPSON	B01	3476
000030	SALLY	A	KWAN	C01	4738
000050	JOHN	B	GEYER	E01	6789

Figure 7-15 Managed Application Web Site Results

## 7.3 Developing an IMS Java Transaction using the IMS Universal JDBC driver

IMS offers two types of IMS dependent regions for Java the JMPs and JBPs:

- ▶ JMPs are Java Message Processing regions which are capable of running IMS transactions written in Java.
- ▶ JBPs are Java Batch Processing regions which are capable of running IMS batch jobs written in Java.

To write an application which can receive messages and reply messages you have to write an application for JMP regions.

Since IMS Version 11 there is a new IMS Java dependent region resource adapter which provides an enhanced programming API for writing JMP and JBP applications for IMS which is used in this scenario.

For more information about writing with this adapter see Chapter 37 Programming for IMS Java dependent regions of *IMS Version 11 Application Programming*, SC19-2428 and also *IMS Version 11 Application Programming APIs*, SC19-2429

This scenario shows you an example code of a IMS Java Transaction which reads the SQL Code in the input message and replies the result as an output message. The input and output message classes extends the `com.ibm.ims.application.IMSFieldMessage`.

The code of the `InputMessage` is shown in Example 7-6.

*Example 7-6 InputMessage.java*

---

```
package samples.ims.openDb;
import com.ibm.ims.base.*;
import com.ibm.ims.application.IMSFieldMessage;

public class InputMessage extends IMSFieldMessage {

    static DLTypeInfo[] fieldInfo = {
        new DLTypeInfo("SQL",    DLTypeInfo.CHAR,  1, 80),
    };

    public InputMessage() {
        super(fieldInfo,89, false);
    }
}
```

---

The `InputMessage` defines only one `DLTypeInfo` with the length of 80 Bytes. This is for the SQL command.

Example 7-7 shows the `OutputMessage`. It also defines only one field with the length of 220 Bytes for the result of the transaction.

*Example 7-7 OutputMessage.java*

---

```
package samples.ims.openDb;
import com.ibm.ims.base.*;
import com.ibm.ims.application.IMSFieldMessage;

public class OutputMessage extends IMSFieldMessage {

    static DLTypeInfo[] fieldInfo = {
        new DLTypeInfo("Results",DLTypeInfo.CHAR,1,220),
    };

    public OutputMessage() {
        super(fieldInfo,220, false);
    }
}
```

---

The transaction program with the main method is the `CarDealerTrans` class. This class uses the IMS Java dependent region resource adapter. See Example 7-8.

*Example 7-8 CarDealerTrans*

---

```
package samples.ims.openDb;
import com.ibm.ims.dli.DLIException;
import com.ibm.ims.dli.tm.Application;
import com.ibm.ims.dli.tm.ApplicationFactory;
import com.ibm.ims.dli.tm.IOMessage;
import com.ibm.ims.dli.tm.MessageQueue;

public class CarDealerTrans {
    public static void main(String[] args) throws DLIException {
```

```

Application app = ApplicationFactory.createApplication();
MessageQueue mq = app.getMessageQueue();
IOMessage inMsg = app.getIOMessage("class://samples.ims.openDb.InputMessage");
IOMessage outMsg = app.getIOMessage("class://samples.ims.openDb.OutputMessage");

CarDealerDBInteractions dbcalls = new CarDealerDBInteractions();
String host = "myhost.itso.ibm.com";
int port = 5555;
String datastoreName = "IMS2";
String username = "USRID";
String password = "PASS";

while (mq.getUnique(inMsg)) {
    String sql = inMsg.getString("SQL");
    String output = dbcalls.sqlMethod(sql,4,host,port,
                                     datastoreName,username,password);
    outMsg.setString("Results", output);
}
}

```

---

The main application access the message queue and retrieves messages from it, as long as there are messages for the application on the queue. It defines a `CarDealerDBInteractions` object and calls the `sqlMethod()` by passing the required parameters to it. The application can be easily switched between a type 2 and type 4 connectivity. This would be helpful if you want to access another database of the same type in a remote IMS. So you could easily access two databases across two IMS systems. There are several scenarios possible.

The database calls are separated in an own `CarDealerDBInteraction` class. This class uses the IMS Universal JDBC Driver. See Example 7-9.

---

*Example 7-9 CarDealerDBInteraction*

---

```

package samples.ims.openDb;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import com.ibm.ims.jdbc.IMSDataSource;

public class CarDealerDBInteractions {
    public String sqlMethod(String sqlQuery, int driverType, String host,int port,
                           String datastoreName, String username,String password){
        String result = "";
        IMSDataSource ds = new IMSDataSource();
        ds.setMetadataURL( "class://samples.ims.openDb.AUTPSB11DatabaseView" );
        ds.setDatastoreName(datastoreName);
        ds.setDriverType(driverType);

        if(driverType==4){
            ds.setDatastoreServer(host);
            ds.setPortNumber(port);
            ds.setUser(username);
            ds.setPassword(password);
        }
        try {
            Connection conn = ds.getConnection();
            Statement st = conn.createStatement();
            ResultSet rs = st.executeQuery(sqlQuery);
            while(rs.next()){

```

```

        for(int i=1; i<=rs.getMetaData().getColumnCount();i++){
            result += rs.getString(i) + "\t";
        }
        result += "\n";
    }
    if(result == ""){
        result += "No records match the provided query";
    }
    rs.close();
    st.close();
    conn.commit();
    conn.close();
} catch (SQLException e) {
    e.printStackTrace();
    result = result + e.toString() + "\n";
}
    return result;
}
}

```

---

This class defines an IMSDataSource Object and specifies the required parameters, depending on a type 2 or type 4 connectivity. Since the application allows only queries against the database, the sample doesn't really need a commit on the Connection.

To set up the transactions, you have to follow the usual steps for a new IMS Java transaction. The JMP region must be configured correctly for the use with the IMS Universal Drivers in IMS Java transactions. The region startup procedure contains the ENVIRON and JVMOPMAS parameter which specifies the members in the IMS PROCLIB. These configuration files must contain the IMS Universal JDBC Driver jar file in the class path and also the metadata class file (here this would be the generated AUTPSB11.jar). If you use type-2 connectivity you also have to specify the location of the native Drivers for type 2 connectivity.







## Scenario 3 - Writing DL/I and mixed applications

In this chapter we show how you can access your IMS databases using the IMS Universal DL/I driver and the IMS Universal DB Resource adapter with the CCI programming approach. This chapter shows how you use the IMS Universal DL/I Driver to write standalone DL/I applications in Java and if you want using batch methods. It also shows how to use the IMS Universal DB Resource Adapter with the CCI programming model to write applications using mixed SQL and DL/I syntax in managed or non-managed environments.

This chapter contains the following:

- ▶ Writing applications with the IMS Universal DL/I Driver
- ▶ Writing application with the IMS Universal DB Resource Adapter and the CCI programming approach

## 8.1 Writing applications with the IMS Universal DL/I Driver

You can use the IMS Universal DL/I Driver for writing applications in Java by using the DL/I syntax to access IMS databases. As with all the IMS Universal drivers they can be used by remote applications, using the type 4, and local applications, using the type 2. We also show how you can batch up your accesses where appropriate to improve the performance when using the IMS Universal DL/I driver.

### 8.1.1 Accessing IMS data with the IMS Universal DL/I driver

In order to execute DL/I calls from your IMS Universal driver application, you must have connected to an IMS database. This is done by using the PSB interface which is part of the `com.ims.dli` package. What needs to be provided to the interface and an example of the code is shown below.

#### Connecting to an IMS database

The connection properties are passed using an `IMSConnectionSpec` instance which is created by calling the `createIMSConnectionSpec` method in the `IMSConnectionSpecFactory` class. You need to set the following connection properties for the `IMSConnectionSpec` instance:

- ▶ **datastoreName** as the name of the datastore to be accessed.
  - For type-4(remote) connectivity, the datastore name must either match the that datastore name defined in the active ODBM CSLDCxxx proclib member or be blank. If the datastore name is defined using the `ALIAS(NAME=` parameter, you must use the name specified as the alias. If the datastore name is left blank or not supplied, IMS connects to any available ODBM as it assumes that data sharing is enabled among all datastores defined to ODBM.
  - For type-2(local) connectivity, the datastore name is set to the name of the Database Resource Adapter (DRA) specified when defining the DFSPRP member whose name is `DFSxxxx0`, where `xxxx` is the datastore name.
- ▶ **metadataURL** is the fully qualified name of the JAVA metadata class generated by the IMS Enterprise Suite `DLIModel` utility plug-in. The URL must be prefixed with `class://`, in the application example below it is defined as `"class://samples.ims.openDb.AUTPSB11DatabaseView"`
- ▶ **portNumber** is the Port Number of IMS Connect
  - For type-4(remote) connectivity, this is the TCP/IP server port number used to communicate with IMS Connect. It is defined using the `DRDAPORT` parameter of the `ODACCESS` statement in the integrated IMS Connect configuration proclib member.
  - For type-2(local) connectivity do not set this property.
- ▶ **datastoreServer** is the IP address of the IMS Connect host.
  - For type-4(remote) connectivity, this is the name or IP address of the datastore server (IMS Connect). In the sample application below we have specified the name, `wtsc63.itso.ibm.com` but we could have used its IP address, `9.12.6.70`, instead
  - For type-2(local) connectivity, do not set this property.
- ▶ **driverType** is used to specify the type of driver connectivity the application will be using.
  - For type-4 connectivity the value must be `IMSConnectionSpec.DRIVER_TYPE_4` or `4`
  - For type-2 connectivity `IMSConnectionSpec.DRIVER_TYPE_2` or `2`

- ▶ **sslConnection** is an optional property used to indicate if Secure Socket Layer (SSL) is being used for the connection.
  - For type-4(remote) connectivity set it to true to enable SSL or false otherwise.
  - For type-2(local) do not set the property.
- ▶ **loginTimeout** is an optional property used to specify the number of seconds the driver will wait for a response from the server before timing out.
  - For type-4 connectivity set the value to a non-negative integer for the number of seconds. Setting it to 0 will allow the driver to wait forever.
  - For type-2 connectivity, do not set this property.
- ▶ **user** is the username for authentication to IMS Connect. Do not set this property for type-2 connectivity.
- ▶ **password** is the password used for the connection. Do not set this property for type-2 connectivity.

Example 8-1 shows how to specify the values.

*Example 8-1 IMSConnectionSpec properties*

---

```
// establish a database connection
IMSConnectionSpec connSpec
= IMSConnectionSpecFactory.createIMSConnectionSpec();
connSpec.setDatastoreName("IMS2");
connSpec.setDatastoreServer("wtsc63.itso.ibm.com");
connSpec.setPortNumber(5555);
connSpec.setMetadataURL(
    "class://samples.ims.openDb.AUTPSB11DatabaseView");
connSpec.setUser("IMS2R");
connSpec.setPassword("password");
connSpec.setDriverType(IMSConnectionSpec.DRIVER_TYPE_4);
```

---

Having set your connection properties you now need to pass the connection properties to the PSBFactory class to create a PSB instance. Once the PSB instance is successfully created you will have a connection to the database. See Example 8-2.

*Example 8-2 Creating a PSB instance*

---

```
psb = PSBFactory.createPSB(connSpec);
```

---

## 8.1.2 Retrieving Data Using the IMS Universal DL/I drivers

In a traditional, that is COBOL, PL/1, Assembler etc, IMS application to delete, insert, replace or retrieve data would make DL/I calls. Using the IMS Universal DL/I driver to perform those same functions, you invoke methods.

These methods are defined in the following interfaces:

- ▶ You have seen above how the PSB instance is created to connect to the database and that PSB interface can be used to obtain a handle on any Program Communication Block (PCB) within the PSB. The PCB handle is used to access the particular IMS database that is referenced by that PCB.
- ▶ The PCB interface represents a cursor position in an IMS database. The PCB interface supports DL/I call functions, including Get Unique (GU), Get Next (GN), Get Next within Parent (GNP), Insert (ISRT), Replace (REPL) and Delete (DLET). The PCB interface can obtain an unqualified list of Segment Search Arguments (SSAs) and perform batch

retrieve, update and delete operations. You can also use the PCB interface to return the Application Interface Block (AIB) associated with the most recent DL/I call.

- ▶ The SSAList interface represents a list of SSAs used to specify which segments to target in a particular DL/I call. You use the SSAList to construct the SSAs and to set any command codes and lock classes the segments referenced. You can set an initial qualification statement and append additional qualifiers based on the values of the segment fields to restrict which segments will be targeted. Each SSA in the list can be qualified or unqualified. You are also able to specify which fields from segments are to be returned by a retrieve call.
- ▶ The Path interface represents a database record for the purpose of a retrieval or update operation. It can be viewed as the concatenation of all of the segment instances in a specific hierarchic path, starting from the highest level segment that is nearest to the root segment to the lowest level segment. Use the path interface to set or retrieve the value of any segment field that is located in the hierarchic path.
- ▶ The PathSet interface provides access to a collection of Path objects that are returned by a batch retrieve operation.
- ▶ The AIB interface and the database PCB(DBPCB) interface return useful information returned by IMS as a result of a DL/I call
- ▶ The GSAMPCB interface represents a GSAM PCB and is essentially a cursor position in a GSAM database. The interface provides data access to GSAM databases with calls similar to DL/I calls.
- ▶ The RSA interface represents a GSAM database record search argument that is the key to a cursor position in the GSAM database.

Example 8-3 shows how to get a handle on the AUTOLPCB PCB specified in the AUTPSB11 PSB and uses the AUTOLDB database specified in that PCB to show how an unqualified SSAList can be constructed. The example gets a path instance by using the SSAList and calling the `getPathForRetrievalReplace` method and uses the `getUnique` method to return a Path that consists of all fields in the STOCSALE segment.

*Example 8-3 Obtaining a PCB handle and specifying SSAs using the SSAList interface*

---

```
pcb = psb.getPCB("AUTLPCB");
SSAList ssaList = pcb.getSSAList("DEALER","STOCSALE");
Path path = ssaList.getPathForRetrievalReplace();
pcb.getUnique(path, ssaList,false);
```

---

In Example 8-3 the `ssaList` represents all segments along the hierarchic path from the topmost segment, DEALER, to the lowest segment, STOCSALE. The `ssaList` will look like this

```
DEALERbbb
MODELbbbb
STOCKbbbb
STOCSALEb
```

An SSAList can be qualified to filter the segments in the hierarchic path to be retrieved or updated. Generally the steps to create a qualified SSAList are:

1. Use the `getSSAList` method to create an unqualified SSAList from the PCB.
2. Use the `addInitialQualification` method to specify the initial search criteria for a segment in the SSAList returned by a `getSSAList` method. You can use one `addInitialQualification` statement for each segment represented in the SSAList. If you use more than one `addInitialQualification` statement for a segment an exception error will be thrown. The relational operator parameter in the `addInitialQualification` method indicates the

conditional criteria that the segment must meet in order to be qualified. The valid operators are

EQUALS  
GREATER\_OR\_EQUAL  
GREATOR\_THAN  
LESS\_OR\_EQUAL  
LESS\_THAN  
NOT\_EQUAL

3. To add additional search criteria to a segment in the SSAList, you have to use the `appendQualification` method. The Boolean Operator (BooleanOp) parameter in the `appendQualification` method statement says how this qualification is logically connected to the previous qualification. The valid BooleanOp values are

AND  
OR  
INDEPENDENT\_AND

4. Segment SSAs in the SSAList can also be qualified by setting DL/I command codes and lock classes. Supported DLI command codes include

CC\_C: The C command code (Concatenated Key). Use the `addConcatenatedKey` method to add the concatenated key to the segments SSA in the SSAList.  
CC\_D: The D command code (path call).  
CC\_F: The F command code (first occurrence).  
CC\_L: The L command code (last occurrence).  
CC\_N: The N command code (path call ignore).  
CC\_P: The P command code (set parentage).  
CC\_U: The U command code (maintain position at this level).  
CC\_V: The V command code (maintain position at this and all higher levels).

The lock class is used to prevent another application from updating a segment until your program has reached a commit point. Use the `addLockClass` method to add a lock class to a segment. The supported lock class letters are "A" to "J". The behavior of a lock class is the same as using a Q command code with the lock class letter.

Example 8-4 shows how to specify and use a qualified SSAList with just an initial qualification statement to retrieve data.

*Example 8-4 Qualified SSAList with initial qualification*

---

```
SSAList ssaList = pcb.getSSAList("DEALER","STOCSALE");
ssaList.addInitialQualification("MODEL","MAKE",SSAList.EQUALS,"FORD");
ssaList.markFieldForRetrieval("STOCK","COLOR",true);
Path path = ssaList.getPathForRetieveReplace();
pcb.getUnique(path, ssaList, false);
```

---

The ssaList for the above example is:

```
DEALERbbb
MODELbbb(MAKEbbbEQFORDbbbbbbb)
STOCKbbb*D
STOCSALEb
```

The data returned by the above example where MAKE =Ford would be the COLOR field in the STOCK segment and all the fields of segment STOCSALE as by default IMS returns all fields of the lowest level segment specified.

Example 8-5 shows how we can add an extra qualification to the SSAList for the MODEL SSA in Example 8-4.

*Example 8-5 Qualified SSAList with multiple qualifications for one SSA*

---

```
SSAList ssaList = pcb.getSSAList("DEALER","STOCSALE");
ssaList.addInitialQualification("MODEL","MAKE",SSAList.EQUALS,"FORD");
ssaList.appendQualification("MODEL",SSAList.AND,"YEAR",SSAList.Greater_than,
2002);
ssaList.marFieldForRetrieval("STOCK","COLOR",true);
.
.
```

---

Example 8-6 shows how we code a qualified SSAList with the command code CC\_P (set parent). DL/I by default sets parent at the lowest level in the path of segments returned from the SSAs specified in the SSAList when a GU or GN call is issued.

*Example 8-6 Qualified SSA using a command code*

---

```
SSAList ssaList = pcb.getSSAList("DEALER","STOCSALE");
ssaList.addInitialQualification("MODEL","MAKE",SSAList.EQUALS,"FORD");
ssaList.addCommandCode("MODEL",SSAList.CC_P);
ssaList.appendQualification("MODEL",SSAList.AND,"YEAR",SSAList.Greater_than,
2002);
ssaList.marFieldForRetrieval("STOCK","COLOR",true);
.
.
```

---

The SSAList for our example is:

```
DEALERbbb
MODELbbb*P(MAKEbbbbbEQFORDbbbbbb&YEARbbbbGT2002)
STOCKbbbb
STOCSALEb
```

The IMS Universal DL/I driver provides support for data retrieval that mirrors DL/I semantics.

You have seen above, how to obtain an SSAList instance from the PCB instance representing the database, how you can add qualification statements to the SSAList, specify the segment fields to retrieve and get a Path instance by using the SSAList instance and calling the getPathForRetrieveReplace method causing all the fields to be retrieved in the resulting Path object that were marked for retrieval. Mention was also made of the getUnique method and this and other DL/I retrieve methods are shown in Table 8-1. There is one more thing to say about marking fields for retrieval, by default all the fields in the lowest level segment specified in the SSAList are returned. However, if any individual fields in that segment are marked for retrieval only those will be returned.

*Table 8-1 Methods for DL/I retrieve from the PBC interface*

JAVA API for DL/I retrieve method	Usage
getUnique	Retrieves a specific unique segment. Method provides the same functionality as a DL/I GU database call. If the isHoldCall parameter is set to true method behaves as a DL/I GHU database call.

JAVA API for DL/I retrieve method	Usage
getNext	Retrieves the next segment in the path. Method provides the same functionality as a DL/I GN database call. If the isHoldCall parameter is set to true, the method behaves as the DL/I GHN database call does.
getNextWithinParent	Retrieves the next segment within the same parent. Method provides the same functionality as the DL/I GNP database call. If the isHoldCall parameter is set to true the method behaves as a DL/I GHNP database call would.
batchRetrieval	Retrieves multiple segments with a single call.

### Methods for retrieving and converting data types

Data can be retrieved and converted from how it is defined in the database metadata to the type your Java application is expecting by using the Path interface. If the application was using either the IMS Classic JDBC or IMS Universal JDBC drivers you would be able to use the ResultSet interface.

Table 8-2 shows the get methods that can be used in the ResultSet interface or the Path interface for accessing certain data of a certain Java data type.

Table 8-2 The get methods

ResultSet.getXXX or Path.getXXXX Method	No truncation or data loss	Legal without data integrity
getBytes	TINYINT	SMALLINT, INTEGER, BIGINT, FLOAT, DOUBLE, BIT, CHAR, VARCHAR, PACKEDDECIMAL, ZONEDDECIMAL
getShort	SMALLINT	TINYINT, INTEGER, BIGINT, FLOAT, DOUBLE, BIT, CHAR, VARCHAR, PACKEDDECIMAL, ZONEDDECIMAL
getInt	INTEGER	TINYINT, SMALLINT, BIGINT, FLOAT, DOUBLE, BIT, CHAR, VARCHAR, PACKEDDECIMAL, ZONEDDECIMAL
getLong	BIGINT	TINYINT, SMALLINT, INTEGER, FLOAT, DOUBLE, BIT, CHAR, VARCHAR, PACKEDDECIMAL, ZONEDDECIMAL
getFloat	FLOAT	TINYINT, SMALLINT, INTEGER, BIGINT, DOUBLE, BIT, CHAR, VARCHAR, PACKEDDECIMAL1, ZONEDDECIMAL1
getDouble	DOUBLE	TINYINT, SMALLINT, INTEGER, BIGINT, FLOAT, BIT, CHAR, VARCHAR, PACKEDDECIMAL, ZONEDDECIMAL
getBoolean	BIT	TINYINT, SMALLINT, INTEGER, BIGINT, FLOAT, DOUBLE, CHAR, VARCHAR, PACKEDDECIMAL, ZONEDDECIMAL
getString	CHAR VARCHAR	TINYINT, SMALLINT, INTEGER, BIGINT, FLOAT, DOUBLE, BIT, PACKEDDECIMAL, ZONEDDECIMAL, BINARY, DATE, TIME, TIMESTAMP
getBigDecimal	PACKEDDECIMAL ZONEDDECIMAL	TINYINT, SMALLINT, INTEGER, BIGINT, FLOAT, DOUBLE, BIT, CHAR, VARCHAR
getClob	CLOB	all others result in an exception

ResultSet.getXXX or Path.getXXX Method	No truncation or data loss	Legal without data integrity
getBytes	BINARY	all others result in an exception
getDate	DATE	CHAR, VARCHAR, TIMESTAMP
getTime	TIME	CHAR, VARCHAR, TIMESTAMP
getTimestamp	TIMESTAMP	CHAR, VARCHAR, DATE, TIME

**Notes:**

- ▶ PACKEDDECIMAL and ZONEDDECIMAL are data type extensions for the IMS Classic JDBC, IMS Universal JDBC and IMS Universal DL/I drivers. All other types are standard SQL types defined in SQL92. PACKEDDECIMAL and ZONEDDECIMAL do not support the Sign Leading or Sign Separate modes. For these two data types data is always stored with the Sign Trailing method.
- ▶ The CLOB data type is only supported for the storage and retrieval of XML. This is currently only supported by the IMS Classic JDBC driver.

### 8.1.3 Inserting data using the IMS Universal DL/I driver

There are two methods, the create and the insert, that can be used in the PCB interface to add a new segment into the database. Used with the IMS Universal DL/I driver the two methods provide functionality similar to the DL/I ISRT call. The insert method returns an IMS status code indicating the result of the operation, while the create method returns a count of the number of segments inserted (this currently will always return 1). The 2 methods provide defaults for any DBD search fields that are not explicitly set, as normal character based fields are set to blanks and numeric based fields are set to zero. Non-DBD fields are set to hex '00'. If a key field is not set an exception is thrown. If a key field is broken down into multiple sub fields in the DBD you must use the key field to set the value, if just the sub fields are allocated values an exception will be thrown. You can specify values for both but make sure they line up identically. If you provide values for both the key field and the sub fields and the values are not the same the actual value of the key field for the segment inserted into the database will depend on the order of the path.setString statements in the application, last overrides first.

Example 8-7 shows how to use the create and insert methods to add a new MODEL and two ORDER segments into the AUTODB where the DLRNO is '1234'.

*Example 8-7 Create and Insert method*

---

```

SSAList ssaList = pcb.getSSAList("DEALER","ORDER");
ssaList.addInitialQualification("DEALER","DLRNO",SSAList.EQUALS,"1234");
Path path = ssaList.getPathForInsert("MODEL");
path.setString("MODEL", "MODKEY", "Lotus      Evora      2010");
path.setString("MODEL", "MSRP", "65000");
path.setString("ORDER", "ORDNBR", "258921");
path.setString("ORDER", "LASTNME", "Thilo");
path.setString("ORDER", "FIRSTNME", "LIEDLOFF");
path.setString("ORDER", "DATE", "04-10-2010");
int i = pcb.create(path, ssaList); // returns i = 1 if successful

SSAList ssaList = pcb.getSSAList("ORDER");
path = ssaList.getPathForInsert("ORDER");

```



```

path.setString("ORDNBR", "258930");
path.setString("LASTNME", "Angelique");
path.setString("FIRSTNME", "GREENHAW");
path.setString("DATE", "04-09-2010");
short status = pcb.insert(path,ssaList); // returns an IMS status code

```

---

### 8.1.4 Updating data with the IMS Universal DL/I driver

This section shows how to update data with the IMS Universal DL/I driver.

**Note:** To persist changes made to the database, your application must call the PSB commit method prior to deallocating the PSB. Failure to do so will result in changes being rolled back to the last point commit was called or the start of the application if commit was never called

The replace method is used to amend data in a segment that exists in the database. Example 8-8 shows how the ORDER segment that would have been inserted in the previous example has the LASTNME and FIRSTNME fields corrected as the values were inserted the wrong way round.

*Example 8-8 The replace method*

```

SSAList ssalist = pcb.getSSAList("DEALER","ORDER");
ssaList.addInitialQualification("DEALER","DLRNO",SSAList.EQUALS,"6788");
ssaList.addInitialQualification("MODEL","MODKEY",
SSAList.EQUALS,"Lotus      Evora      2010");
ssaList.addInitialQualification("ORDER","ORDNBR",SSAList.EQUALS,"258921");
Path path = ssaList.getPathForRetrieveReplace();
if(pcb.getUnique(path, ssaList,true)) {
    path.setString("LASTNME", "LIEDLOFF");
    path.setString("FIRSTNME", "Thilo");
    pcb.replace(path);
}

```

---

### 8.1.5 Deleting data with the IMS Universal DL/I driver

The delete method is used to remove existing segments from the database. Functionally the delete method in the IMS Universal DL/I driver is similar to the DL/I DLET call. The delete of a segment will cause all of its child segments to be deleted as well. The delete call must be preceded by a HOLD operation. The Hold operation can be done using any one of either a getUnique, getNext or getNextWithinParent method call. An IMS status code is returned by the delete method which indicates the result of the DL/I operation. When a Path of segments has been retrieved by the HOLD operation you can either delete all segments in that path or a subset of them. To delete them all you would call the delete method with no parameters. To delete segments from a point on the Path, other than the top, to the bottom, use the delete method that takes an SSAList argument and pass in an unqualified SSAList for the segment where you want deletion to begin. If you use a qualified SSAList an exception will be thrown.

Example 8-9 demonstrates how to delete all segments in a PATH. It shows how to use delete to selectively remove all MODEL segments and its dependent segments STOCK, STOC SALE, ORDER and SALES where the DLRNO is '222222', MAKE is 'FORD' and the MODEL is 'FOCUS'.

*Example 8-9 Deleting all segments in the path*


---

```

SSAList ssaList = pcb.getSSAList("DEALER","STOCK");
ssaList.addInitialQualification("DEALER","DLRNO",SSAList.EQUALS,"1234");
ssaList.addInitialQualification("MODEL","MAKE",SSAList.EQUALS,"FORD");
ssaList.addCommandCode("MODEL",SSAList.CC_D);
Path path = ssaList.getPathForRetrieveReplace();
if (pcb.getUnique(path, ssaList, true)) {
    if (path.getString("MODEL", "MODEL").equals("FOCUS")) {
        pcb.delete();
    }
}
while (pcb.getNext(path, ssaList, true)) {
    if (path.getString("MODEL", "MODEL").equals("FOCUS")) {
        pcb.delete();
    }
}

```

---

Example 8-10 shows how to delete with an unqualified SSAList. The delete will remove all STOCK segments and its logical child segment STOC SALE where the DLRNO is '222222', the MAKE is 'FORD' and the MODEL is 'FOCUS'.

*Example 8-10 Deleting segments with an unqualified ssalist*


---

```

SSAList ssaList = pcb.getSSAList("DEALER","STOCK");
ssaList.addInitialQualification("DEALER","DLRNO",SSAList.EQUALS,"1234");
ssaList.addInitialQualification("MODEL","MAKE",SSAList.EQUALS,"FORD");
ssaList.markAllFieldsForRetrieval("MODEL", true);
Path path = ssaList.getPathForRetrieveReplace();
SSAList stockSSAList = pcb.getSSAList("STOCK");
if (pcb.getUnique(path, ssaList, true)) {
    if (path.getString("MODEL", "MODEL").equals("FOCUS")) {
        pcb.delete(stockSSAList);
    }
}
while (pcb.getNext(path, ssaList, true)) {
    if (path.getString("MODEL", "MODEL").equals("FOCUS")) {
        pcb.delete(stockSSAList);
    }
}

```

---

Example 8-11 is a total application showing all the topics previously talked about in this chapter. This is downloadable from the file IMS Universal DLI Driver sample.zip as described in Appendix D, "Additional material" on page 249.

*Example 8-11 dlitest1 - A Complete DL/I application*


---

```

package dlitest1;
import com.ibm.ims.dli.*;
public class DLIprogram {
    public static void main(String[] args) {
        PSB psb = null;
        PCB pcb = null;
        SSAList ssaList = null;
        Path path = null;
        PathSet pathSet = null;
        try {
            // establish a database connection
            IMSConnectionSpec connSpec
                = IMSConnectionSpecFactory.createIMSConnectionSpec();

```

---

```

connSpec.setDatastoreName("IMSZ");
connSpec.setDatastoreServer("wtsc63.itso.ibm.com");
connSpec.setPortNumber(5555);
connSpec.setMetadataURL("class://samples.ims.openDb.AUTPSB11DatabaseView");
connSpec.setUser("userid");
connSpec.setPassword("password");
connSpec.setDriverType(IMSConnectionSpec.DRIVER_TYPE_4);

psb = PSBFactory.createPSB(connSpec);
System.out.println("***** Created a connection to the IMS database");
pcb = psb.getPCB("AUTOLPCB");
System.out.println("***** Created PCB object");
// specify the segment search arguments
ssaList = pcb.getSSAList("DEALER", "STOCK");
// add the initial qualification
// specify the fields to retrieve
ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
ssaList.markFieldForRetrieval("DEALER", "DLRNAME", true);
ssaList.markFieldForRetrieval("DEALER", "CITY", true);
ssaList.markFieldForRetrieval("DEALER", "ZIP", true);
ssaList.markAllFieldsForRetrieval("MODEL", true);
ssaList.markAllFieldsForRetrieval("STOCK", true);
ssaList.markFieldForRetrieval("STOCK", "LOT", false);
System.out.println("***** Created SSAList object");
// obtain a Path containing the segments
// that match the SSAList criteria
path = ssaList.getPathForRetrieveReplace();
System.out.println("***** Created Path object");
// issue a DL/I GU call to retrieve the first segment on the Path
if (pcb.getUnique(path, ssaList, true)) {
    System.out.println("Dealer Number: " + path.getString("DEALER", "DLRNO"));
    System.out.println("Dealer Name: " + path.getString("DEALER", "DLRNAME"));
    System.out.println("City: " + path.getString("DEALER", "CITY"));
    System.out.println("ZIP: " + path.getString("DEALER", "ZIP"));
    System.out.println("Type of Model: " + path.getString("MODEL", "MODTYPE"));
    System.out.println("Manufacturer: " + path.getString("MODEL", "MAKE"));
    System.out.println("Model: " + path.getString("MODEL", "MODEL"));
    System.out.println("Year: " + path.getString("MODEL", "YEAR"));
    System.out.println("MSRP: " + path.getString("MODEL", "MSRP"));
    System.out.println("# in Stock: " + path.getString("MODEL", "COUNT1"));
    System.out.println("VIN Number: " + path.getString("STOCK", "STKVIN"));
    System.out.println("Colour: " + path.getString("STOCK", "COLOR"));
    System.out.println("Price: " + path.getString("STOCK", "PRICE"));
    System.out.println("Warrenty: " + path.getString("STOCK", "WRNTY"));
}
// issue multiple DL/I GN calls until
// there are no more segments to retrieve
while (pcb.getNext(path, ssaList, true)) {
    System.out.println("Dealer Number: " + path.getString("DEALER", "DLRNO"));
    System.out.println("Dealer Name: " + path.getString("DEALER", "DLRNAME"));
    System.out.println("City: " + path.getString("DEALER", "CITY"));
    System.out.println("ZIP: " + path.getString("DEALER", "ZIP"));
    System.out.println("Type of Model: " + path.getString("MODEL", "MODTYPE"));
    System.out.println("Manufacturer: " + path.getString("MODEL", "MAKE"));
    System.out.println("Model: " + path.getString("MODEL", "MODEL"));
    System.out.println("Year: " + path.getString("MODEL", "YEAR"));
    System.out.println("MSRP: " + path.getString("MODEL", "MSRP"));
    System.out.println("# in Stock: " + path.getString("MODEL", "COUNT1"));
    System.out.println("VIN Number: " + path.getString("STOCK", "STKVIN"));
    System.out.println("Colour: " + path.getString("STOCK", "COLOR"));
}

```

```

        System.out.println("Price:          "+ path.getString("STOCK", "PRICE"));
        System.out.println("Warrenty:       "+ path.getString("STOCK", "WRNTY"));
    }
    // Insert a new DEALER, MODEL & 3 STOCK segments
    ssaList = pcb.getSSAList("DEALER", "STOCK");
    path = ssaList.getPathForInsert("DEALER");
    path.setString("DLRNO", "7575");
    path.setString("DLRNAME", "IBM Super Systems");
    path.setString("CITY", "Portsmouth");
    path.setString("ZIP", "PO6 3AU");
    path.setString("PHONE", "0239256");
    path.setString("MODTYPE", "MF");
    path.setString("MODKEY", "IBM          Z10 GT      2010");
    path.setString("MSRP", "12500");
    path.setString("COUNT1", "3");
    path.setString("STKVIN", "VT11234677098557729C");
    path.setString("COLOR", "Blue");
    path.setString("PRICE", "99999");
    path.setString("LOT", "BS12345678");
    path.setString("WRNTY", "Y");
    short status = pcb.insert(path, ssaList);
    if(status == IMSStatusCodes.BLANKS){
        System.out.println(" Insert of STOCK Segment 1 Successful");
    } else {
        System.out.println(" Insert of STOCK Segment 1 FAILED");
    }
    ssaList = pcb.getSSAList("STOCK");
    path = ssaList.getPathForInsert("STOCK");
    path.setString("STKVIN", "VT11234677098557730C");
    path.setString("COLOR", "Deep Blue");
    path.setString("PRICE", "99999");
    path.setString("LOT", "BS12345679");
    path.setString("WRNTY", "Y");
    status = pcb.insert(path, ssaList);
    if(status == IMSStatusCodes.BLANKS){
        System.out.println(" Insert of STOCK Segment 2 Successful");
    } else {
        System.out.println(" Insert of STOCK Segment 2 FAILED");
    }
    path.setString("STKVIN", "VT11234677098557731C");
    path.setString("COLOR", "Light Blue");
    path.setString("PRICE", "99999");
    path.setString("LOT", "BS12345680");
    path.setString("WRNTY", "Y");
    int i = pcb.insert(path, ssaList);
    if(i > 0){
        System.out.println(" Insert of STOCK Segment 3 Successful");
    } else {
        System.out.println(" Insert of STOCK Segment 3 FAILED");
    }
    }
    psb.commit();
    ssaList = pcb.getSSAList("DEALER", "STOCK");
    // add the initial qualification
    ssaList.addInitialQualification("DEALER", "DLRNO",
        ssaList.EQUALS, "7575");
    ssaList.addCommandCode("DEALER", ssaList.CC_D);
    ssaList.addCommandCode("MODEL", ssaList.CC_D);
    ssaList.addCommandCode("MODEL", ssaList.CC_P);
    // specify the fields to retrieve
    ssaList.markAllFieldsForRetrieval("DEALER", true);

```

```

ssaList.markAllFieldsForRetrieval("MODEL", true);
ssaList.markAllFieldsForRetrieval("STOCK", true);
System.out.println("**** Created SSAList object");
// obtain a Path containing the segments
// that match the SSAList criteria
path = ssaList.getPathForRetrieveReplace();
System.out.println("**** Created Path object");
// issue a DL/I GU call to retrieve the
// first segment on the Path
if (pcb.getUnique(path, ssaList, true)) {
    System.out.println("DEALER NUMBER:    "+ path.getString("DEALER", "DLRNO"));
    System.out.println("DEALER NAME:      "+ path.getString("DEALER", "DLRNAME"));
    System.out.println("CITY:           "+ path.getString("DEALER", "CITY"));
    System.out.println("ZIP:            "+ path.getString("DEALER", "ZIP"));
    System.out.println("PHONE NUMBER:   "+ path.getString("DEALER", "PHONE"));
    System.out.println("TYPE of MODEL:  "+ path.getString("MODEL", "MODTYPE"));
    System.out.println("MAKE MODEL YEAR: "+ path.getString("MODEL", "MODKEY"));
    System.out.println("MSRP:           "+ path.getString("MODEL", "MSRP"));
    System.out.println("Number in Stock: "+ path.getString("MODEL", "COUNT1"));
    System.out.println("VIN NUMBER:     "+ path.getString("STOCK", "STKVIN"));
    System.out.println("COLOR:          "+ path.getString("STOCK", "COLOR"));
    System.out.println("PRICE:          "+ path.getString("STOCK", "PRICE"));
    System.out.println("LOT:           "+ path.getString("STOCK", "LOT"));
    System.out.println("WARRENTY?:      "+ path.getString("STOCK", "WRNTY"));
}
while (pcb.getNextWithinParent(path, ssaList, true)) {
    System.out.println("VIN NUMBER:     "+ path.getString("STOCK", "STKVIN"));
    System.out.println("COLOR:          "+ path.getString("STOCK", "COLOR"));
    System.out.println("PRICE:          "+ path.getString("STOCK", "PRICE"));
    System.out.println("LOT:           "+ path.getString("STOCK", "LOT"));
    System.out.println("WARRENTY?:      "+ path.getString("STOCK", "WRNTY"));
}
ssaList = pcb.getSSAList("DEALER", "STOCK");
// add the initial qualification
ssaList.addInitialQualification("DEALER", "DLRNO", SSAList.EQUALS, "7575");
// specify the fields to retrieve
ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
ssaList.markAllFieldsForRetrieval("MODEL", true);
ssaList.markAllFieldsForRetrieval("STOCK", true);
ssaList.addCommandCode("MODEL", SSAList.CC_P);
// obtain a Path containing the segments
// that match the SSAList criteria
path = ssaList.getPathForRetrieveReplace();
System.out.println("**** Get DLRNO 7575 ready for DELETE ****");
String blue = "Blue";
// issue a DL/I GU call to retrieve
if (pcb.getUnique(path, ssaList, true)) {
    String color = path.getString("STOCK", "COLOR").trim();
    System.out.println("Color is : '"+color+"'");
    if (color.equals(blue)) {
        System.out.println("STKVIN = "+ path.getString("STOCK", "STKVIN"));
        System.out.println("This one is Blue do not delete");
    } else {
        SSAList stockSSAList = pcb.getSSAList("STOCK");
        status = pcb.delete(stockSSAList);
        if (status == IMSSStatusCodes.BLANKS) {
            System.out.println(" DELETE of STOCK Segment "
+ path.getString("STOCK", "STKVIN")+" Successful");
        } else {
            System.out.println(" DELETE of STOCK Segment "

```

```

        + path.getString("STOCK", "STKVIN")+" FAILED");
    }
}
}
while (pcb.getNextWithinParent(path, ssaList, true)) {
    String color = path.getString("STOCK","COLOR").trim();
    System.out.println("Color is : '"+color+"'");
    if (color.equals(blue)){
        System.out.println("This one is Blue so do not delete");
    } else {
        SSAList stockSSAList = pcb.getSSAList("STOCK");
        status = pcb.delete(stockSSAList);
        if(status == IMSStatusCodes.BLANKS) {
            System.out.println(" DELETE of STOCK Segment "
+ path.getString("STOCK", "STKVIN")+" Successful");
        } else {
            System.out.println(" DELETE of STOCK Segment "
+ path.getString("STOCK", "STKVIN")+" FAILED");
        }
    }
}
}
System.out.println(" All STOCK segments not COLOR = BLUE deleted");
ssaList = pcb.getSSAList("DEALER");
// add the initial qualification
ssaList.addInitialQualification("DEALER", "DLRNO",
    SSAList.EQUALS, "7575");
// specify the fields to retrieve
ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
path = ssaList.getPathForRetrieveReplace();
if (pcb.getUnique(path, ssaList, true)==true) {
    status = pcb.delete();
}
if(status == IMSStatusCodes.BLANKS) {
    System.out.println(" DEALER DLRNO = '7575' and dependents deleted");
} else {
    System.out.println(" DELETE PROBLEM");
}
psb.commit();

// specify the segment search arguments
ssaList = pcb.getSSAList("DEALER", "STOCK");
// add the initial qualification
ssaList.addInitialQualification("DEALER", "DLRNO",
    SSAList.EQUALS, "1234");
ssaList.addCommandCode("MODEL",SSAList.CC_D);
ssaList.addCommandCode("MODEL",SSAList.CC_P);
// specify the fields to retrieve
ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
ssaList.markFieldForRetrieval("DEALER", "DLRNAME", true);
ssaList.markFieldForRetrieval("DEALER", "CITY", true);
ssaList.markAllFieldsForRetrieval("MODEL", true);
ssaList.markFieldForRetrieval("MODEL", "MAKE", false);
ssaList.markFieldForRetrieval("MODEL", "YEAR", false);
ssaList.markAllFieldsForRetrieval("STOCK", true);
System.out.println("***** Created SSAList object");
// obtain a Path containing the segments that match the SSAList criteria
path = ssaList.getPathForRetrieveReplace();
System.out.println("***** Created Path object");
// issue a DL/I GU call to retrieve the segments on the Path
if (pcb.getUnique(path, ssaList, true)){

```

```

System.out.println("DEALER NUMBER:      "+ path.getString("DEALER", "DLRNO"));
System.out.println("DEALER NAME:        "+ path.getString("DEALER", "DLRNAME"));
System.out.println("CITY:               "+ path.getString("DEALER", "CITY"));
System.out.println("MAKE, MODEL & YEAR:  "+ path.getString("MODEL", "MODKEY"));
System.out.println("MSRP:               "+ path.getString("MODEL", "MSRP"));
System.out.println("VIN NUMBER:         "+ path.getString("STOCK", "STKVIN"));
System.out.println("COLOR:              "+ path.getString("STOCK", "COLOR"));
}
// If data was retrieved update the values in fields MSRP in the MODEL
// segment and COLOR in the STOCK segment
if (pcb.getUnique(path, ssaList, true))
    path.setString("MSRP", "45999");
path.setString("COLOR", "Green");
pcb.replace(path);
if (pcb.getUnique(path, ssaList, true)){
System.out.println("DEALER NUMBER:      "+ path.getString("DEALER", "DLRNO"));
System.out.println("DEALER NAME:        "+ path.getString("DEALER", "DLRNAME"));
System.out.println("CITY:               "+ path.getString("DEALER", "CITY"));
System.out.println("MAKE, MODEL & YEAR:  "+ path.getString("MODEL", "MODKEY"));
System.out.println("MSRP:               "+ path.getString("MODEL", "MSRP"));
System.out.println("VIN NUMBER:         "+ path.getString("STOCK", "STKVIN"));
System.out.println("COLOR:              "+ path.getString("STOCK", "COLOR"));
}
// Rollback the updates
psb.rollback();
// Get the data from the path for DLRNO 1234 to show that the
// updates have been rolled back.
ssaList = pcb.getSSAList("DEALER", "STOCK");
// add the initial qualification
ssaList.addInitialQualification("DEALER", "DLRNO",
    SSAList.EQUALS, "1234");
// specify the fields to retrieve
ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
ssaList.markFieldForRetrieval("DEALER", "DLRNAME", true);
ssaList.markFieldForRetrieval("DEALER", "CITY", true);
ssaList.markAllFieldsForRetrieval("MODEL", true);
ssaList.markFieldForRetrieval("MODEL", "MAKE", false);
ssaList.markFieldForRetrieval("MODEL", "YEAR", false);
ssaList.markAllFieldsForRetrieval("STOCK", true);
System.out.println("***** Created SSAList object");
// obtain a Path containing the segments that match the SSAList criteria
path = ssaList.getPathForRetrieveReplace();
// issue a DL/I GU call to retrieve the segments on the Path
if (pcb.getUnique(path, ssaList, true)){
System.out.println("DEALER NUMBER:      "+ path.getString("DEALER", "DLRNO"));
System.out.println("DEALER NAME:        "+ path.getString("DEALER", "DLRNAME"));
System.out.println("CITY:               "+ path.getString("DEALER", "CITY"));
System.out.println("MAKE, MODEL & YEAR:  "+ path.getString("MODEL", "MODKEY"));
System.out.println("MSRP:               "+ path.getString("MODEL", "MSRP"));
System.out.println("VIN NUMBER:         "+ path.getString("STOCK", "STKVIN"));
System.out.println("COLOR:              "+ path.getString("STOCK", "COLOR"));
}
// rollback the updates to MODEL & STOCK
psb.rollback();
// close the database connection
psb.close();
System.out.println("***** Disconnected from IMS database");
} catch (DLIException e){
    System.out.println(e);
    System.exit(0);
}

```

```

    }
  }
}

```

---

Example 8-12 shows the output of the application in Example 8-11.

*Example 8-12 Output from the application*

---

```

**** Created a connection to the IMS database
**** Created PCB object
**** Created SSAList object
**** Created Path object
Dealer Number: 1234
Dealer Name:   SAN JOSE FORD
City:         SAN JOSE
ZIP:          95777-3333
Type of Model: S
Manufacturer: FORD
Model:        FOCUS
Year:         2002
MSRP:         17995
# in Stock:   03
VIN Number:   V234567890123456789V
Colour:       LIGHT BLUE
Price:        16000
Warrenty:     Y
  Insert of STOCK Segment 1 Successful
  Insert of STOCK Segment 2 Successful
  Insert of STOCK Segment 3 Successful
**** Created SSAList object
**** Created Path object
DEALER NUMBER: 7575
DEALER NAME:   IBM Super Systems
CITY:          Portsmouth
ZIP:           P06 3AU
PHONE NUMBER:  0239256
TYPE of MODEL: MF
MAKE MODEL YEAR: IBM      Z10 GT    2010
MSRP:          12500
Number in Stock: 3
VIN NUMBER:    VT11234677098557729C
COLOR:         Blue
PRICE:         99999
LOT:          BS12345678
WARRENTY?:     Y
VIN NUMBER:    VT11234677098557730C
COLOR:         Deep Blue
PRICE:         99999
LOT:          BS12345679
WARRENTY?:     Y
VIN NUMBER:    VT11234677098557731C
COLOR:         Light Blue
PRICE:         99999
LOT:          BS12345680
WARRENTY?:     Y
**** Get DLRNO 7575 ready for DELETE ****
Color is : 'Blue'
STKVIN = VT11234677098557729C
This one is Blue do not delete
Color is : 'Deep Blue'

```



```

DELETE of STOCK Segment VT11234677098557730C Successful
Color is : 'Light Blue'
DELETE of STOCK Segment VT11234677098557731C Successful
All STOCK segments not COLOR = BLUE deleted
DEALER DLRNO = '7575' and dependents deleted
**** Created SSAList object
**** Created Path object
DEALER NUMBER:      1234
DEALER NAME:        SAN JOSE FORD
CITY:                SAN JOSE
MAKE, MODEL & YEAR:  FORD      FOCUS      2002
MSRP:                17995
VIN NUMBER:          V234567890123456789V
COLOR:               LIGHT BLUE
DEALER NUMBER:      1234
DEALER NAME:        SAN JOSE FORD
CITY:                SAN JOSE
MAKE, MODEL & YEAR:  FORD      FOCUS      2002
MSRP:                45999
VIN NUMBER:          V234567890123456789V
COLOR:               Green
**** Created SSAList object
DEALER NUMBER:      1234
DEALER NAME:        SAN JOSE FORD
CITY:                SAN JOSE
MAKE, MODEL & YEAR:  FORD      FOCUS      2002
MSRP:                17995
VIN NUMBER:          V234567890123456789V
COLOR:               LIGHT BLUE
**** Disconnected from IMS database

```

---

## 8.1.6 Using the Batch Methods with the IMS Universal DL/1 driver

When you hear that there is a “batch method” of using the DL/1 driver, you might immediately think it would be something you ran in batch using System z Job Control Language (JCL). You will soon found out that it is wrong!

It is a way of accessing/updating multiple segments with a single call. Instead of the application having to make multiple getUnique and getNext calls, IMS performs all the calls and returns the results back to the client in a single batch operation. The number of rows to be returned for each batch network operation is determined by the fetch size property which is set for you internally but can be overridden by the pcb.getFetchSize(n); statement where n is a number you deem acceptable. This is especially relevant for a distributed client in order to maximise network efficiency. The driver will build a request for the number of rows asked for to be returned and send it to ODBM (via IMS Connect) who will interact with IMS to retrieve this number of rows (if available), one network interaction will retrieve multiple rows (or segments). If the remote client application continues to ask for more rows, the driver will submit a request for another set of rows to be returned. This facility is available in all the drivers:-

- ▶ Universal DB Resource Adapter – for JDBC, and for CCI SQL or DL/1 access
- ▶ Universal JDBC Driver
- ▶ Universal DL/1 Driver

Update and delete operations can also make use of the batch method where you have multiple In a batch update or delete operation, the IMS host will do a GU/GN loop and inside the loop update or delete each record until there are no more segments matching the SSAList

and return the number of records deleted.segments that match the selection criteria in the SSAList that are to be updated or deleted.

Example 8-13 shows an application using the batch methods for retrieval, updating and deleting data.This is downloadable from the file IMS Universal DLI Driver sample.zip as described in Appendix D, “Additional material” on page 249.

*Example 8-13 dlitest2 - Batch methods example*

---

```
package dlitest2;
import com.ibm.ims.dli.*;
public class DLiprogram2 {
    public static void main(String[] args) {
        PSB psb = null;
        PCB pcb = null;
        SSAList ssaList = null;
        Path path = null;
        PathSet pathSet = null;
        int cnt = 0;
        try {
            // establish a database connection
            IMSConnectionSpec connSpec
            = IMSConnectionSpecFactory.createIMSConnectionSpec();
            connSpec.setDatastoreName("IMSZ");
            connSpec.setDatastoreServer("wtsc63.itso.ibm.com");
            connSpec.setPortNumber(5555);
            connSpec.setMetadataURL(
            "class://samples.ims.openDb.AUTPSB11DatabaseView");
            connSpec.setUser("IMS2R");
            connSpec.setPassword("t0byjugs");
            connSpec.setDriverType(IMSConnectionSpec.DRIVER_TYPE_4);
            psb = PSBFactory.createPSB(connSpec);
            System.out.println(
                "***** Created a connection to the IMS database");
            pcb = psb.getPCB("AUTOLPCB");
            System.out.println("***** Created PCB object");
            // add the initial qualification
            // specify the segment search arguments
            ssaList = pcb.getSSAList("DEALER","MODEL");
            ssaList.addInitialQualification("DEALER","DLRNAME",
                SSAList.NOT_EQUAL,"IBM Bristol Cars");
            ssaList.addInitialQualification("MODEL","MODEL",
                SSAList.EQUALS,"FOCUS");
            ssaList.appendQualification("MODEL",SSAList.AND,
                "COUNT1",SSAList.GREATER_THAN, "02");
            // add the initial qualification
            // specify the fields to retrieve
            ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
            ssaList.markFieldForRetrieval("DEALER", "DLRNAME", true);
            ssaList.markFieldForRetrieval("DEALER", "CITY", true);
            ssaList.markFieldForRetrieval("DEALER", "ZIP", true);
            ssaList.markAllFieldsForRetrieval("MODEL", true);
            // ssaList.markAllFieldsForRetrieval(3, true);
            System.out.println("***** Created SSAList object");
            // obtain a Path containing the segments
            // that match the SSAList criteria
```

```

System.out.println("**** Created Path object");
// issue a DL/I GU call to retrieve the first segment on the Path
pathSet = pcb.batchRetrieve(ssaList);
while(pathSet.hasNext()){
    path = pathSet.next();
    cnt = cnt + 1;
    System.out.println("Dealer Number: "
        + path.getString("DEALER", "DLRNO")
        + " Dealer Name: "
        + path.getString("DEALER", "DLRNAME")
        + " City: "
        + path.getString("DEALER", "CITY")
        + " ZIP: "
        + path.getString("DEALER", "ZIP"));
    System.out.println("Model Type: "
        + path.getString("MODEL", "MODTYPE")
        + " Manufacturer: "
        + path.getString("MODEL", "MAKE")
        + " Model: "
        + path.getString("MODEL", "MODEL")
        + " Year: "
        + path.getString("MODEL", "YEAR")
        + " MSRP: "
        + path.getString("MODEL", "MSRP")
        + " # in Stock: "
        + path.getString("MODEL", "COUNT1"));
}
System.out.println("Number of loops = "+ cnt);
cnt = cnt - cnt;
ssaList = pcb.getSSAList("DEALER","MODEL");
ssaList.addInitialQualification("DEALER","DLRNAME",
    SSAList.NOT_EQUAL,"IBM Bristol Cars");
ssaList.addInitialQualification("MODEL","MODEL",
    SSAList.EQUALS,"FOCUS");
ssaList.appendQualification("MODEL",SSAList.AND,
    "COUNT1",SSAList.GREATER_THAN,"02");
path = ssaList.getPathForBatchUpdate("MODEL");
path.setString("MSRP", "15500");
int i = pcb.batchUpdate(path, ssaList);
System.out.println("Number of Segments Updated = "+ i);
System.out.println("Updates Done");
// pathSet = pcb.batchRetrieve(ssaList);
// issue multiple DL/I GN calls until
// there are no more segments to retrieve
ssaList = pcb.getSSAList("DEALER","MODEL");
ssaList.addInitialQualification("DEALER","DLRNAME",
    SSAList.NOT_EQUAL,"IBM Bristol Cars");
ssaList.addInitialQualification("MODEL","MODEL",
    SSAList.EQUALS,"FOCUS");
ssaList.appendQualification("MODEL",SSAList.AND,
    "COUNT1",SSAList.GREATER_THAN,"02");
ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
ssaList.markFieldForRetrieval("DEALER", "DLRNAME", true);
ssaList.markFieldForRetrieval("DEALER", "CITY", true);
ssaList.markFieldForRetrieval("DEALER", "ZIP", true);

```

```

ssaList.markAllFieldsForRetrieval("MODEL", true);
pathSet = pcb.batchRetrieve(ssaList);
while(pathSet.hasNext()){
    path = pathSet.next();
    cnt = cnt + 1;
    System.out.println("Dealer Number: "
        + path.getString("DEALER", "DLRNO")
        + " Dealer Name: "
        + path.getString("DEALER", "DLRNAME")
        + " City: "
        + path.getString("DEALER", "CITY")
        + " ZIP: "
        + path.getString("DEALER", "ZIP"));
    System.out.println("Model Type: "
        + path.getString("MODEL", "MODTYPE")
        + " Manufacturer: "
        + path.getString("MODEL", "MAKE")
        + " Model: "
        + path.getString("MODEL", "MODEL")
        + " Year: "
        + path.getString("MODEL", "YEAR")
        + " MSRP: "
        + path.getString("MODEL", "MSRP")
        + " # in Stock: "
        + path.getString("MODEL", "COUNT1"));
}
System.out.println("Numer of loops = "+ cnt);
ssaList = pcb.getSSAList("DEALER","MODEL");
ssaList.addInitialQualification("DEALER","DLRNAME",
    SSAList.NOT_EQUAL,"IBM Bristol Cars");
ssaList.addInitialQualification("MODEL","MODEL",
    SSAList.EQUALS,"FOCUS");
ssaList.appendQualification("MODEL",SSAList.AND,
    "COUNT1",SSAList.GREATER_THAN,"02");
i = pcb.batchDelete(ssaList);
System.out.println("Number of Segments Deleted = "+ i);
System.out.println("Deletes Done");
cnt = cnt - cnt;
ssaList = pcb.getSSAList("DEALER","MODEL");
ssaList.markFieldForRetrieval("DEALER", "DLRNO", true);
ssaList.markFieldForRetrieval("DEALER", "DLRNAME", true);
ssaList.markFieldForRetrieval("DEALER", "CITY", true);
ssaList.markFieldForRetrieval("DEALER", "ZIP", true);
ssaList.markAllFieldsForRetrieval("MODEL", true);
pathSet = pcb.batchRetrieve(ssaList);
while(pathSet.hasNext()){
    path = pathSet.next();
    cnt = cnt + 1;
    System.out.println("Dealer Number: "
        + path.getString("DEALER", "DLRNO")
        + " Dealer Name: "
        + path.getString("DEALER", "DLRNAME")
        + " City: "
        + path.getString("DEALER", "CITY")
        + " ZIP: "

```

```

        + path.getString("DEALER", "ZIP"));
    System.out.println("Model Type: "
        + path.getString("MODEL", "MODTYPE")
        + " Manufacturer: "
        + path.getString("MODEL", "MAKE")
        + " Model: "
        + path.getString("MODEL", "MODEL")
        + " Year: "
        + path.getString("MODEL", "YEAR")
        + " MSRP: "
        + path.getString("MODEL", "MSRP")
        + " # in Stock: "
        + path.getString("MODEL", "COUNT1"));
    }
    System.out.println("Numer of loops = "+ cnt);
    // rollback the updates to MODEL
    psb.rollback();
    // close the database connection
    psb.close();
    System.out.println("***** Disconnected from IMS database");
} catch (DLIException e){
    AIB aib = e.getAib();
    if (aib != null) {
        String sc = aib.getDBPCB().getStatusCodeChars();
        String retcode = aib.getReturnCodeHex();
        String reascode = aib.getReasonCodeHex();
        System.out.println("Status code: " + sc + " Return Code: "
            + retcode + " Reason Code: " + reascode);
    }
    System.out.println(e);
    System.exit(0);
}
}
}
}

```

---

## 8.2 Writing application with the IMS Universal DB Resource Adapter and the CCI programming approach

The IMS Universal DB Resource Adapter offers the capability of writing applications using the CCI programming model. As JDBC is our recommended approach for accessing IMS data with the IMS Universal Drivers, it can have advantages to use the CCI programming model. JDBC doesn't allow you the granularity of accessing hierarchical data then DL/I does. The CCI programming model allows you to use SQL and DL/I syntax within one Connection and application. You could also use the standalone JDBC and standalone DL/I drivers within one application but you would need two separate connections and you would have to coordinate the commit or rollbacks by yourself (or with global transaction support).

**Note:** The IMS Universal DB Resource Adapters are intended to be used in managed JEE Application Servers, but they also give you the possibility of using them in standalone non-managed applications.

The IMS Universal DB Resource Adapters are RAR files and contain jar files. In order to use the IMS Universal DB Resource Adapters in standalone applications, it is useful to extract the jar files by using a extract utility, that is capable of extracting RAR archives. The extracted Jar files must be (like with all other drivers) be in the Class path of the application.

The example in this paragraphs explains the code of an application using the IMS Universal DB Resource Adapter and the CCI programming approach in a standalone environment.

For more information about the CCI API see the CCI API in the Information Center under

[http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.ims11.doc.apr/ims\\_odbjcasupportforcci.htm](http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.ims11.doc.apr/ims_odbjcasupportforcci.htm)

## 8.2.1 Writing the application step by step

The following steps explain step by step the important steps of writing the application. For a full reference of the program see Appendix D, "Additional material" on page 249.

1. You will need the import statements listed in Example 8-14 in you application.

### *Example 8-14 Import Statements of CCI Application*

---

```
import javax.naming.InitialContext; //only in managed environemnts
import javax.transaction.UserTransaction; //only in manged environments
import javax.resource.cci.*;
import com.ibm.ims.db.cci.*;
```

---

Therefore you must have the WebSphere Application Library in you Class path as it contains the j2ee.jar which contains the used javax.\* classes.

2. In a non managed environment you have to create a new IMSManagedConnectionFactory object and specify the needed values for the connection. From the ManagedConnectionFactory you can create a ConnectionFactory which creates the Connection itself. See Example 8-15.

### *Example 8-15 Create MCF*

---

```
IMSManagedConnectionFactory mcf = new IMSManagedConnectionFactory();
mcf.setUser("user");
mcf.setPassword("password");
mcf.setDatastoreName("IMS2");
mcf.setDatastoreServer("myhost.itso.ibm.com");
mcf.setPortNumber(5555);
mcf.setMetadataURL("class://samples.ims.openDb.AUTPSB11DatabaseView");
mcf.setSSLConnection(false);
mcf.setDriverType(IMSManagedConnectionFactory.DRIVER_TYPE_4);
mcf.setLoginTimeout(10);
ConnectionFactory cf = (ConnectionFactory) mcf.createConnectionFactory();
Connection conn = cf.getConnection();
```

---

In a managed environment you just have to lookup the JNDI name of the ConnectionFactory with the help of the InitialContext which is passed to the application from the container. See Example 8-16.

### *Example 8-16 Lookup MCF*

---

```
InitialContext ic = new InitialContext();
ConnectionFactory cf = (ConnectionFactory) ic.lookup("java:comp/env/MyMCF");
Connection conn = cf.getConnection();
```

---

3. For transaction handling you can use the code in Example 8-17 in your unmanaged application.

*Example 8-17 Transaction calls*

---

```
LocalTransaction trans =conn.getLocalTransaction();
trans.begin();
...
trans.commit(); // or trans.rollback();
```

---

In a managed environment you can get the UserTransaction object from the SessionContext as shown in 5.3.1, “JCA/Common Client Interface approach” on page 105.

4. Create an Interaction object and a SQLInteractionSpec and a DLIInteractionSpec object which contains informations about the Interaction itself. See Example 8-18.

*Example 8-18 Create Interaction and InteractionSpecs*

---

```
Interaction ix = conn.createInteraction();
SQLInteractionSpec sqlSpec = new SQLInteractionSpec();
DLIInteractionSpec dliSpec = new DLIInteractionSpec();
```

---

5. Insert 1 DEALER root segment and 3 dependent MODEL segments with the SQL syntax shown in Example 8-19.

*Example 8-19 Insert segments with SQL*

---

```
sqlSpec.setSQL("INSERT INTO AUTOLPCB.DEALER (DLRNO, ZIP, DLRNAME, CITY, PHONE) " +
    "VALUES('7777', '70565','Thilos A,B and X Model Cars','Stuttgart','555-888')");
ix.execute(sqlSpec, null);
sqlSpec.setSQL("INSERT INTO AUTOLPCB.MODEL " +
    "(DEALER_DLRNO,MODTYPE,MAKE,MODEL,YEAR,MSRP,COUNT1)" +
    " VALUES ('7777','S','LIDLA','A Normal','2010','20000','05')");
ix.execute(sqlSpec, null);
sqlSpec.setSQL("INSERT INTO AUTOLPCB.MODEL " +
    "(DEALER_DLRNO,MODTYPE,MAKE,MODEL,YEAR,MSRP,COUNT1) " +
    "VALUES ('7777','M','LIDLA','B Plus','2010','40000','03')");
ix.execute(sqlSpec, null);
sqlSpec.setSQL("INSERT INTO AUTOLPCB.MODEL " +
    "(DEALER_DLRNO,MODTYPE,MAKE,MODEL,YEAR,MSRP,COUNT1) " +
    "VALUES ('7777','L','LIDLA','X Large','2010','60000','01')");
ix.execute(sqlSpec, null);
```

---

6. Next we retrieve the information from the database. With the DL/I syntax the code would look like Example 8-20.

*Example 8-20 Retrieve information with DL/I*

---

```
dliSpec.setFunctionName("RETRIEVE");
dliSpec.setPCBName("AUTOLPCB");
dliSpec.setSSAList("DEALER *D (DLRNO = '7777') MODEL ");
ResultSet dlirs = (ResultSet) ix.execute(dliSpec, null);
while (dlirs.next()) {
    System.out.print(dlirs.getString("DLRNO")+" ; ");
    System.out.print(dlirs.getString("DLRNAME")+" ; ");
    System.out.print(dlirs.getString("MODTYPE")+" ; ");
    System.out.print(dlirs.getString("MAKE")+" ; ");
    System.out.print(dlirs.getString("MODEL")+" ; ");
    System.out.print(dlirs.getString("MSRP")+" ; ");
    System.out.print(dlirs.getString("COUNT1")+" \n");
}
```

---

---

}

Here the RETRIEVE function is used by specifying a IMS Path Call in the SSA statement. The execute function does not need an input object. As output object you will get back a ResultSet object which can be used as usually.

The same function with SQL syntax would look like Example 8-21.

---

*Example 8-21 Retrieve information with SQL*

---

```
sqlSpec.setSQL("SELECT * FROM AUTOLPCB.DEALER,AUTOLPCB.MODEL " +
    "WHERE DEALER.DLRNO='7777' AND MODEL.DEALER_DLRNO='7777'");
ResultSet sqlrs = (ResultSet) ix.execute(sqlSpec, null);
while (sqlrs.next()) {
    System.out.print(sqlrs.getString("DLRNO")+" ; ");
    System.out.print(sqlrs.getString("DLRNAME")+" ; ");
    System.out.print(sqlrs.getString("MODTYPE")+" ; ");
    System.out.print(sqlrs.getString("MAKE")+" ; ");
    System.out.print(sqlrs.getString("MODEL")+" ; ");
    System.out.print(sqlrs.getString("MSRP")+" ; ");
    System.out.print(sqlrs.getString("COUNT1")+" \n");
}
```

---

It uses the WHERE clause to specify which segments should be received and gets also a ResultSet which can be used in the usual way.

- One thing that cannot be done with SQL syntax is for example to update the last dependent segment in the database. For this call we use the DL/I syntax in Example 8-22.

---

*Example 8-22 Update information with DL/I*

---

```
dliSpec.setFunctionName("UPDATE");
dliSpec.setSSAList("DEALER *D (DLRNO = '7777') MODEL *L");
RecordFactory rf = cf.getRecordFactory();
MappedRecord input = rf.createMappedRecord("DEALER");
input.put("DEALER.DLRNAME", "Thilos A and B Model Cars");
input.put("MODEL.MSRP", "00000");
input.put("MODEL.COUNT1", "00");
ix.execute(dliSpec, input);
```

---

The SSA contains the \*L command and uses a path call from DEALER and MODEL by specifying \*D command. In this case you have to create a MappedRecord with DEALER as parameter, but as it is a path call you can modify any segment in the path that is specified. This example sets the last occurrence of MODEL in this path to MSRP=00000 and COUNT1=00 to indicate that this model is no longer available. Therefore we change also the DLRNAME in the parent DEALER segment. If you just want to update the MODEL segment you do not have to specify a \*D path call command and can map the record to the MODEL segment.

**Note:** The last occurrence is not necessarily the last inserted. It depends on the key field of the Segment.

The key field of the Model segment is build from the MAKE,MODEL and YEAR. So the MODKEY fields would look like Example 8-23 for our three inserts.

---

*Example 8-23 Output of MODKEY retrieve*

---

```
MODKEY No 1 is "LIDLA      A Normal  2010"
MODKEY No 2 is "LIDLA      B Plus    2010"
```



---

MODKEY No 3 is "LIDLA      X Large      2010"

---

The last occurrence would be the last one as the key differentiates the position on A,B or X.

8. In the end we do a delete of the inserted DEALER segment with the code in Example 8-24.

---

*Example 8-24 Delete data with SQL*

---

```
sqlSpec.setSQL("DELETE FROM AUTOLPCB.DEALER WHERE DLRNO='7777'");
sqlrs = (ResultSet) ix.execute(sqlSpec, null);
if(sqlrs.next()){
    System.out.println("DEALER Rows deleted :"+sqlrs.getInt("UPDATE_COUNT"));
}
```

---

With the deletion of the DEALER segment, also all referenced (dependent) segments get deleted. To get the information if the delete was successful, you can use the ResultSet and get the generated UPDATE\_COUNT field.

## 8.2.2 Complete Code Example of CCI mixed application

The code in Example 8-25 shows the complete application which is described in 8.2.1, "Writing the application step by step" on page 200.

---

*Example 8-25 CCISTandaloneDLIandSQL*

---

```
package samples.ims.applications;
import javax.resource.cci.*;
import com.ibm.ims.db.cci.*;

public class CCISTandaloneDLIandSQL {
    public static void main(String[] args) {
        System.out.println("Creating Connection Factory and specifying values");
        Connection conn = null;
        IMSManagedConnectionFactory mcf = new IMSManagedConnectionFactory();
        mcf.setUser("user");
        mcf.setPassword("password");
        mcf.setDatastoreName("IMS2");
        mcf.setDatastoreServer("myhost.itso.ibm.com");
        mcf.setPortNumber(5555);
        mcf.setMetadataURL("class://samples.ims.openDb.AUTPSB11DatabaseView");
        mcf.setSSLConnection(false);
        mcf.setDriverType(IMSMangedConnectionFactory.DRIVER_TYPE_4);
        mcf.setLoginTimeout(10);
        try{
            ConnectionFactory cf = (ConnectionFactory) mcf.createConnectionFactory();
            conn = cf.getConnection();
            LocalTransaction trans =conn.getLocalTransaction();
            trans.begin();
            Interaction ix = conn.createInteraction();
            System.out.println("Started Transaction");
            SQLInteractionSpec sqlSpec = new SQLInteractionSpec();
            System.out.println("Insert DEALER segment and 3 MODELS");
            sqlSpec.setSQL("INSERT INTO AUTOLPCB.DEALER " +
                "(DLRNO, ZIP, DLRNAME, CITY, PHONE) " +
                "VALUES('7777', '70565','Thilos A,B and X Model Cars','Stuttgart','555-888')");
            ix.execute(sqlSpec, null);
            sqlSpec.setSQL("INSERT INTO AUTOLPCB.MODEL " +
```

```

        "(DEALER_DLRNO,MODTYPE,MAKE,MODEL,YEAR,MSRP,COUNT1)" +
        " VALUES ('7777','S','LIDLA','A Normal','2010','20000','05')");
ix.execute(sqlSpec, null);
sqlSpec.setSQL("INSERT INTO AUTOLPCB.MODEL " +
        "(DEALER_DLRNO,MODTYPE,MAKE,MODEL,YEAR,MSRP,COUNT1)" +
        "VALUES ('7777','M','LIDLA','B Plus','2010','40000','03')");
ix.execute(sqlSpec, null);
sqlSpec.setSQL("INSERT INTO AUTOLPCB.MODEL " +
        "(DEALER_DLRNO,MODTYPE,MAKE,MODEL,YEAR,MSRP,COUNT1)" +
        "VALUES ('7777','L','LIDLA','X Large','2010','60000','01')");
ix.execute(sqlSpec, null);
System.out.println("Display DEALER and MODEL with DL/I");
DLIInteractionSpec dliSpec = new DLIInteractionSpec();
dliSpec.setFunctionName("RETRIEVE");
dliSpec.setPCBName("AUTOLPCB");
dliSpec.setSSAList("DEALER *D (DLRNO = '7777') MODEL ");
ResultSet dlirs = (ResultSet) ix.execute(dliSpec, null);
while (dlirs.next()) {
    System.out.print(dlirs.getString("DLRNO")+" ; ");
    System.out.print(dlirs.getString("DLRNAME")+" ; ");
    System.out.print(dlirs.getString("MODTYPE")+" ; ");
    System.out.print(dlirs.getString("MAKE")+" ; ");
    System.out.print(dlirs.getString("MODEL")+" ; ");
    System.out.print(dlirs.getString("MSRP")+" ; ");
    System.out.print(dlirs.getString("COUNT1")+" \n");
}
System.out.println("Modify Last Occurrence of MODEL of the inserted DEALER");
dliSpec.setFunctionName("UPDATE");
dliSpec.setSSAList("DEALER *D (DLRNO = '7777') MODEL *L");
RecordFactory rf = cf.getRecordFactory();
MappedRecord input = rf.createMappedRecord("DEALER");
input.put("DEALER.DLRNAME", "Thilos A and B Model Cars");
input.put("MODEL.MSRP", "00000");
input.put("MODEL.COUNT1", "00");
ix.execute(dliSpec, input);
sqlSpec.setSQL("SELECT * FROM AUTOLPCB.DEALER,AUTOLPCB.MODEL " +
        "WHERE DEALER.DLRNO='7777' AND MODEL.DEALER_DLRNO='7777'");
ResultSet sqlrs = (ResultSet) ix.execute(sqlSpec, null);
while (sqlrs.next()) {
    System.out.print(sqlrs.getString("DLRNO")+" ; ");
    System.out.print(sqlrs.getString("DLRNAME")+" ; ");
    System.out.print(sqlrs.getString("MODTYPE")+" ; ");
    System.out.print(sqlrs.getString("MAKE")+" ; ");
    System.out.print(sqlrs.getString("MODEL")+" ; ");
    System.out.print(sqlrs.getString("MSRP")+" ; ");
    System.out.print(sqlrs.getString("COUNT1")+" \n");
}
sqlSpec.setSQL("DELETE FROM AUTOLPCB.DEALER WHERE DLRNO='7777'");
sqlrs = (ResultSet) ix.execute(sqlSpec, null);
if(sqlrs.next()){
    System.out.println("DEALER Rows deleted : "+sqlrs.getInt("UPDATE_COUNT"));
}
}
trans.rollback();
dlirs.close();
sqlrs.close();
ix.close();
conn.close();
System.out.println("Transaction rolled back and Connection closed");
} catch (Exception e) {
    e.printStackTrace();
}

```

```

        try {
            conn.close();
        } catch (Exception e1) {
            e1.printStackTrace();
        }
    }
}

```

---

The output of the application should look like the lines in Example 8-26.

---

*Example 8-26 Output of CCIStandaloneSQLandDLI Java application*

---

Creating Connection Factory and specifying values

Started Transaction

Insert DEALER segment and 3 MODELS

Display DEALER and MODEL with DL/I

7777 ; Thilos A,B and X Model Cars ; S ; LIDLA ; A Normal ; 20000 ; 05

7777 ; Thilos A,B and X Model Cars ; M ; LIDLA ; B Plus ; 40000 ; 03

7777 ; Thilos A,B and X Model Cars ; L ; LIDLA ; X Large ; 60000 ; 01

Modify Last Occurrence of MODEL of the inserted DEALER

7777 ; Thilos A and B Model Cars ; S ; LIDLA ; A Normal ; 20000 ; 05

7777 ; Thilos A and B Model Cars ; M ; LIDLA ; B Plus ; 40000 ; 03

7777 ; Thilos A and B Model Cars ; L ; LIDLA ; X Large ; 00000 ; 00

DEALER Rows deleted :1

Transaction rolled back and Connection closed

---



# 9

## Operational considerations

This chapter is about general recommendations and best practices. It contains information on the following topics:

- ▶ Architectural suggestions
- ▶ Enhancing existing applications
- ▶ Tracing in problem cases
- ▶ Using Tools with IMS Open database
- ▶ Additional sample programs

## 9.1 Architectural suggestions

We have put together some information which could help you to find the right solution for your requirement.

### 9.1.1 Application middle layer

Sometimes it can make sense to develop a middle layer to prevent accessing IMS databases directly from application developers. Such cases would be:

- ▶ Usually the application developers for distributed environments do not have a valid userid and password on z/OS side to access IMS resources directly. To handle the security for all developers involved in a large application development project can cause much effort.
- ▶ A technical user id could help in the above case, but in pre-production environments some databases or parts of it should only be accessible to a certain group. Sometimes this security is currently implemented by the IMS transaction, but with this solution you would bypass the transaction. A middle layer which implements this security again could help in this case. The alternative would be to specify different PSBs for the database with different access levels.
- ▶ An application middle layer could also help to prevent heavy database workloads in IMS. Sometimes unspecific database calls can cause a database scan and causes much I/O work on the database which could affect performance for other transactions and users. A middle layer can help to allow only specific database calls to IMS.

Such a middle layer could be implemented in various ways:

- ▶ One option would be to create Web Services for application developers to do specific calls against an IMS database. This approach makes it possible to switch easily to different databases and allows a loosen coupling for a service oriented approach.
- ▶ Another option could be to implement a persistence layer like Hibernate™ or JPA for your application. The application developer uses in this case only Java objects in his application and the database calls are done by the persistence layer under the cover. Since the IMS Open Database feature uses industry standards, this approach should be easily implementable with small changes to the current restriction or supported SQL syntax of the IMS Universal Drivers.
- ▶ There exist several more options like proxying the request through an EJB bean or a self written solution.

**Note:** An alternative to implementing the middle layer would be using the CSL ODBM user exits to allow/disallow certain incoming statements or manipulating the outcome of the statements. See “ODBM user exits” on page 53 for more information.

A general recommendation cannot be given as the approach refers to the specific requirements of the application and company. IMS is opened up with the Open Database feature to the complete distributed world with all chances and threats which must be taken in consideration before you use the IMS Open Database feature in production environments.

### 9.1.2 Sysplex considerations

We did not focus on a sysplex environment in this book to keep the scenarios and the explanation simple. There are several things you must keep in mind for the IMSplex itself. If you have Data Sharing in the IMSplex in place, it does not matter to which IMS the Open DB

request is routed. You can have each component more often for backup or workload balancing reasons.

Figure 9-1 shows an example system environment with two IMS systems in a IMSplex with enabled Data Sharing. Each LPAR has its own IMS Connect Address space (IMSHWSA and IMSHWSB) and ODBM dress space (IMSODA and IMSODB). IMS Connect communicates with ODBM within the LPAR via a local PC call, but IMS Connect is also capable of communicating across LPAR boundaries via the Structured Call Interface (SCI) which supports XCF communication. So you would just need one IMS Connect address space (IMSHWSB) which routes the request depending on the ODBM configuration to IMSODB or IMSODA. To avoid a single point of failure in the IMS Connect address space is possible to use the z/OS Sysplex Distributor to route the requests to a different IMS Connect address space (here IMSHWSA) in case of a failure.

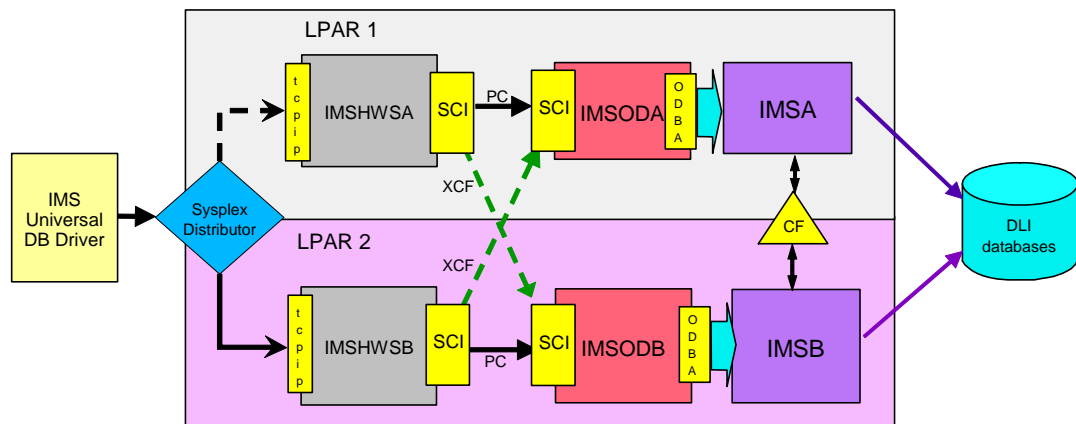


Figure 9-1 A two member IMSplex sample environment

You should have one ODBM task per LPAR, which has a control region. One ODBM can connect to more than one control region of the same IMSplex on the same LPAR.

The IMS PROCLIB member CSLDCxxx can be shared between the ODBMs in a Sysplex. You can also use for each ODBM its own configuration member. The configuration member has two parts:

- ▶ A global section, which sets values for the IMSplex as a whole
- ▶ A local section, which defines the various ODBM address spaces and control regions. This overrides the global section.

The two IMS Connect and each IMS must belong to the same IMSPLEX (CSL group). When IMS Connect starts up, connection is made to all members of the CSL group, and information is exchanged, so that IMS Connect acquires knowledge of the real ODBM address spaces and the database component of the DBCTL and DB/DC systems they are in connection with, so that they can be addressed for IMS Universal Driver queries.

For a detailed example with configuration members for a IMS Open Database Sysplex environment see Appendix "B.1 IMS configuration" of *IMS Version 11 Technical Overview*, SG24-7807.

### 9.1.3 Performance considerations

Performance is a tough topic to deal with. The focus in this book wasn't on performance aspects. The performance is strong influenced by the design of your queries in combination

with the database layout. But also configuration parameters and application parameters are important under performance considerations. We want to highlight the parameters that can influence performance:

- ▶ IMS Universal Drivers related
  - When possible use type-2 connectivity instead of type-4, as you can save the way across TCP/IP and IMS Connect. When you use WebSphere Application Server for z/OS on the same LPAR as IMS, you can also use the RRSLocalOption of the IMS Database Resource Adapter.
  - Use Local transaction support instead of using XA transaction support where possible, because you will save the expensive two-phase commit processing.
  - Return only needed columns in SELECT statements by specifying them instead of using the star (\*) value.
  - Compare columns to specific values instead to other columns.
  - Use PreparedStatements if you are doing several queries or updates, as the IMS Universal Drivers doesn't have to build every time the SSA list for the DL/I call and just have to put the values in
  - Increase the fetchSize() parameter to get more results back with each network call. This can increase performance for large queries, as you save several network calls.
  - Use secondary indexes and maybe reorganize your database where possible to fit the requirements of the application, especially for batch using applications.
  - Use DL/I for specific requests what JDBC and SQL cannot do, like a get next parent call.
- ▶ IMS Connect and ODBM related performance
  - Increase the MAXTHREAD parameter in the CSLDCxxx member to allow more parallel access to your database
  - Distribute your requests to several IMSplex members where possible
  - Use PC calls instead of XCF calls, this means when possible have IMS Connect or the type-2 application and ODBM with IMS on the same LPAR.
- ▶ RRS logging performance
  - Because of the use of RRS, ODBA performance is related to the RRS logging performance. RRS uses z/OS logger and five log streams that can be shared by multiple systems in a sysplex. You have several choices of the z/OS logger implementation; you must consider that your choice should meet your performance and recovery requirements. For a description of configuring and defining RRS logging requirements, see z/OS V1R6.0 MVS Programming: Resource Recovery, SA22-7616.

For database performance use the usual procedures you have for your daily IMS database maintenance to keep your IMS database healthy like reorganizations or rebuild pointers etc. There are no special requirements necessary for the IMS Open Database feature.

## 9.2 Enhancing existing applications

The IMS Open Database feature and the IMS Universal Database Drivers allow you to enhance your existing environments.



## 9.2.1 ODBA access through ODBM

The former architecture of the IMS DB Resource Adapter needed WebSphere Application Server z/OS to be on the same LPAR as IMS. This solution uses ODBA to get to the DL/I data. You do not require a DRA module to access IMS Open Database through IMS Connect. But you do not need to redevelop your application as the IMS Universal DB Resource Adapter is backwards compatible. The ODBA interface from previous versions of IMS can coexist with IMS 11 without modification, but it is recommended to use the new features to avoid some issues.

ODBA can make cross address space calls (PC calls) but only within the same logical partition. The ODBA modules are loaded into the address space of the application, which is running under its container's TCB, which poses a potential problem because if WebSphere Application Server (with ODBA within it) is terminated while an application's DLI call is in process, then the IMS control region receives an abend U113 when it detects that the ODBA TCB is no longer around.

You should add the IMSPLEX= parameter to your existing DRAs and reassemble them. This information is what provides u113 isolation to ODBA applications because it causes ODBA to use ODBM instead of connecting directly to IMS. You might also want to add the ODBMNAME= parameter if you want to ensure connection to a specific ODBM. Example 9-1 shows a sample DRA.

*Example 9-1 Sample DRA for ODBA access through ODBM*

---

```
//DFSIVP10 EXEC PROC=ASMDRA,MBR=DFSIMSB0
//ASM.SYSIN DD *
DFSIMSB0 CSECT
    DFSPRP DSECT=NO,
        FUNCLV=2,
        DDNAME=DFSDB2SP,
        DSNAME=IMS11B.SDFSRESL,
        DBCTLID=IMSB,
        USERID=,
        MINTHRD=1,
        MAXTHRD=1,
        TIMER=60,
        FPBUF=,
        FPBOF=,
        SOD=A,
        TIMEOUT=60,
        IDRETRY=0,
        CNBA=,
        IMSPLEX=PLEXB,
        ODBMNAME=ODOB
        ODBA FUNCTION LEVEL
        DDNAME FOR DRA RESLIB
        DSNAME FOR DRA RESLIB
        DBCTL IDENTIFIER
        USER IDENTIFIER
        MINIMUM NUMBER OF THREADS
        MAXIMUM NUMBER OF THREADS
        IDENTIFY TIMER VALUE DEFAULT
        NUMBER OF FP BUFFERS PER THREAD
        NUMBER OF FP OVERFLOW BUFFERS
        SNAP DATASET OUTPUT CLASS
        DRATERM TIMEOUT VALUE
        IDENTIFY RETRY COUNT
        TOTAL FP NBA BUFFERS FOR CCTL
        IMSPLEX NAME
        OPTIONAL ODBMNAME
END
//*
```

---

Although the ODBA program connects to IMS through ODBM, connecting to an ODBM is only possible if it is running on the same LPAR. If your programs have to access an ODBM on another LPAR, you can choose between implementing your own DRDA solution or using the ODBM API.

For more information about using the ODBA interface see section “19.1 Accessing IMS databases through the ODBA” of *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity*, SG24-6794

## 9.2.2 Enabling unsupported Java environments

The IMS Universal JDBC Drivers should normally work in most Java using applications which supports the standards like JDBC. There are certainly some restrictions as the drivers currently only support what is possible with IMS. You are currently not possible to create a database or table via a SQL statement for example.

IMS has currently no catalog where the metadata information is stored which is needed by the IMS Universal drivers. Therefore you have to use the metadata class file normally packaged in a Jar file. But some applications support JDBC, but you can select just one Jar file for the driver. To get the driver operating in such environments you have different approaches:

- ▶ You can pack your generated metadata files in the driver jar file. So you can just reference to it in the application. The disadvantage of this solution is that you have to edit each time the driver is there is a new driver version to deploy or you have changed or want to add any database.
- ▶ Another approach is to add the metadata file to the class path of the application. Therefore you have two options depending on the architecture of the application you want to enable:
  - You can specify the class path of the application to contain the Jar files of the IMS Universal Driver and metadata files. This can be done whether by specifying the system's environment variables or adding it otherwise to the class path of the application, e.g. specifying it with a start parameter if possible.
  - The alternative is to put the Jar file in the JRE/lib/ext folder which is used by the Java application, so that it will be loaded by starting the JVM. Here can also be the driver deployed, if the application doesn't give the option to specify a Jar file for the Driver. This approach is usually not a good programming model, but it can be used if no other option is available.

Of course you can also ask the application provider to add official support for the IMS Universal drivers to his product by allowing to select the classpath or Jar files of the IMS Universal JDBC driver and the metadata file.

## 9.3 Tracing in problem cases

There are several situations where you might want enable tracing to find the problem cause.

### 9.3.1 IMS Universal driver tracing

The IMS Universal Drivers have several places where you can set the tracing option depending on your runtime environment

#### Tracing in JRE applications

One option is to turn on automatic tracing in the JRE logging.properties file.

You can set the trace level for the IMS Universal drivers loggers in the logging.properties file of your Java Runtime Environment (JRE). Using this method, the application does not need to be recompiled. The file is located on the install path of your JRE, under `\jre\lib\logging.properties`. To set the trace for all IMS Universal drivers loggers, add the lines in Example 9-2 to the logging.properties file to send the trace output to a file.

*Example 9-2 logging.properties entries for tracing*


---

```
com.ibm.ims.db.opendb.level = FINEST
java.util.logging.FileHandler.level = FINEST
java.util.logging.FileHandler.pattern = c:/UniversalDriverTrace.txt
java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter
```

---

**Note:** Some JREs overwrite these settings and you will find the log file in a different location. In the case of Rational Developer for System z with enabled logging it will put the log file as .log file in the .metadata directory of your workspace directory.

**Tracing in J2EE environments**

Tracing can be turned on from your J2EE application server. In WebSphere Application Server, this is configured through the administrative console. The IMS Universal DB resource adapter must be deployed on WebSphere Application Server before tracing can be configured. To get the most detailed trace from the IMS Universal DB resource adapter, follow these steps:

- ▶ Log on to the WebSphere Application Server administration console and select **Troubleshooting->Logs and Trace**.
- ▶ Select your **application server**.
- ▶ Select **General Properties->Diagnostic Trace**.
- ▶ Select **Additional Properties->Change Log Detail Levels**.
- ▶ Switch to the **Runtime tab**.
- ▶ Make sure that the Save runtime changes to configuration as well check box is turned ON.
- ▶ Select Change Log Details Levels and choose the component `com.ibm.ims.db.opendb.*`.
- ▶ In the Message and Trace levels menu select the message level FINEST and click Apply.

To save these changes for the next time the application server is started, click the Save link at the top of the page. WebSphere Application Server does not need to be restarted.

**Tracing in applications**

You can also programmatically turn on tracing in your IMS Universal drivers application. This requires the application to be recompiled.

- ▶ Import the `java.util.logging` package in your application and create a logger by calling the `Logger.getLogger` method with the String argument `"com.ibm.ims.db.opendb"`.
- ▶ In your application, you can set the level of tracing for the logger by using the `Logger.setLevel` method.

The sample code in Example 9-3 shows how programmatic trace is enabled for any IMS Universal drivers application.

*Example 9-3 Application enabled tracing*


---

```
import java.util.logging;
...
private static final Logger imslog = Logger.getLogger("com.ibm.ims.db.opendb");
imslog.setLevel(Level.FINEST);
FileHandler fh = new FileHandler("C:/UniversalTrace.txt");
fh.setFormatter(new SimpleFormatter());
fh.setLevel(Level.FINEST);
imslog.addHandler(fh);
```

---

### 9.3.2 ODBM tracing

ODBM is the layer between IMS Connect and IMS. It is a Common Service Layer (CSL) address space and offers the usual tracing capabilities of CSL and Base Primitive Environment Tracing. This type of tracing should be used to determine if it is a component error or not.

- To start or stop a trace use the following commands:

```
UPDATE ODBM START(TRACE) DATASTORE(names)
UPDATE ODBM STOP(TRACE) DATASTORE(names)
```

For more information about CSL and BPE tracing see

[http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.ims11.doc.sag/system\\_intro/ims\\_tracingbpecomponents.htm](http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.ims11.doc.sag/system_intro/ims_tracingbpecomponents.htm)

and

[http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.ims11.doc.dgr/ims\\_csl\\_service\\_aids.htm](http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.ims11.doc.dgr/ims_csl_service_aids.htm)

### 9.3.3 IMS Tracing

If you receive an unexpected result for a query, it is a good idea to evaluate how your SQL call is described in the DL/I call format. For this purpose, you can use the tracing facility for DL/I calls with image captures by using the IMS trace command. To take a DL/I call image capture (for example, for the PSB AUTPSB11), use the following sequence of events for tracing:

1. Turn on the trace with the following command:

```
/TRACE SET ON PSB AUTPSB11 COMP
```

2. Run the Java application.

3. Turn off the trace with the following command:

```
/TRACE SET OFF PSB AUTPSB11
```

4. Switch the online log data sets (OLDSs):

```
/SWITCH OLDS
```

5. Execute the print utility DFSERA10, specifying the latest SLDS as input in SYSUT1 DD. The DFSERA10 input control statements to retrieve the information from the log look similar to the statements in Example 9-4.

*Example 9-4 DFSERA10 options*

---

```
OPTION PRINT OFFSET=5,VALUE=5F,COND=M
OPTION PRINT EXITR=DFSERA50,OFFSET=25,FLDTYP=C,      X
VALUE=AUTPSB11,FLDLLEN=7,DDNAME=OUTDDN,COND=E
```

---

## 9.4 Using Tools with IMS Open database

IBM IMS tools deliver the reliability and affordability you need to maximize the value of your IMS environment. Providing on demand access to IMS applications and data, the tools help optimize data across your enterprise. Information on IMS tools is available at:

<http://www.ibm.com/software/data/db2imstools/products/ims-tools.html>

Tools are grouped in solution packs for convenience. Of particular interest is the Performance Solution Pack, which is being extended to offer functionalities in the IMS Open Database area. The Performance Solution pack is described at:

<http://www.ibm.com/software/data/db2imstools/imstools/ims-performance-solution-pack/>

IMS Performance Solution Pack (program number: 5655-S42) provides improved productivity for problem analysts, more efficient IMS application performance, improved IMS resource utilization, and higher system availability. It includes:

- ▶ IMS Connect Extensions helps improve the availability, reliability, and performance of IMS Connect.
- ▶ IMS Performance Analyzer provides information on IMS system performance for monitoring, tuning, managing service levels, analyzing trends, and capacity planning.
- ▶ IMS Problem Investigator provides services to that can help determine the cause of problems and trace the flow of events end to end.

We describe functions for Open Database provided by IMS Connect Extensions and IMS Problem Investigator.

### 9.4.1 IMS Connect Extensions

We have seen that IMS Connect, an integrated component of IMS since Version 9, provides TCP/IP access to IMS applications. With the introduction of Open Database, its role has been extended to also provide the TCP/IP transport for direct access to IMS data. Its role is to transport and route DDM messages to the appropriate ODBM address space and return responses to clients via TCP/IP. It is a crucial component for distributed access with DRDA.

IBM IMS Connect Extensions for z/OS (IMS Connect Extensions) is an IMS Tool that provides event collection and instrumentation for IMS Connect. It collects information about outgoing and incoming requests that are transported via IMS Connect. This information is then available for analysis using standard IMS Tools such as IMS Performance Analyzer and IMS Problem Investigator.

In the context of Open Database, we can divide the information that IMS Connect Extensions collects into two categories:

- ▶ Framing events
- ▶ Application-level events

#### Framing events

These are events that mark significant points in the lifecycle of an Open Database request. Each such event contains a timestamp that gives you information about the relative timings and duration of the entire DRDA request. These events include:

- ▶ Opening of the TCP/IP socket
- ▶ Processing of the DRDA request
- ▶ Security authentication and authorization
- ▶ Dispatch of request to ODBM address space
- ▶ Response received from ODBM address space
- ▶ Processing of response
- ▶ Dispatching to client

- Sync-point processing (if RRS is used)

Such events help you understand the flow of an Open Database request and obtain timings not only for the overall request but for individual processing steps for the request.

### Application-level events

These events provide a record of exactly what request was sent by the client and what response was received back from IMS. They provide a complete breakdown of individual DDM objects and data. They help you debug and tune applications as well as audit and monitor exactly what activity is being performed by clients.

Figure 9-2 summarizes the value of this information.

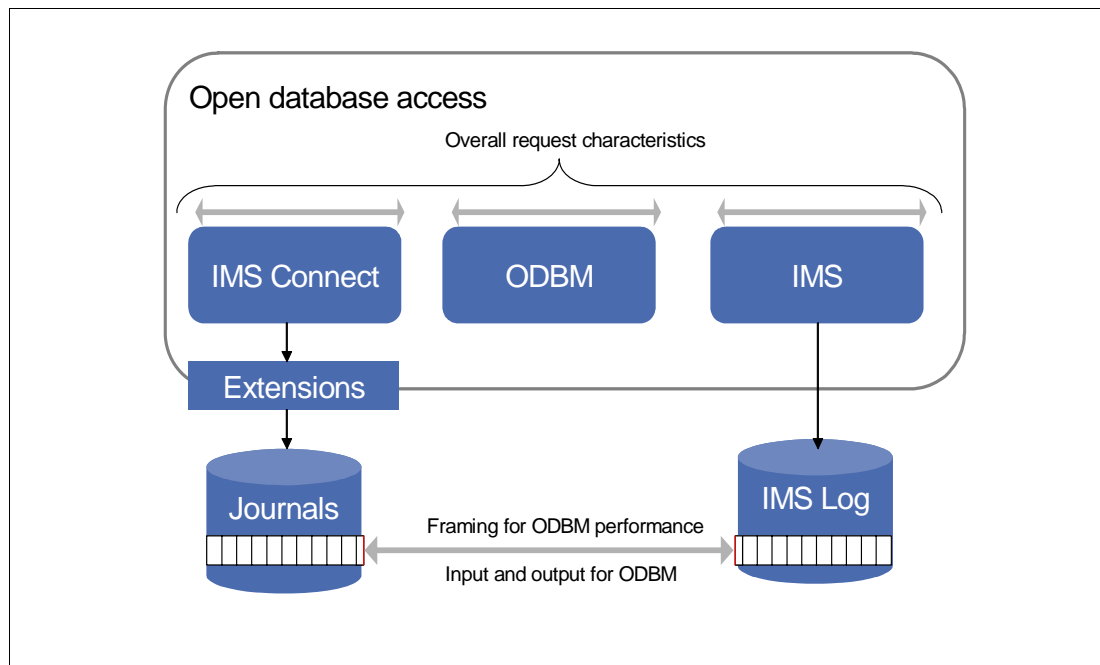


Figure 9-2 IMS Connect Extensions event collection

The information collected by IMS Connect Extensions can be analyzed using IMS Problem Investigator and IMS Performance Analyzer. Both tools can interpret the data and place it in the context of other information sources such as the IMS log. IMS Problem Investigator is focused on giving you detailed insight into the characteristics of individual requests, while IMS Performance Analyzer is focused on providing you aggregate characteristics of Open Database activity so that you can study and analyze the performance of certain request types, server nodes, and so forth.

## 9.4.2 IMS Problem Investigator

IBM IMS Problem Investigator for z/OS (IMS Problem Investigator) is a tool that allows you to analyze your IMS environment by interrogating IMS, CQS, and Monitor logs, DB2 logs, SMF data, information collected by OMEGAMON ATF and TRF, as well as the information collected in IMS Connect Extensions journals. In the Open Database context, we focus on the analysis of the IMS Connect Extensions journals.

By allowing you to navigate, format, query, and extract these journals you can better understand and gain insight into an Open Database request. IMS Problem Investigator can

automatically identify records related to a request sequence from an Open Database client and display this information tracking related log records.

In the example of Figure 9-3 we are tracking a sequence of Open Database requests. We can see framing events: the elapsed time between when the message was sent to ODBM and a response received, as well as information about the actual DDM objects.

File Menu Edit Mode Navigate Filter Time Labels Options Help			
-----			
BROWSE	CEX220.QAUNIT.EVNTLOG(WAS110)		Tracking inactive
Command ==>			Scroll ==> CSR
Forwards / Backwards . .	00.00.00.000100	Time of Day . .	16.46.22.845746
Code Description	Date 2009-06-29 Monday	Time (Elapsed)	
/	-----		
E	A049 READ Socket	09.52.05.464051	
	A049 READ Socket	0.000095	
	A05B DRDA 2001 ACCRDB-Access RDB	0.000008	
	A05D ODBM begin Allocate PSB (APSB) Program=AUTPSB11	0.000020	
	A061 ODBM Routing Exit called	0.000010	
	A062 ODBM Routing Exit returned	0.000364	
	A069 Message sent to ODBM	0.000945	
	A06A Message received from ODBM	0.430011	
	A05E ODBM end Allocate PSB (DPSB) Program=AUTPSB11	0.000386	
	A05C DRDA 2201 ACCRDBRM-Access RDB Reply Message	0.000025	
	A04A WRITE Socket	0.000167	
	A03C Prepare READ Socket	0.864489	
	A049 READ Socket	0.000122	
	A05B DRDA 200C OPNQR-Open Query	0.000009	
	A049 READ Socket	0.000024	
	A049 READ Socket	0.000026	
	A05B DRDA CC05 DLIFUNC-DL/I function	0.000008	

Figure 9-3 Tracking a sequence of Open Database requests

If you are primarily interested in where the request spent its time, you can use this elapsed time view to identify how the requests are being processed. In the above example, the most significant time is spent waiting on a response from ODBM and IMS (0.43 seconds) and between the client's first request and second request (0.86 seconds). Most other aspects of the request, such as security authorization and processing within IMS Connect itself, took milliseconds.

The A05B and A05C log records contain the most relevant information from an application's standpoint. Figure 9-4 shows how IMS Problem Investigator filtering allows you to view just these record types to give us the "story" of what the application was trying to do and how it was interacting with IMS itself:

File	Menu	Edit	Mode	Navigate	Filter	Time	Labels	Options	Help
-----									
BROWSE	CEX000.QADATA.REDBOOK.DRDATA110.ICON.D1003					Record	00000043	More: < >	
Command ==>						Scroll ==> CSR			
Forwards / Backwards . . 00.00.00.000100				Time of Day . . 16.46.22.845746					
Code Description				Date 2010-03-31 Wednesday		Time (LOCAL)			
/ -----									
A049	READ Socket					13.21.47.348142			
A05B	DRDA CC04 RTRVFLD-Field client wants to retrieve data					13.21.47.348149			
A049	READ Socket					13.21.47.348171			
A049	READ Socket					13.21.47.348194			
A05B	DRDA CC06 SSALIST-List of segment search argument					13.21.47.348202			
A0AA	ODBM Trace: Message sent to ODBM					13.21.47.348617			
A069	Message sent to ODBM					13.21.47.348627			
A0AA	ODBM Trace: Message received from ODBM					13.21.47.350167			
A06A	Message received from ODBM					13.21.47.350178			
A05C	DRDA 2205 OPNQRYM-Open Query Complete					13.21.47.350322			
A04A	WRITE Socket					13.21.47.350496			
A048	Trigger Event for ODBMMSG					13.21.47.350526			
A03C	Prepare READ Socket					13.21.48.120400			
A049	READ Socket					13.21.48.120456			
A05B	DRDA 2006 CNTQRY-Continue Query					13.21.48.120464			
A0AA	ODBM Trace: Message sent to ODBM					13.21.48.120628			
A069	Message sent to ODBM					13.21.48.120684			

Figure 9-4 Application records filtering

IMS Problem Investigator formats the standard “DDM” code points based on the DRDA specification as well as the IMS-specific code points. As shown above, we can see the initiation of the sync-point request, access request, the DL/I call, SSA list, and so forth. We can select these records to view more detail of what they contain.

In Figure 9-5 we can see the initiation of a request by an Open Database client.



```

File  Menu  Format  Help
-----
BROWSE      CEX000.QADATA.REDBOOK.DRDAT110.ICON.D Record 00000021 Line 00000019
Command ==>                                     Scroll ==> CSR
Form      ==>          +      Use Form in Filter      Format ==> FORM
+001D  Type..... 01          RQSCRR..... 0001

+0020  Object..... 2001 ACCRDB-Access RDB
+0020  Length..... +60          CP..... 2001

+0024  Object..... 2110 RDBNAM-Relational Database Name
+0024  Length..... +17          CP..... 2110
+0028  Data..... 'AUTPSB11.ODB1'

+0035  Object..... 210F RDBACCCL-RDB Access Manager Class
+0035  Length..... +6          CP..... 210F          Data..... 2407

+003B  Object..... 112E PRDID-Product-specific Identifier
+003B  Length..... +20          CP..... 112E
+003F  Data..... 'IMS OPEN DB V1.0'

+004F  Object..... 002F TYPDEFNAM-Data Type Definition Name
+004F  Length..... +13          CP..... 002F
+0053  Data..... 'QTDSQL370'

```

Figure 9-5 Tracing a request initiation

You can also view segment search arguments (SSA) as shown in Figure 9-6.

```

File  Menu  Format  Help
-----
BROWSE      CEX000.QADATA.REDBOOK.DRDAT110.ICON.D Record 00000047 Line 00000015
Command ==>                                     Scroll ==> CSR
Form      ==>          +      Use Form in Filter      Format ==> FORM
+0018  CERE_5B_VAR_CODEPOINT..... CC06

+001A  DSSHDR..... DSS header for DDM command
+001A  DSSlen..... +29          DDMID..... D0          FormatID... 03
+001D  Type..... 03          RQSCRR..... 0001

+0020  Object..... CC06 SSALIST-List of segment search argument
+0020  Length..... +23          CP..... CC06

+0024  Object..... C905 SSACOUNT-Number of segment search arguments
+0024  Length..... +6          CP..... C905          Data..... 0001

+002A  Object..... C906 SSA-Segment search argument
+002A  Length..... +13          CP..... C906
+002E  Data..... 'DEALER '
***** End of data *****

```

Figure 9-6 Viewing segment search arguments

Importantly, you can also view exactly what the ODBM address space is sending as a response to the client. This allows you to isolate problems in parsing and interpreting output from IMS, as shown in the example of Figure 9-7.

```

File  Menu  Format  Help
-----
BROWSE      CEX000.QADATA.REDBOOK.DRDAT110.ICON.D Record 00000062 Line 00000019
Command ==>                                     Scroll ==> CSR
Form      ==>      +      Use Form in Filter      Format ==> FORM
+001D  Type..... 03      RQSCRR..... 0001

+0020  Object..... 241B QRYDTA-Query Answer Set Data
+0020  Length..... +113      CP..... 241B
+0024  AIB..... aibStream
+0024  AIBflag.... 00      AIBused.... +61      AIBretc.... 00000000
+002D  AIBreas.... 00000000  AIBerrc.... 00000000
+0035  DBPCB..... dbpcbStream
+0035  DBPflag.... 00      DBflag..... 00
+0037  DBname..... 'AUTOLDB '  SL..... '01'      SC..... ' '
+0043  Segment.... 'DEALER '  KFBAflag... 00      KFBAlen.... +4
+0050  KFBA..... '1234'
+0054  IOarea..... IO area
+0000  F1F2F3F4 E2C1D540 D1D6E2C5 40C6D6D9  *1234SAN JOSE FOR*
+0010  C4404040 40404040 40404040 40404040  *D      *
+0020  4040E2C1 D540D1D6 E2C54040 F9F5F7F7  * SAN JOSE 9577*
+0030  F760F3F3 F3F3F7F7 F7F4F4F4 F4      *7-33337774444 *
***** End of data *****

```

Figure 9-7 ODBM response tracking

If you need to understand specific flags or fields, you can zoom on them to get more information, as shown in Figure 9-8.

```

+----- Field Zoom -----+
File  Menu  Help
-----
BROWSE      CEX220.QAUNIT.EVNTLOG(WAS110)      Line 00000000
Command ==>                                     Scroll ==> CSR
***** Top of data *****
+0024  AIBflag.... 00  AIB null indicator

On    Present.... 00  aibStream data structure is present
Off   Null..... FF  aibStream data structure contains no data after
                        the AIB null indicator. The total length of the
                        aibStream data structure is one byte.
***** End of data *****

```

Figure 9-8 Zooming on specific fields

Other than browsing and drilling down on information specific to particular requests, you can also use filters to identify requests with particular characteristics. For example, identify requests arriving from a given server or for a given database.

### 9.4.3 Identifying and resolving problems

This topic shows how IMS Tools can be used to identify and resolve common problem with the sample applications. You can classify Open Database issues into the following categories:

- ▶ Session errors: these are conditions that generate distinct error. For example, specifying the wrong alias name, trying to access a stopped PSB.
- ▶ Performance problems: IMS provides an output but processing time is slow.
- ▶ Unexpected responses: the client receives information from IMS but it is not the feedback that the client was expecting.

Previously, we have shown that IMS Problem Investigator can be used to examine the requests the client sends, how they are processed by IMS, and the replies that are returned, allowing you to understand the context of unexpected responses. We now discuss session errors and performance problems.

#### Session errors

When a session error occurs, IMS Connect Extensions writes an A047 record to the journal. You can use IMS Problem Investigator's filtering to identify any session errors in the journal. In the example of Figure 9-9, it is a simple filter that shows only A047 records in the journal.

File	Menu	Edit	Mode	Navigate	Filter	Time	Labels	Options	Help
-----									
BROWSE	CEX000.QADATA.REDBOOK.ERR01.ICON.D100331							Tracking inactive	
Command ==>							Scroll ==> CSR		
Forwards / Backwards . .				00.00.00.000100		Time of Day . .		16.46.22.845746	
Code Description				Date 2010-03-31 Wednesday		Time (LOCAL)			
/ -----									
A047 Session Error							11.54.55.710442		
A047 Session Error							12.03.02.490039		
A047 Session Error							12.32.09.670281		
***** Bottom of Data *****									

Figure 9-9 Filtering by A047 records

The log record itself often contains enough information to understand the cause of the problem. For example, as shown in Figure 9-10, if you select the last session error, you can see the message area, identifying that the client requested RRS when RRS was not available.

```

File  Menu  Format  Help
-----
BROWSE      CEX000.QADATA.REDBOOK.ERR01.ICON.D100 Record 00000344 Line 00000010
Command ==>                                     Scroll ==> CSR
Form      ==>      +      Use Form in Filter      Format ==> FORM
+000A CERE_47_TASKID..... ID of task recording event
+000A CERE_47_COL#..... 01 CERE_47_TKS#..... 04
+000C CERE_47_EVKEY..... C5C21834F1A8FC61
+0014 CERE_47_VAR_LL..... 009C
+0016 CERE_47_VAR_APAR... 0001
+0018 CERE_47_VAR_FLAG3..... 80
+001A CERE_47_VAR_MSG.... 134 byte message area
      +0000 00640000 C8E6E2D2 F2F8F8F0 C540D9D9 *...HWSK2880E RR*
      +0010 E240C3D6 D4D4C1D5 C440C6C1 C9D3C5C4 *S COMMAND FAILED*
      +0020 5E40C37E D6C4C2F2 C5F4F2F3 6B40C3D7 *; C=0DB2E423, CP*
      +0030 7EE2E8D5 C3C3E3D3 4040406B 40D77EF4 *=SYNCCTL , P=4*
      +0040 F8F8F5F5 4040406B 40D97EF0 F0F0F46B *8855 , R=0004,*
      +0050 40D9E27E D9D9E2D5 C1E5C9D3 6B40D47E * RS=RRSNAVIL, M=*
      +0060 D4D9C3E5 00000000 00000000 00000000 *MRCV.....*
      +0070 00000000 00000000 00000000 00000000 *.....*
      +0080 00000000 0000 *.....*
+00A0 CERE_47_VAR_SESRN..... 'WRITE '
+00A8 CERE_47_VAR_TOKEN..... 0000000000000000
***** End of data *****

```

Figure 9-10 Displaying the message area

In some cases you may be interested in examining the complete flow of a request that generated a session error. In such cases, you can use tracking (TX line action) to reveal all the event records associated with a particular session error as shown in Figure 9-11.

In this case, you can see the progression of the request leading up to the session error. The client successfully completes security authentication, but when the Access RDB request is made, triggering the allocation of the PSB, the request then receives an RDB Not Found DRDA object. This finally triggers the session error.

File	Menu	Edit	Mode	Navigate	Filter	Time	Labels	Options	Help
-----									
BROWSE	CEX000.QADATA.REDBOOK.ERR01.ICON.D100331					Record	00000013	More: < >	
Command ==>						Scroll	==> CSR		
Forwards / Backwards . . 00.00.00.000100				Time of Day . . 16.46.22.845746					
Code Description				Date 2010-03-31 Wednesday		Time (Relative)			
/ -----									
TX	A049	READ Socket					-0.256794		
	A05B	DRDA 106E SECCHK-Security Check					-0.256786		
	A063	ODBM Security Exit called					-0.256755		
	A064	ODBM Security Exit returned					-0.256668		
	A05C	DRDA 1219 SECCHKRM-Security Check Reply Message					-0.256594		
	A04A	WRITE Socket					-0.256516		
	A049	READ Socket					-0.000293		
	A049	READ Socket					-0.000223		
	A05B	DRDA 2001 ACCRDB-Access RDB					-0.000216		
	A05D	ODBM begin Allocate PSB (APSB) Program=AUTPSB11					-0.000194		
	A061	ODBM Routing Exit called					-0.000185		
	A062	ODBM Routing Exit returned					-0.000033		
	A05C	DRDA 2211 RDBNFNRM-RDB Not Found					11.54.55.710366		
	A04A	WRITE Socket					+0.000064		
	A047	Session Error					+0.000075		
	A00C	Begin CLOSE Socket					+0.000108		
	A00D	End CLOSE Socket					+0.000325		

Figure 9-11 Tracking the complete flow

## Performance problems

Using IMS Problem Investigator, you can map relative timing of an Open Database request and broadly identify how much time requests spend in various stages of processing. That is:

- ▶ Client
- ▶ IMS Connect
- ▶ ODBM
- ▶ IMS

The log records available for IMS (often the area where critical insight is required) are similar to those available for CICS DBCTL transactions. As such, they provide only limited information on the timings of the DL/I calls themselves. If the requests are predominantly queries and not updates, very little information is actually available in the IMS logs.

To mitigate this, you will need to use database trace such as the IMS Monitor. However, the most detailed trace is available from the OMEGAMON for IMS on z/OS application trace facility (ATF). ATF allows you to record DL/I call characteristics. IMS Tools allow you to connect between the DLI call activity and the DRDA requests that are being passed by the client, as shown in Figure 9-12.

In this example, you can see the Open Database requests and the DLI calls themselves, including duration of each call and CPU utilization for each call.

File	Menu	Edit	Mode	Navigate	Filter	Time	Labels	Options	Help
-----									
BROWSE	CEX000.QADATA.REDBOOK.DRDAT111.ICON.D1003					Record	00000308	More: < >	
Command ==>						Scroll ==> CSR			
Forwards / Backwards . . 00.00.00.000100				Time of Day . . 16.46.22.845746					
Code Description				Date 2010-03-31 Wednesday		Time (LOCAL)			
/ -----									
A049	READ Socket					13.46.47.095038			
A05B	DRDA CC06 SSALIST-List of segment search argument					13.46.47.095045			
A0AA	ODBM Trace: Message sent to ODBM					13.46.47.095985			
A069	Message sent to ODBM					13.46.47.096016			
06	OSAM IWAIT start TranCode=ODBA02CD Region=0003					13.46.47.142891			
20	Database Open Database=EMPDB2 Region=0003					13.46.47.143647			
06	OSAM IWAIT start TranCode=ODBA02CD Region=0003					13.46.47.181506			
20	Database Open Database=AUTODB Region=0003					13.46.47.182252			
06	OSAM IWAIT start TranCode=ODBA02CD Region=0003					13.46.47.191442			
01	DLI GHU Database=EMPLDB2 SC=' ' Elapse=0.095875					13.46.47.096570			
B021	DLI Database Trace Database=EMPLDB2 Func=GHU					13.46.47.192378			
A0AA	ODBM Trace: Message received from ODBM					13.46.47.192881			
A06A	Message received from ODBM					13.46.47.192909			
A05C	DRDA 2205 OPNQRYRM-Open Query Complete					13.46.47.193186			
A04A	WRITE Socket					13.46.47.193515			
A048	Trigger Event for ODBMSG					13.46.47.193554			
A03C	Prepare READ Socket					13.46.48.120636			

Figure 9-12 DRDA and DLI flow

You can control the amount of information IMS Problem Investigator shows: to view extended details, scroll right (F11). See Figure 9-13.

```
File  Menu  Edit  Mode  Navigate  Filter  Time  Labels  Options  Help
-----
BROWSE      CEX000.QADATA.REDBOOK.DRDAT111.ICON.D1003  Record 00000316 More: < >
Command ==>                                         Scroll ==> CSR
Forwards / Backwards . . 00.00.00.000100      Time of Day . . 16.46.22.845746
Code Description                                Date 2010-03-31 Wednesday  LSN
/ -----
06  OSAM IWAIT start                                2-000000000000004
    TranCode=ODBA02CD Program=AUTPSB11 Userid=AUTPSB11 Region=0003
    IMSID=ODBA02CD RecToken=ODBA02CD/0000000100000000 Elapse=0.000725
-----
01  DLI GHU                                          2-000000000000005
    TranCode=ODBA02CD Program=AUTPSB11 Userid=AUTPSB11 Database=EMPLDB2
    Region=0003 IMSID=ODBA02CD RecToken=ODBA02CD/0000000100000000 SC=' '
    Elapse=0.095875 CPU=0.002190
-----
B021 DLI Database Trace                            3-000000000000105
    LTerm=EMPLDB2 Database=EMPLDB2 Region=0003
    RecToken=ODBA02CD/0000000100000000 Func=GHU Elapsed=00.095631
-----
A0AA ODBM Trace: Message received from ODBM        1-000000000000359
    IMSID=IBDEODOD LogToken=C5C228E17B387162
-----
A06A Message received from ODBM                    1-00000000000035A
```

Figure 9-13 Displaying detailed information with F11t

As shown in Figure 9-14, if you select the ATF generated 01 record you can see the IO area used for the request itself.

```

File  Menu  Format  Help
-----
BROWSE      CEX000.QADATA.REDBOOK.DRDAT111.ICON.D Record 00000317 Line 00000042
Command ==>                                     Scroll ==> CSR
Form      ==>          +      Use Form in Filter          Format ==> FORM
+00DC  ATRDXEL.... DL/I Trace Element
+00DC  ATRDX@E.... 14ABD400  ATRDXTY.... 02          ATRDXF..... 00
+00E2  ATRDX#..... 0008
+00E4  ATRDXV..... Element Data - Key Feedback Area
      +0000  F2F2F2F2 F2F2C3C1          *222222CA          *

+00EC  ATRDXEL.... DL/I Trace Element
+00EC  ATRDX@E.... 14AAD2E8  ATRDXTY.... 01          ATRDXF..... 00
+00F2  ATRDX#..... 004C
+00F4  ATRDXV..... Element Data - I/O Area
      +0000  F2F2F2F2 F2F2C299 96A69540 40404040  *222222Brown          *
      +0010  40404040 40404040 40404040 404040D9  *          R*
      +0020  96954040 40404040 40404040 40404040  *on          *
      +0030  40404040 40404040 E2D6D4C5 40E2E3D9  *          SOME STR*
      +0040  C5C5E340 40404040 40404040          *EET          *

+0140  ATRDXEL.... DL/I Trace Element
+0140  ATRDX@E.... 14AAD540  ATRDXTY.... 04          ATRDXF..... 00
+0146  ATRDX#..... 0036

```

Figure 9-14 Displaying the I/O area

## 9.5 Additional sample programs

If you need more examples for your understanding of using the IMS Universal DB drivers in your application you will find several self-explaining samples available for download on the web.

If you want to use the **CSLDMI interface** for writing an application you will find a full description of the CSLDMI interface in section “Writing a CSL ODBM client” of *IMS Version 11 System Programming APIs*, SC19-2445. An example of using this can be found in section “C.2 Sample ODBM program” of *IMS Version 11 Technical Overview*, SG24-7807.

If you want to use a **REXX script** for accessing IMS Connect with Open Database you will find an example in section “C.1 Sample DRDA program” of *IMS Version 11 Technical Overview*, SG24-7807.

In addition to the examples used in this book, we provide some more samples as additional material. For information about how to download the samples see Appendix D, “Additional material” on page 249.





**A**

# IBM DB2 Data Server Drivers and Clients

This appendix provides an overview of the DB2 Drivers which could be used to access DB2 data from the same client Java application using IMS drivers to use DL/I data.

For more information, refer to *DB2 9 for z/OS: Distributed Functions*, SG24-6952-01, which is the base for this appendix.

The IBM DB2 Data Server Driver for JDBC and SQLJ (formerly known as IBM DB2 Driver for JDBC and SQLJ) is a single application driver to support the most demanding Java applications. This driver can be used as in type 4 or type 2 mode, includes support for pureXML®, SQL/XML and XQuery, and it is optimized for DB2 across all platforms (Linux, UNIX, Windows, z/OS and iSeries®) and Indormix Dynamic Server (IDS).

## A.1 IBM Data Server Drivers and Clients

IBM strategy is to promote the usage of IBM Data Server Drivers or Clients. DB2 Connect™ licenses (in the form of DB2 Connect license files) are still required, but you can replace DB2 Connect modules with the IBM Data Server Drivers or Clients and receive equivalent or superior function. In addition, you can reduce complexity, improve performance, and deploy application solutions with smaller footprints for your business users.

With DB2 for LUW Version 9.5 FixPack 3 or FixPack 4 you can implement the DRDA AR functions for your distributed applications with varied degrees of granularity. Instead of the current function and large footprint of DB2 Connect, you can choose one of the IBM client products.

IBM has delivered a variety of client products for application developers and database administrators to support distributed access to data stored in DB2 for z/OS.

Here is the list of the IBM Data Server Clients and Drivers:

- ▶ IBM Data Server Client
- ▶ IBM Data Server Runtime Client
- ▶ IBM Data Server Driver for ODBC and CLI
- ▶ IBM Data Server Driver for JDBC and SQLJ
- ▶ IBM Data Server Driver Package

In the IMS scenario in this book, we are using the IBM Data Server Driver for JDBC and SQLJ but you can use any of the Driver which supports JDBC (see Appendix A.1.6, “Driver and Client comparison” on page 231). To get the driver you need the Driver itself which you can download using your IBM ID from the following website:

<http://www.ibm.com/support/docview.wss?rs=4020&uid=swg21385217>

Additional if you want to connect to DB2 on z/OS you will need the correct license jar file.

In this appendix we briefly describe these drivers and clients and provide a table that compares the driver and client products. Refer to the IBM DB2 9.7 Information Center at the following link for more information on these products:

<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp?topic=/com.ibm.swg.im.dbclient.install.doc/doc/c0022612.html>

### A.1.1 IBM Data Server Driver for JDBC and SQLJ

JDBC is an application programming interface (API) that Java applications use to access relational databases. SQLJ provides support for embedded static SQL in Java applications. In general, Java applications use JDBC for dynamic SQL and SQLJ for static SQL.

Note that this driver is also called the JAVA Common Client (JCC) driver and was formerly known as the IBM DB2 Universal Database Driver. The DB2 JDBC Type 2 driver for LUW, also called the CLI legacy driver, has been deprecated.

We can refer to these clients as Java-based clients.

See the Java application development for IBM data servers section at the DB2 Version 9.5 for Linux, UNIX, and Windows Information Center:

<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.apdv.java.doc/doc/c0024189.html>

### **DB2 support for JDBC drivers**

The IBM Data Server Driver for JDBC and SQLJ provides Type 4 and Type 2 connectivity. To communicate with remote servers using DRDA, the Type 4 driver is used as a DRDA Application Requester..

► Driver for JDBC Type 2 connectivity (Type 2 driver)

Type 2 drivers are written partly in the Java programming language and partly in native code. The drivers use a native client library specific to the data source to which they connect. Because of the native code, their portability is limited. Use of the Type 2 driver to connect to DB2 for z/OS is recommended for WebSphere Application Server running on System z.

► Driver for JDBC Type 4 connectivity (Type 4 driver)

Type 4 drivers are pure Java and implement the network protocol for a specific data source. The client connects directly to the data source. The JDBC Type 4 driver is recommended to connect distributed Java applications to DB2 for z/OS data.

IBM ships two versions of the JDBC Type 4 driver with the IBM Data Server Driver for JDBC and SQLJ V9.5 FP3 product:

- Version 3.5x is JDBC 3.0-compliant. It is packaged as db2jcc.jar and sqlj.zip and provides JDBC 3.0 and earlier support.
- Version 4.x is JDBC 3.0-compliant and supports some JDBC 4.0 functions. It is packaged as db2jcc4.jar and sqlj4.zip.

The Type 4 driver provides support for distributed transaction management. This support implements the Java 2 Platform, Enterprise Edition (J2EE), Java Transaction Service (JTS), and Java Transaction API (JTA) specifications, which conform to the X/Open standard for distributed transactions (Distributed Transaction Processing: The XA Specification, available from:

<http://www.opengroup.org>

Applications that use the IBM Data Server Driver for JDBC and SQLJ to access DB2 for z/OS data across a network implement the Type 4 driver. In Figure A-1 an application uses the IBM Data Server Driver for JDBC and SQLJ to access a standalone DB2.

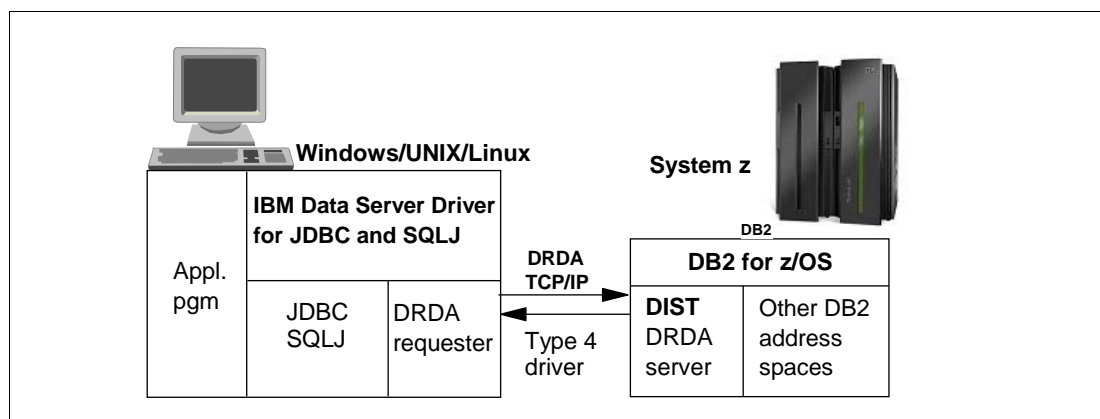


Figure A-1 IBM Data Server Driver for JDBC and SQLJ connecting directly to DB2 for z/OS

In the remainder of the discussion and examples of the IBM Data Server Driver for JDBC and SQLJ, we refer to the Type 4 driver support.

### A.1.2 IBM Data Server Driver for ODBC and CLI (CLI driver)

This product is for applications using ODBC or CLI only and provides a lightweight deployment solution designed for ISV deployments. This driver, also referred to as 'CLI driver', provides runtime support for applications using the ODBC API or CLI API, without the need of installing the IBM Data Server Client or the IBM Data Server Runtime Client.

The CLI driver is conceptually similar to the JDBC Type 4 driver. The CLI driver is packaged in a small footprint, providing the DRDA AR functions necessary to connect to DB2 for z/OS for those application scenarios where you do not require robust tools, development or administration functions.

### A.1.3 IBM Data Server Driver Package

IBM Data Server Driver Package provides a lightweight deployment solution providing runtime support for applications using ODBC, CLI, .NET, OLE DB, open source, or Java APIs without the need of installing Data Server Runtime Client or Data Server Client. This driver has a small footprint and is designed to be redistributed by independent software vendors (ISVs), and to be used for application distribution in mass deployment scenarios typical of large enterprises.

The IBM Data Server Driver Package capabilities include:

- ▶ Support for applications that use ODBC, CLI, or open source (PHP or Ruby) to access databases.
- ▶ Support for client applications and applets that are written in Java using JDBC, and for embedded SQL for Java (SQLJ).
- ▶ IBM Informix Dynamic Server support for .NET, PHP, and Ruby.
- ▶ Application header files to rebuild the open source drivers.
- ▶ Support for DB2 Interactive Call Level Interface (db2cli).
- ▶ On Windows operating systems, IBM Data Server Driver Package also provides support for applications that use .NET or OLE DB to access databases. In addition, this driver is available as an installable image, and a merge module is available to allow you to easily embed the driver in a Windows Installer-based installation.
- ▶ On Linux and UNIX operating systems, IBM Data Server Driver Package is not available as an installable image.

### A.1.4 IBM Data Server Runtime Client

This product allows you to run applications on remote databases. Graphical user interface (GUI) tools are not included. Capabilities include:

- ▶ command line processor (CLP)
- ▶ base client support for database connections, SQL statements, XQuery statements and commands
- ▶ support for common database access interfaces (JDBC, SQLJ, ADO.NET, OLE DB, ODBC, command line interface (CLI), PHP and Ruby), including drivers and ability to define data sources
- ▶ Lightweight Directory Access Protocol (LDAP) exploitation
- ▶ support for TCP/IP and Named Pipe
- ▶ support for multiple concurrent copies and various licensing and packaging options

The IBM Data Server Runtime Client (Runtime Client) has a rich set of SQL APIs for deployment in more complex application environments.

## A.1.5 IBM Data Server Client

This is the full-function product for application development, database administration and client/server configuration. Capabilities include:

- ▶ Configuration Assistant
- ▶ Control Center and other graphical tools
- ▶ First Steps for new users
- ▶ Visual Studio tools
- ▶ IBM Data Studio
- ▶ Application header files
- ▶ Precompilers for various programming languages
- ▶ Bind support
- ▶ All The functions included in the IBM Data Server Runtime Client

## A.1.6 Driver and Client comparison

The highlights of the IBM Data Server products are summarized in Table A-1. Refer to standard DB2 for LUW product documentation for additional details.

Table A-1 IBM Data Server Drivers and Clients comparison

Product	Smallest footprint	JDBC and SQLJ	ODBC and CLI	OLE DB and .NET	Open Source	CLP	DBA, Dev, GUI tools
IBM Data Server Driver for JDBC and SQLJ	X	X					
IBM Data Server Driver for ODBC and CLI	X		X				
IBM Data Server Driver Package		X	X	X	X		
IBM Data Server Runtime Client		X	X	X	X	X	
IBM Data Server Client		X	X	X	X	X	X

For the correct version of the driver see  
<http://www.ibm.com/support/docview.wss?rs=71&uid=swg21363866>

## A.2 Support for JDBC and SQLJ

In order to develop a Java application from your client you need the appropriate level of IBM Software Development Kit (SDK) for Java or DB2 for Linux, UNIX, and Windows. This is true for to use Java-based tools and to create and run Java applications, including stored procedures and user-defined functions.

If the IBM SDK for Java is required by a component being installed and the SDK for Java is not already installed in that path, the SDK for Java is installed if you use either the DB2 Setup wizard or a response file to install the product.

Refer to the Application Development with DB2 Web site for details:

<http://www.ibm.com/software/data/db2/ad/java.html>

If you need to provide support for JDBC and SQLJ requesters for the first time, there are several steps you must complete. Most of these steps are for DB2 for z/OS as the DRDA AS, but some relate to the requesters. The steps to install support for JDBC and SQLJ are:

1. Allocate and load IBM Data Server Driver for JDBC and SQLJ libraries. You perform this step on the client(s).
2. On DB2 for z/OS, set the DESCSTAT parameter to YES (DESCRIBE FOR STATIC on the DSNITPF installation panel). This is necessary for SQLJ support.
3. In z/OS USS, edit the .profile file to customize environment variable settings.
4. Optional: customize IBM Data Server Driver for JDBC and SQLJ configuration properties. This refers to the properties on the clients.
5. On DB2 for z/OS, enable the DB2-supplied stored procedures and define the tables that are used by the IBM Data Server Driver for JDBC and SQLJ.
6. In z/OS UNIX System Services, run the DB2Binder utility to bind the packages for the IBM Data Server Driver for JDBC and SQLJ.
7. Install z/OS Application Connectivity to DB2 for z/OS feature. This applies if you have a JDBC or SQLJ application on z/OS in an LPAR where you do not have a DB2 for z/OS subsystem. This feature is effectively the Type 4 driver for z/OS and provides the DRDA AR function without a local DB2 (and DDF) to communicate with a DB2 for z/OS server elsewhere in the network.

Refer to *DB2 Version 9.1 for z/OS Installation Guide*, GC18-9846 for more information on these steps.

## A.3 Using the IBM Data Server Driver for JDBC and SQLJ

We focus on the Type 4 driver in this section since it can be used to connect to a DB2 for z/OS server via DRDA. The Type 4 driver is now delivered in the jcc3 and jcc4 streams. The jcc4 stream requires JDK 1.6 to be installed.

You also need to customize and run the DSNITJMS job that creates stored procedures and tables required by the Type 4 driver. Refer to the following links for complete information:

<http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/topic/com.ibm.db2.luw.apdv.java.doc/doc/t0024156.html>

<http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/topic/com.ibm.db2.luw.apdv.java.doc/doc/c0052041.html>

To know which version of the driver you are using, use the command in Example A-1 when your driver is in your Windows class path

*Example A-1 Determining the driver version*

---

```
C:\DDF\test>java com.ibm.db2.jcc.DB2Jcc -version
IBM Data Server Driver for JDBC and SQLJ 4.8.23
```

---

### Connecting to a DB2 for z/OS server using the Type 4 driver

You can use either the DriverManager or the DataSource interface to obtain a connection to the database server. Here is an example using the DriverManager.getConnection() interface

where the connection properties are embedded in the application program. Example A-2 shows the usage of the `getConnection()` interface to obtain a connection to the DB2 for z/OS server.

*Example A-2 Using the `getConnection()`*

---

```
String url = "jdbc:db2://wtsc63.itso.ibm.com:12347/DB9A";
java.util.Properties prop = new java.util.Properties();
prop.put("user", user);
prop.put("password", password);
prop.put("driverType", "4");
conn1 = DriverManager.getConnection(url,prop);
```

---

It is possible to create and use a `DataSource` object in the same application similar to the `DriverManager` interface, this method does not provide portability. The recommended way to use a `DataSource` object is for your system administrator to create and manage it separately, using WebSphere Application Server or some other tool. It is then possible for your system administrator to modify the data source attributes and you do not need to change your application program. Example A-3 shows the use of the `DataSource` interface to obtain a connection to the DB2 for z/OS server.

*Example A-3 Connecting to DB2 for z/OS via the `DataSource` interface*

---

```
DB2SimpleDataSource dataSource = new com.ibm.db2.jcc.DB2SimpleDataSource();
dataSource.setServerName (servername);
dataSource.setPortNumber (Integer.parseInt(port));
dataSource.setDatabaseName (databasename);
dataSource.setUser (user);
dataSource.setPassword (password);
dataSource.setDriverType (4);
conn1 = dataSource.getConnection();
```

---

For details on connections using the `DriverManager` or `DataSource` interfaces, refer to:

<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.apdv.java.doc/doc/cjvjdcon.html>

To learn more about using WebSphere to deploy `DataSource` objects, use the link:

<http://www.ibm.com/software/webservers/appserv/>

In general, the default properties enabled for the Type 4 driver are designed to maximize distributed performance. If you want to change either the configuration properties that have driver-wide scope, or Connection/Datasource properties that are application-specific, refer to the following link.

<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.apdv.java.doc/doc/rjvdsprp.html>







## **Car Dealer IVP Database**

The Car Dealer database is used in most of the examples in this book. This database is part of the Installation Verification Procedure (IVP) of IMS. It should be readily available in your IMS environments, usually in test environments rather than production environments.

## B.1 Car Dealer database overview

The Car Dealer database uses mainly two physical databases (autos and employees) and two indexes. The database builds a logical database across these two databases. We do not use the indexes in our example so we skipped them in the following sources and overviews.

### B.1.1 AUTOLPCB overview diagram

Figure B-1 provides an overview diagram of the AUTOLPCB.

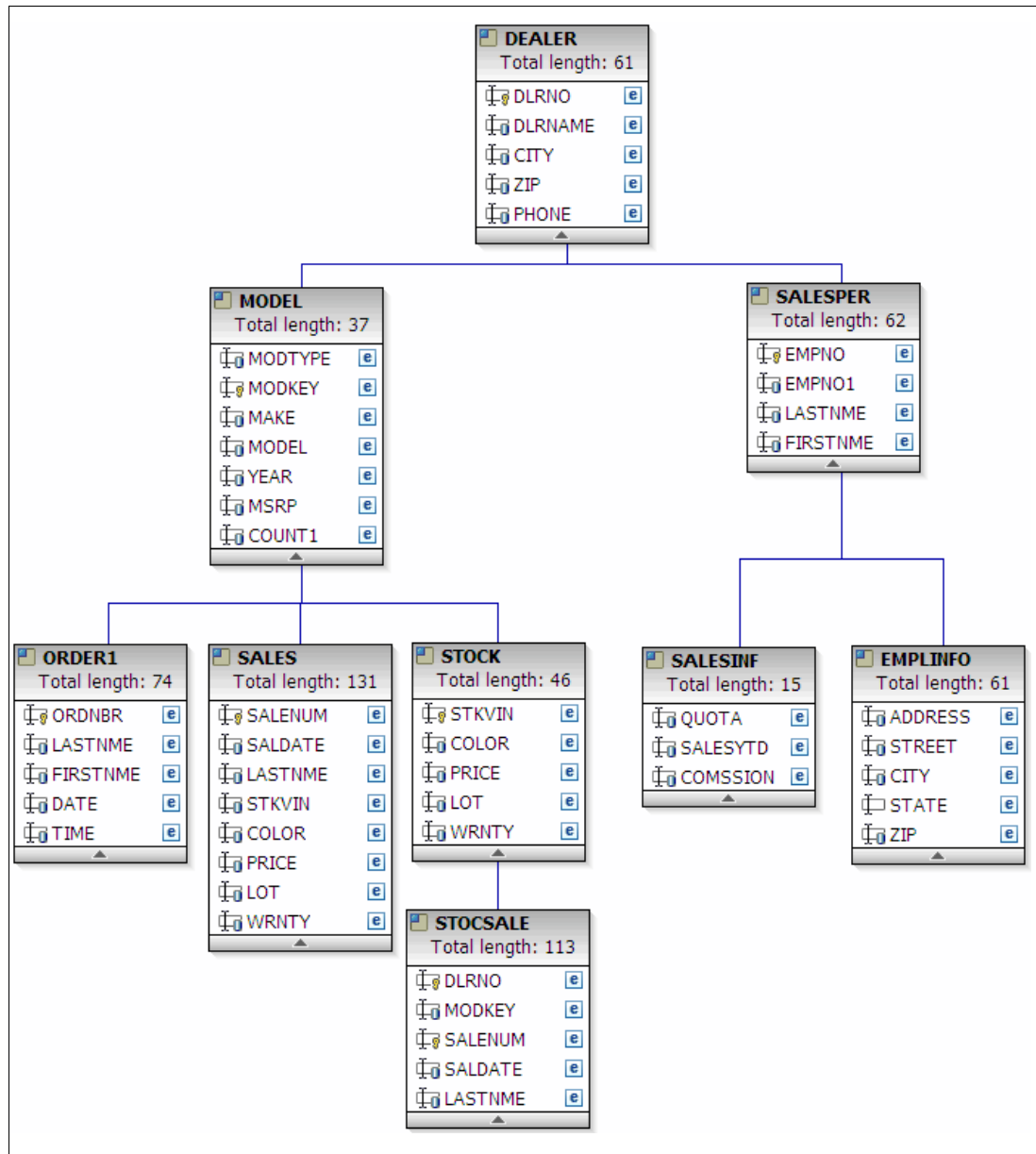


Figure B-1 AUTOLPCB Overview diagram

### B.1.2 EMPLPCB overview diagram

Figure B-2 provides an overview diagram of the EMPLPCB.

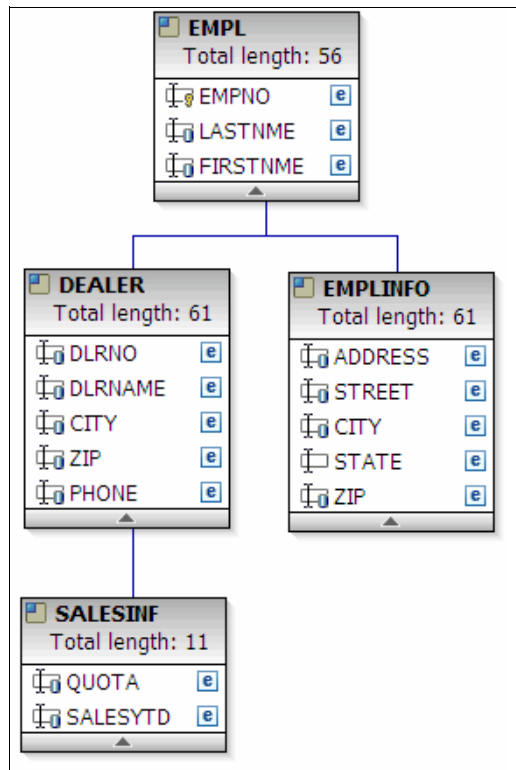


Figure B-2 EMPLPCB Overview diagram

### B.1.3 Metadata description

Example B-1 shows the used segments and field of the database unmodified as CHAR fields:

#### Example B-1 DLIModel IMS Java Report

Class: AUTPSB11DatabaseView in package: samples.ims.openDb generated for PSB: AUTPSB11

=====

PCB: AUTOLPCB

=====

Segment: DEALER

Field: DLRNO	Type=CHARLength=4++	Primary Key Field ++
Field: DLRNAME	Type=CHARLength=30	(Search Field)
Field: CITY	Type=CHARLength=10	(Search Field)
Field: ZIP	Type=CHARLength=10	(Search Field)
Field: PHONE	Type=CHARLength=7	(Search Field)

=====

Segment: MODEL

Field: MODTYPE	Type=CHARLength=2	(Search Field)
Field: MODKEY	Type=CHARLength=24++	Primary Key Field ++
Field: MAKE	Type=CHARLength=10	(Search Field)
Field: MODEL	Type=CHARLength=10	(Search Field)
Field: YEAR	Type=CHARLength=4	(Search Field)
Field: MSRP	Type=CHARLength=5	(Search Field)
Field: COUNT1	Type=CHARLength=2	(Search Field)

=====

Segment: ORDER1

Field: ORDNR	Type=CHARLength=6++	Primary Key Field ++
Field: LASTNME	Type=CHARLength=25	(Search Field)
Field: FIRSTNME	Type=CHARLength=25	(Search Field)
Field: DATE	Type=CHARLength=10	(Search Field)

```

      Field: TIME      Type=CHARLength=8      (Search Field)
=====
Segment: SALES
      Field: SALENUM   Type=CHARLength=4++ Primary Key Field ++
      Field: SALDATE   Type=CHARLength=8      (Search Field)
      Field: LASTNME   Type=CHARLength=25     (Search Field)
      Field: STKVIN     Type=CHARLength=20     (Search Field)
      Field: COLOR      Type=CHARLength=10     (Search Field)
      Field: PRICE      Type=CHARLength=5      (Search Field)
      Field: LOT        Type=CHARLength=10     (Search Field)
      Field: WRNTY      Type=CHARLength=1      (Search Field)
=====
Segment: STOCK
      Field: STKVIN     Type=CHARLength=20++ Primary Key Field ++
      Field: COLOR      Type=CHARLength=10     (Search Field)
      Field: PRICE      Type=CHARLength=5      (Search Field)
      Field: LOT        Type=CHARLength=10     (Search Field)
      Field: WRNTY      Type=CHARLength=1      (Search Field)
=====
      Segment: STCSALE
      Field: DLRNO      Type=CHARLength=4++ Primary Key Field ++
      Field: MODKEY      Type=CHARLength=24     (Search Field)
      Field: SALENUM     Type=CHARLength=4++ Primary Key Field ++
      Field: SALDATE     Type=CHARLength=8      (Search Field)
      Field: LASTNME     Type=CHARLength=25     (Search Field)
=====
Segment: SALESPER
      Field: EMPNO      Type=CHARLength=6++ Primary Key Field ++
      Field: EMPN01     Type=CHARLength=6      (Search Field)
      Field: LASTNME     Type=CHARLength=25     (Search Field)
      Field: FIRSTNME    Type=CHARLength=25     (Search Field)
=====
      Segment: SALESINF
      Field: QUOTA       Type=CHARLength=5      (Search Field)
      Field: SALESYTD    Type=CHARLength=5      (Search Field)
      Field: COMSSION    Type=CHARLength=5      (Search Field)
=====
      Segment: EMPLINFO
      Field: ADDRESS     Type=CHARLength=61     (Search Field)
      Field: STREET      Type=CHARLength=25     (Search Field)
      Field: CITY         Type=CHARLength=25     (Search Field)
      Field: STATE        Type=CHARLength=2++ Primary Key Field ++
      Field: ZIP          Type=CHARLength=9      (Search Field)
=====
PCB: EMPLPCB
=====
Segment: EMPL
      Field: EMPNO      Type=CHARLength=6++ Primary Key Field ++
      Field: LASTNME     Type=CHARLength=25     (Search Field)
      Field: FIRSTNME    Type=CHARLength=25     (Search Field)
=====
      Segment: DEALER
      Field: DLRNO      Type=CHARLength=4      (Search Field)
      Field: DLRNAME     Type=CHARLength=30     (Search Field)
      Field: CITY         Type=CHARLength=10     (Search Field)
      Field: ZIP          Type=CHARLength=10     (Search Field)
      Field: PHONE       Type=CHARLength=7      (Search Field)
=====
      Segment: SALESINF
      Field: QUOTA       Type=CHARLength=5      (Search Field)

```

```

Field: SALESYTD  Type=CHARLength=5      (Search Field)
=====
Segment: EMPLINFO
Field: ADDRESS  Type=CHARLength=61      (Search Field)
Field: STREET   Type=CHARLength=25      (Search Field)
Field: CITY     Type=CHARLength=25      (Search Field)
Field: STATE    Type=CHARLength=2++ Primary Key Field ++
Field: ZIP      Type=CHARLength=9       (Search Field)

```

---

## B.2 Car Dealer database source files

This section contains the source files of the Car Dealer IVP database for your reference.

### B.2.1 AUTPSB11.psb

The source of the PSB, AUTPSB11, that we used in our scenarios is listed in Example B-2.

*Example B-2 AUTPSB11 PSB source*

---

```

AUTOLPCB PCB TYPE=DB,DBDNAME=AUTOLDB,PROCOPT=AP,KEYLEN=100
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
SENSEG NAME=STOCKSALE,PARENT=STOCK
SENSEG NAME=SALESPER,PARENT=DEALER
SENSEG NAME=SALESINF,PARENT=SALESPER
SENSEG NAME=EMPLINFO,PARENT=SALESPER
AUTS1PCB PCB TYPE=DB,DBDNAME=AUTOLDB,PROCOPT=GRP,KEYLEN=100,      X
PROCSEQ=INDEX11
SENSEG NAME=ORDER,PARENT=0
SENSEG NAME=MODEL,PARENT=ORDER
SENSEG NAME=DEALER,PARENT=MODEL
SENSEG NAME=STOCK,PARENT=MODEL
*
AUTS2PCB PCB TYPE=DB,DBDNAME=AUTOLDB,PROCOPT=GRP,KEYLEN=64,      X
PROCSEQ=INDEX22
SENSEG NAME=DEALER,PARENT=0
SENSEG NAME=MODEL,PARENT=DEALER
SENSEG NAME=STOCK,PARENT=MODEL
*
AUSI2PCB PCB TYPE=DB,DBDNAME=INDEX22,PROCOPT=GRDP,KEYLEN=28
SENSEG NAME=SINDEXB,PARENT=0
EMPLPCB PCB TYPE=DB,DBDNAME=EMPLDB2,PROCOPT=AP,KEYLEN=10
SENSEG NAME=EMPL,PARENT=0
SENSEG NAME=DEALER,PARENT=EMPL
SENSEG NAME=SALESINF,PARENT=DEALER
SENFLD NAME=QUOTA,START=1
SENFLD NAME=SALESYTD,START=7
SENSEG NAME=EMPLINFO,PARENT=EMPL
PSBGEN PSBNAME=AUTPSB11,LANG=JAVA

```

END

---

## B.2.2 AUTODB.dbd

The source of the physical DBD, AUTODB, that is needed for use with PSB AUTPSB11, is listed in Example B-3.

*Example B-3 AUTODB DBD source*

---

DBD	NAME=AUTODB,ACCESS=(HDAM,OSAM),	X
	RMNAME=(DFSHDC40,1,5,200)	
	DATASET DD1=DFSCLR	
	SEGM NAME=DEALER,PARENT=0,BYTES=61	
	FIELD NAME=(DLRNO,SEQ,U),BYTES=4,START=1,TYPE=C	
	FIELD NAME=DLRNAME,BYTES=30,START=5,TYPE=C	
	FIELD NAME=CITY,BYTES=10,START=35,TYPE=C	
	FIELD NAME=ZIP,BYTES=10,START=45,TYPE=C	
	FIELD NAME=PHONE,BYTES=7,START=55,TYPE=C	
	LCHILD NAME=(SINDXB,INDEX22),POINTER=INDX	
	XDFLD NAME=XFLD2,SEGMENT=MODEL,	X
	SRCH=(MAKE,MODEL),	X
	SUBSEQ=(YEAR,/SX1),	X
	DDATA=COUNT	
	SEGM NAME=MODEL,PARENT=DEALER,BYTES=37	
	FIELD NAME=(MODKEY,SEQ,U),BYTES=24,START=3,	X
	TYPE=C	
	FIELD NAME=MODTYPE,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	
	FIELD NAME=COUNT,BYTES=2,START=32,TYPE=P	
	FIELD NAME=/SX1	
	SEGM NAME=ORDER,PARENT=MODEL,BYTES=74	
	FIELD NAME=(ORDNBR,SEQ,U),BYTES=6,START=1,TYPE=C	
	FIELD NAME=LASTNME,BYTES=25,START=7,TYPE=C	
	FIELD NAME=FIRSTNME,BYTES=25,START=32,TYPE=C	
	FIELD NAME=DATE,BYTES=10,START=57,TYPE=C	
	FIELD NAME=TIME,BYTES=8,START=67,TYPE=C	
	LCHILD NAME=(SINDXA,INDEX11),POINTER=INDX	
	XDFLD NAME=XFLD1,SRCH=(LASTNME,FIRSTNME,ORDNBR),	X
	DDATA=DATE	
	SEGM NAME=SALES,PARENT=((MODEL,),(STOCK,PHYSICAL,AUTODB)),	X
	BYTES=85,	X
	POINTER=(LPARNT,LTWINBWD,TWINBWD),	X
	RULES=(VVV)	
	FIELD NAME=(SALENUM,SEQ,U),BYTES=4,START=49,TYPE=C	
	FIELD NAME=SALDATE,BYTES=8,START=53,TYPE=C	
	FIELD NAME=LASTNME,BYTES=25,START=61,TYPE=C	
	SEGM NAME=STOCK,PARENT=MODEL,BYTES=46	
	LCHILD NAME=(SALES,AUTODB),PAIR=STOCSALE,PTR=DBLE	
	FIELD NAME=(STKVIN,SEQ,U),BYTES=20,START=1,TYPE=C	
	FIELD NAME=COLOR,BYTES=10,START=21,TYPE=C	
	FIELD NAME=PRICE,BYTES=5,START=31,TYPE=C	
	FIELD NAME=LOT,BYTES=10,START=36,TYPE=C	

```

FIELD NAME=WRNTY,BYTES=1,START=46,TYPE=X
SEGM NAME=STOCSALE,PARENT=STOCK,PTR=PAIRED,
      SOURCE=( (SALES,DATA,AUTODB))
FIELD NAME=(SALENUM,SEQ,U),BYTES=4,START=29,TYPE=C
FIELD NAME=(DLRNO,SEQ,U),BYTES=4,START=1,TYPE=C
FIELD NAME=MODKEY,BYTES=24,START=5,TYPE=C
FIELD NAME=SALDATE,BYTES=8,START=33,TYPE=C
FIELD NAME=LASTNME,BYTES=25,START=41,TYPE=C
SEGM NAME=SALESPER,
      PARENT=( (DEALER,),(EMPL,PHYSICAL,EMPDB2)),
      BYTES=6,
      POINTER=(LPARNT,LTWINBWD,TWINBWD),
      RULES=(VVV)
FIELD NAME=(EMPNO,SEQ,U),BYTES=6,START=1,TYPE=C
SEGM NAME=SALESINF,PARENT=SALESPER,BYTES=15
FIELD NAME=QUOTA,BYTES=5,START=1,TYPE=C
FIELD NAME=SALESYTD,BYTES=5,START=6,TYPE=C
FIELD NAME=COMSSION,BYTES=5,START=11,TYPE=C
DBDGEN

```

---

### B.2.3 EMPDB2.dbd

The source of the physical DBD, EMPDB2, that is needed for PSB AUTPSB11, is listed in Example B-4.

*Example B-4 EMPDB2 DBD source*

---

```

DBD NAME=EMPDB2,ACCESS=(HDAM,OSAM),
      RMNAME=(DFSHDC40,1,5,200)
DATASET DD1=DFSEMP
SEGM NAME=EMPL,PARENT=0,BYTES=56
LCHILD NAME=(SALESPER,AUTODB),PAIR=EMPSAL,POINTER=DBLE
FIELD NAME=(EMPNO,SEQ,U),BYTES=6,START=1,TYPE=C
FIELD NAME=LASTNME,BYTES=25,START=7,TYPE=C
FIELD NAME=FIRSTNME,BYTES=25,START=32,TYPE=C
SEGM NAME=EMPSAL,PARENT=EMPL,PTR=PAIRED,
      SOURCE=( (SALESPER,DATA,AUTODB))
FIELD NAME=(DLRNO,SEQ,U),BYTES=4,START=1,TYPE=C
SEGM NAME=EMPLINFO,PARENT=EMPL,BYTES=61
FIELD NAME=(STATE,SEQ,M),BYTES=2,START=51,TYPE=C
FIELD NAME=ADDRESS,BYTES=61,START=1,TYPE=C
FIELD NAME=STREET,BYTES=25,START=1,TYPE=C
FIELD NAME=CITY,BYTES=25,START=26,TYPE=C
FIELD NAME=ZIP,BYTES=9,START=53,TYPE=C
DBDGEN
FINISH
END

```

---

### B.2.4 SINDEXT11.dbd

Example B-5 lists the source of the secondary index DBD, SINDEXT11, that is needed for both DBD AUTODB and PSB AUTPSB11. It is sourced and targeted on the segment ORDER.

*Example B-5 SINDEXT11 DBD source*


---

```

DBD    NAME=SINDEXT11,ACCESS=(INDEX,VSAM)
        DATASET DD1=SINDX1P
        SEGM    NAME=SINDXA,PARENT=0,BYTES=66
        FIELD   NAME=(XFLDA,SEQ,U),BYTES=56,START=1,TYPE=C
        FIELD   NAME=DATE,BYTES=10,START=57,TYPE=C
        LCHILD  NAME=(ORDER,AUTODB),INDEX=XFLD1
        DBDGEN
        FINISH
        END

```

---

**B.2.5 SINDEXT22.dbd**

Example B-6 lists the source of the secondary index DBD, SINDEXT22, that is needed for both DBD AUTODB and PSB AUTPSB11. It is sourced on segment MODEL and targeted on the root segment DEALER.

*Example B-6 SINDEXT22 DBD source*


---

```

DBD    NAME=SINDEXT22,ACCESS=(INDEX,VSAM)
        DATASET DD1=SINDX2P
        SEGM    NAME=SINDXB,PARENT=0,BYTES=34
        FIELD   NAME=(XFLDB,SEQ,U),BYTES=28,START=1,TYPE=C
        FIELD   NAME=COUNT,BYTES=2,START=25,TYPE=C
        FIELD   NAME=ENQUIRS,BYTES=4,START=25,TYPE=P
        LCHILD  NAME=(DEALER,AUTODB),INDEX=XFLD2
        DBDGEN
        FINISH
        END

```

---

**B.2.6 AUTOLDB.dbd**

Example B-7 lists the source of the logical DBD, AUTOLDB, that is needed for PSB AUTPSB11. It is based on the the physical DBD AUTODB and uses its two internal logical relationships and its logical relationship with the physical DBD EMPDB2.

*Example B-7 AUTOLDB DBD source*


---

```

DBD    NAME=AUTOLDB,ACCESS=LOGICAL
        DATASET LOGICAL
        SEGM NAME=DEALER,PARENT=0,SOURCE=((DEALER,,AUTODB))
        SEGM NAME=MODEL,PARENT=DEALER,SOURCE=((MODEL,,AUTODB))
        SEGM NAME=ORDER,PARENT=MODEL,SOURCE=((ORDER,,AUTODB))
        SEGM NAME=SALES,PARENT=MODEL,                                     X
            SOURCE=((SALES,DATA,AUTODB),(STOCK,DATA,AUTODB))
        SEGM NAME=STOCK,PARENT=MODEL,SOURCE=((STOCK,,AUTODB))
        SEGM NAME=STOCSALE,PARENT=STOCK,                                 X
            SOURCE=((STOCSALE,DATA,AUTODB),(MODEL,KEY,AUTODB))
        SEGM NAME=SALESPER,PARENT=DEALER,                               X
            SOURCE=((SALESPER,DATA,AUTODB),(EMPL,DATA,EMPDB2))
        SEGM NAME=SALESINF,PARENT=SALESPER,                             X
            SOURCE=((SALESINF,,AUTODB))
        SEGM NAME=EMPLINFO,PARENT=SALESPER,                             X
            SOURCE=((EMPLINFO,,EMPDB2))

```

---



```

DBDGEN
FINISH
END

```

---

## B.2.7 EMPLDB2.dbd

Example B-8 lists the source of the logical DBD, EMPLDB, that is needed for PSB AUTPSB11. It is based on the the physical DBD EMPDB2 and uses its logical relationship with the physical DBD AUTODB.

*Example B-8 EMPLDB DBD source*

---

```

DBD      NAME=EMPLDB2,ACCESS=LOGICAL
        DATASET LOGICAL
        SEGM  NAME=EMPL,PARENT=0,SOURCE=((EMPL,,EMPDB2))
        SEGM  NAME=DEALER,PARENT=EMPL,                                X
              SOURCE=((EMPSAL,KEY,EMPDB2),(DEALER,DATA,AUTODB))
        SEGM  NAME=SALESINF,PARENT=DEALER,                            X
              SOURCE=((SALESINF,,AUTODB))
        SEGM  NAME=EMPLINFO,PARENT=EMPL,                              X
              SOURCE=((EMPLINFO,,EMPDB2))
        DBDGEN
        FINISH
        END

```

---





C

## The environment for our scenarios

This appendix gives information about the configuration that we used and the versions of the programs used for the Open Database scenarios, which are available as source as described in Appendix D, “Additional material” on page 249.

Topics in this appendix include:

- ▶ Used system configuration
- ▶ Used application versions
- ▶ Required APAR numbers

## C.1 Used system configuration

Our IMS system is an IMS Version 11 and is located on an LPAR called WTSC63. The DB2 for System z which is used in one of the scenarios is also located on the LPAR WTSC63.

The IMS has been enhanced with an ODBM and a IMS Connect address spaces as shown in Figure C-1. We have omitted all the other address spaces in the picture for clarity. Notice that the SCI connection from IMS Connect to ODBM is by program call (PC) if they are within the same LPAR, or through XCF if they are on a different LPAR. This is the normal behavior of SCI. If XA is used in a managed environment the application server will keep track of the transactionality. On the z/OS side, this would involve RRS, which was therefore enabled in all used components.

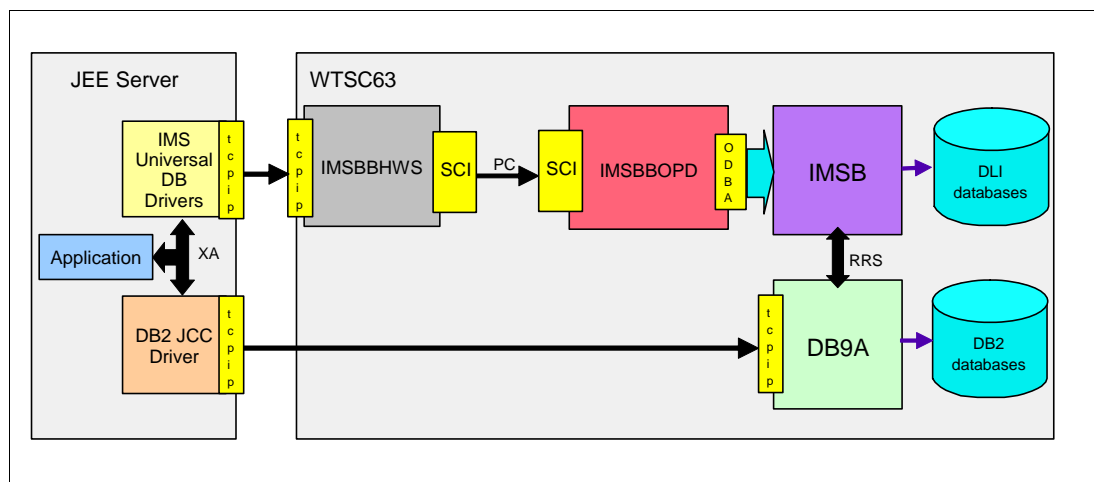


Figure C-1 The system configuration used for this book

For more informations about the used configuration see Chapter 3, “System environment” on page 41.

## C.2 Used application versions

Table C-1 lists the versions of the products used in this book.

Table C-1 Products and versions

Product	Version
IBM Cognos® Virtual View Manager	V 8.4.1
IBM Data Studio	V 2.2.0.1 (2.2.0.20091124_1634)
IBM Rational Application Developer for WebSphere Software	V 7.5.5 (7.5.5.20091203_0703)
IBM Rational Developer for System z	V 7.6.0.1 (7.6.0.20091216_1622)
WebSphere Application Server Version 7.0	V 7.0.0.7 (2.0.1.20091203_0240)
IMS Enterprise Suite DLIModel Utility Plug-in	V 2.0.3.20091102_1145 with DLIModel Utility Fix V (2.0.3.20100209_1546)
IBM Installation Manager	V 1.3.4

Product	Version
IBM Mashup Center	V2.0
IMS Universal Drivers	see required APARs
Java Development Kit (JDK)	IBM J9 VM (build 2.4, JRE 1.6.0 IBM J9 2.4 Windows XP x86-32 jvmwi3260sr6-20091001_43491
DB2 JCC Driver	V 3.57 / V 4.7

### C.3 Required APAR numbers

Table C-2 lists the APARs that we applied in our environment in order to have working samples.

**Author Comment:** To be completed as necessary before publishing

Table C-2 Products and APARs

Product/function	APARs	PTF
IMS 11 IVP problem (missing DFSODSCR in the GA tape)	PM01374	OPEN
IMS V11 Generic JDBC support (for Cognos and Data Source Explorer integration.)	PM12893	OPEN
IMS Problem Investigator. New function to track and format CEX event records for ODBA transactions.	PM09535	UK56453
IMS Connect Extensions. Improved use of OMEGAMON to capture the event buffers.	PM04643	UK55059





## Additional material

This book refers to additional material that can be downloaded from the Internet as described below.

### Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

<ftp://www.redbooks.ibm.com/redbooks/SG247856>

Alternatively, you can go to the IBM Redbooks Web site at:

[ibm.com/redbooks](http://ibm.com/redbooks)

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247856.

### Using the Web material

The additional Web material that accompanies this book includes the following files:

<i><b>File name</b></i>	<i><b>Description</b></i>
<b>AUTPSB11.jar</b>	It contains the necessary AUTPSB11DatabaseView Java class in the package samples.ims.openDb generated by the IMS Enterprise Suite DLIModel Utility, as well as all other generated artefacts like the XML schemas.
<b>AUTPSB11Modified.jar</b>	It contains the same as the AUTPSB11.jar, but the DatabaseView is modified to match the PSB and DBD special Datatypes. To use this class you have to delete the contents of the IVP CarDealer Database and insert data with the corresponding representation (Packeddecimal and Hexadecimal)

**CarDealer\_DBDandPSBSources.zip**

It is a zip file and contains the DBDs and the PSB of the Car Dealer IVP Database which are used for this example.

**IMSandDB2.ear**

It is the Enterprise Archive file for the deployment to a WebSphere Application Server Version 7. When you import this file to your Eclipse, you have to add the required libraries to your Java Build path (imsudbJXA.rar contents, JDK, WAS 7 runtime and AUTPSB11.jar). For installing it in WebSphere Application Server you have to install the IMS Universal JDBC XA resource adapter and the IBM DB2 Data Server Driver as described in the scenario.

**IMSOpenDBTransaction.zip**

It is the project interchange file for the IMS Transaction example. When you import this file to your Eclipse, you have to add the required libraries to your Java Build path (imsudb.jar, imsutm.jar, JDK and AUTPSB11.jar)

**IMSOpenDBApp.zip**

It is the project interchange file for the IMS Standalone examples as well as some more self explaining Standalone examples. When you import this file to your Eclipse, you have to add the required libraries to your Java Build path (imsudb.jar, JDK and AUTPSB11.jar)

**IMSOpenDBSource.zip**

It is the project interchange file for the IMS Managed environment examples as well as some used code fragments in the book. When you import this file to your Eclipse, you have to add the required libraries to your Java Build path (imsudbJXA.rar contents, JDK, WAS 7 runtime and AUTPSB11.jar)

**dlitest1**

It is an example of using the IMS Universal DLI driver to retrieve, update and delete rows in the IMS sample DB AUTODB database and the PCB that references it in the PSB AUTPSB11.

**dlitest2**

It is an example of using the batch method feature of the IMS Universal DLI driver to retrieve, update and delete rows in the IMS sample DB AUTODB database and the PCB that references it in the PSB AUTPSB11. Over 8000 segments have been added to the sample data supplied with the IMS product for this test.

## System requirements for downloading the Web material

The following system configuration is recommended:

<b>Hard disk space:</b>	2 MB minimum
<b>Operating System:</b>	Windows XP or alternatively Linux
<b>Processor:</b>	x86 architecture
<b>Memory:</b>	256 MB minimum

## How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder if you want to view the code.



**Tip:** Jar files are also packed archives and can be extracted by renaming the file extension to .zip.

The Project Interchange™ files can be imported in eclipse by using the File->Import and select Project Interchange file. The necessary build path has to be configured that the reference errors disappear.

Therefore you need additionally the IMS Universal DB Drivers and other resources like the WebSphere Application Server Runtime and Java Development Kit.



# Abbreviations and acronyms

<b>ACEE</b>	access control environment element	<b>DRDA</b>	Distributed Relational Database Architecture
<b>AGN</b>	Application Group Name	<b>DSN</b>	Data Source Name
<b>AIB</b>	Application Interface Block	<b>DTP</b>	distributed transaction processing
<b>AOI</b>	Automated Operator Interface	<b>EAB</b>	Enterprise Access Builder
<b>APF</b>	authorized program facility	<b>EAI</b>	enterprise application integration
<b>API</b>	application programming interface	<b>ECB</b>	Event Control Block
<b>APPC</b>	Advanced Program-to-Program Communication	<b>EIS</b>	enterprise information systems
<b>APSB</b>	Allocate PSB	<b>EISS</b>	Enterprise Information Systems
<b>ARM</b>	Automatic Restart Manager	<b>EISs</b>	Enterprise Information Systems
<b>ASN</b>	Abend Search and Notification	<b>EJB</b>	Enterprise JavaBean
<b>BMP</b>	batch messaging program	<b>EJBs</b>	Enterprise JavaBeans
<b>BPE</b>	Base Primitive Environment	<b>EMF</b>	Eclipse Modeling Framework
<b>CBM</b>	Component Business Modeling	<b>ETO</b>	Extended Terminal Option
<b>CCF</b>	Common Connector Framework	<b>EWLM</b>	Enterprise Workload Manager™
<b>CCI</b>	Common Client Interface	<b>GEF</b>	Graphical Editor Framework
<b>CDE</b>	contents directory entry	<b>GN</b>	Get Next
<b>CGI</b>	Common Gateway Interface	<b>GNP</b>	Get Next within Parent
<b>CI</b>	Control Interval	<b>GU</b>	Get Unique
<b>CICS</b>	Customer Information Control System	<b>GUI</b>	graphical user interface
<b>CLI</b>	command line interface	<b>HALDB</b>	High Availability Large Data Bases
<b>CLP</b>	command line processor	<b>HTML</b>	Hypertext Markup Language
<b>CLS</b>	Common Service Layer	<b>HTTP</b>	Hypertext Transfer Protocol
<b>CSL</b>	Common Service Layer	<b>IBM</b>	International Business Machines Corporation
<b>CSM</b>	Complete Status Message	<b>IDE</b>	Integrated Development Environment
<b>CTG</b>	CICS Transaction Gateway	<b>IDS</b>	Indormix Dynamic Server
<b>DB2</b>	Database 2	<b>ILDS</b>	indirect list data set
<b>DBCTL</b>	Database Control	<b>IMS</b>	Information Management System
<b>DBD</b>	database description	<b>IPCS</b>	Interactive Problem Control System
<b>DBMS</b>	database management system	<b>IPL</b>	initial program load
<b>DBPCB</b>	database PCB	<b>IRLM</b>	internal resource lock manager
<b>DBRA</b>	Database Resource Adapter	<b>IRM</b>	IMS request message
<b>DBRC</b>	Database Recovery Control	<b>ISC</b>	intersystem communication
<b>DDM</b>	Distributed Data Management	<b>ISPF</b>	Interactive Systems Productivity Facility
<b>DEDB</b>	data entry database	<b>ISRT</b>	Insert
<b>DLET</b>	Delete	<b>ISVs</b>	independent software vendors
<b>DMB</b>	data management block	<b>ITSO</b>	International Technical Support Organization
<b>DRA</b>	Database Resource Adapter		

<b>IVP</b>	Installation Verification Procedure	<b>OTMA C/I</b>	OTMA Callable Interface
<b>IVPEX</b>	IVP Export Utility	<b>OTMA</b>	Open Transaction Manager Access
<b>J2C</b>	Java EE Connector Architecture	<b>PC</b>	program call
<b>J2EE</b>	Java 2 Platform, Enterprise Edition	<b>PCB</b>	program communication block
<b>JBP</b>	Java Batch Program	<b>PCBs</b>	Program Control Blocks
<b>JCA</b>	Java Connector Architecture	<b>PDU</b>	Partition Definition Utility
<b>JCC</b>	JAVA Common Client	<b>PPT</b>	program properties table
<b>JCL</b>	job control language	<b>PRA</b>	Parallel RECON Access
<b>JDBC</b>	Java Database Connectivity	<b>PSB</b>	program specification block
<b>JDK</b>	Java Development Kit	<b>PSBs</b>	program specification blocks
<b>JMP</b>	Java Message Program	<b>RACF</b>	Resource Access Control Facility
<b>JMS</b>	Java Message Server	<b>RAD</b>	Rational Application Developer
<b>JNDI</b>	Java Naming and Directory Interface	<b>RAR</b>	resource adapter archive
<b>JRE</b>	Java Runtime Environment	<b>RAS</b>	Resource Access Security
<b>JSP</b>	JavaServer Pages	<b>RDDS</b>	Resource Definition Data Set
<b>JTA</b>	Java Transaction API	<b>RECON</b>	Recovery Control
<b>JTS</b>	Java Transaction Service	<b>REPL</b>	Replace
<b>JVM</b>	Java Virtual Machine	<b>RM</b>	resource manager
<b>Java EE</b>	Java Platform Enterprise Edition	<b>RMM</b>	Request MOD Message
<b>KBLA</b>	Knowledge-Based Log Analysis	<b>RRS</b>	Resource Recovery Services
<b>LAN</b>	local area network	<b>RRS/MVS</b>	Resource Recovery Services/MVS
<b>LDAP</b>	Lightweight Directory Access Protocol	<b>RSM</b>	request status message
<b>LPAR</b>	logical partition	<b>RYO</b>	Roll-Your-Own
<b>LSQA</b>	local system queue area	<b>SAF</b>	System Authorization Facility
<b>LTERM</b>	logical terminal	<b>SCI</b>	Structured Call Interface
<b>LU</b>	logical unit	<b>SDK</b>	Software Development Kit
<b>LU2</b>	logical unit 2	<b>SGML</b>	Standard Generalized Markup Language
<b>MCI</b>	Message Control Information	<b>SLSB</b>	stateless session bean
<b>MDB</b>	message-driven bean	<b>SMP/E</b>	System Modification Program/Extended
<b>MFS</b>	Message Format Service	<b>SMU</b>	Security Maintenance Utility
<b>MOD</b>	message output descriptor	<b>SNA</b>	Systems Network Architecture
<b>MPP</b>	message processing program	<b>SOA</b>	service-oriented architecture
<b>MSC</b>	Multiple Systems Coupling	<b>SOMA</b>	Service Oriented Modeling and Architecture
<b>MVS</b>	Multiple Virtual System	<b>SPE</b>	small programming enhancement
<b>ODBA</b>	Open Database Access	<b>SPOC</b>	single point of control
<b>ODBC</b>	Open DataBase Connectivity	<b>SQLJ</b>	SQL for Java
<b>ODBM</b>	Open Database Manager	<b>SSAs</b>	segment search arguments
<b>OLDS</b>	online log data set	<b>SSL</b>	Secure Socket Layer
<b>OLDSSs</b>	online log data sets	<b>SSPM</b>	Sysplex serialized program management
<b>OLR</b>	Online Reorganization	<b>STE</b>	storage tracking element
<b>OM</b>	Operations Manager	<b>STSN</b>	set and test sequence numbers
<b>OO</b>	object-oriented		

<b>SVL</b>	Silicon Valley Laboratories
<b>TCO</b>	Time-Controlled Operations
<b>TCP/IP</b>	Transmission Control Protocol/Internet Protocol
<b>THREAD</b>	TYPE
<b>TMRA</b>	Transaction Manager Resource Adapter
<b>TPIPE</b>	transaction pipe
<b>TRACE</b>	TYPE
<b>Thilo</b>	through tooling
<b>UOR</b>	Unit of Recovery
<b>VIPA</b>	Virtual IP Addressing
<b>VVM</b>	Virtual View Manager
<b>W3C</b>	World Wide Web Consortium
<b>WAN</b>	wide area network
<b>WAS</b>	WebSphere Application Server
<b>WID</b>	WebSphere Integration Developer
<b>WLM</b>	workload manager
<b>WPS</b>	WebSphere Process Server
<b>WSDL</b>	Web Service Description Language
<b>WWW</b>	World Wide Web
<b>XCF</b>	cross-system coupling facility
<b>XMI</b>	XML Metadata Interchange
<b>XML</b>	Extensible Markup Language
<b>db2cli</b>	DB2 Interactive Call Level Interface



# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

## IBM Redbooks

For information about ordering these publications, see “How to get Redbooks” on page 258. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *IMS Version 11 Technical Overview*, SG24-7807
- ▶ *DB2 9 for z/OS: Distributed Functions*, SG24-6952-01
- ▶ *Powering SOA Solutions with IMS*, SG24-7662
- ▶ *Powering SOA with IBM Data Servers*, SG24-7259
- ▶ *IMS Performance and Tuning Guide*, SG24-7324
- ▶ *IMS V10 Implementation Guide: A Technical Overview*, SG24-7526
- ▶ *IMS Connectivity in an On Demand Environment: A Practical Guide to IMS Connectivity*, SG24-6794
- ▶ *Publishing IMS and DB2 Data Using WebSphere Information Integrator: Configuration and Monitoring Guide*, SG24-7132
- ▶ *IMS V9 Implementation Guide: A Technical Overview*, SG24-6398
- ▶ *IMS V8 Implementation Guide: A Technical Introduction of the New Features*, SG24-6594
- ▶ *Reorganizing Databases Using IMS Tools: A Detailed Look at the IBM IMS High Performance Tools*, SG24-6074
- ▶ *IMS Installation and Maintenance Processes*, SG24-6574
- ▶ *Using IMS Data Management Tools for Fast Path Databases*, SG24-6866
- ▶ *IMS in the Parallel Sysplex Volume I: Reviewing the IMSplex Technology*, SG24-6908
- ▶ *IMS in the Parallel Sysplex Volume II: Planning the IMSplex*, SG24-6928

## Other publications

These publications are also relevant as further information sources:

- ▶ *IMS Version 11 Application Programming*, SC19-2428
- ▶ *IMS Version 11 Application Programming APIs*, SC19-2429
- ▶ *IMS Version 11 Commands, Volume 1: IMS Commands*, SC19-2430
- ▶ *IMS Version 11 Commands, Volume 2: IMS Commands N-V*, SC19-2431
- ▶ *IMS Version 11 Commands, Volume 3: IMS Component and z/OS Commands*, SC19-2432
- ▶ *IMS Version 11 Communications and Connections*, SC19-2433
- ▶ *IMS Version 11 Database Administration*, SC19-2434

- ▶ *IMS Version 11 Database Utilities*, SC19-2435
- ▶ *IMS Version 11 Diagnosis*, GC19-2436
- ▶ *IMS Version 11 Exit Routines*, SC19-2437
- ▶ *IMS Version 11 Installation*, GC19-2438
- ▶ *IMS Version 11 Master Index and Glossary*, SC19-2440
- ▶ *IMS Messages and Codes, Volume 1: DFS Messages*, GC18-9712
- ▶ *IMS Messages and Codes, Volume 2: Non-DFS Messages*, GC18-9713
- ▶ *IMS Messages and Codes, Volume 3: IMS Abend Codes*, GC18-9714
- ▶ *IMS Messages and Codes, Volume 4: IMS Component Codes*, GC18-9715
- ▶ *IMS Version 11 Operations and Automation*, SC19-2441
- ▶ *IMS Version 11 Release Planning*, GC19-2442
- ▶ *IMS Version 11 System Administration*, SC19-2443
- ▶ *IMS Version 11 System Definition*, GC19-2444
- ▶ *IMS Version 11 System Programming APIs*, SC19-2445
- ▶ *IMS Version 11 System Utilities*, SC19-2446
- ▶ *IMS and SOA Executive Overview*, GC19-2516
- ▶ *IMS TM Resource Adapter User's Guide and Reference*, SC19-1211
- ▶ *Program Directory for Information Management System Transaction and Database Servers V11.0*, GI10-8788
- ▶ *IRLM Messages and Codes*, GC19-2666

## Online resources

These Web sites are also relevant as further information sources:

- ▶ IMS products and tools  
<http://www.ibm.com/ims>
- ▶ IMS Information Center  
<http://publib.boulder.ibm.com/infocenter/imzic>
- ▶ The IMS Enterprise Suite set of topics  
[http://publib.boulder.ibm.com/infocenter/imzic/topic/com.ibm.ims.es.doc/ies\\_home.htm](http://publib.boulder.ibm.com/infocenter/imzic/topic/com.ibm.ims.es.doc/ies_home.htm)
- ▶ System z hardware  
<http://www.ibm.com/systems/z/hardware/>
- ▶ IMS tools :  
<http://www.ibm.com/software/data/db2imstools/products/ims-tools.html>

## How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this Web site:

[ibm.com/redbooks](http://ibm.com/redbooks)



## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)



# Index

## A

ACBLIB 97  
 ACEE 28  
 Address space 209, 211  
 address space xvii, 4, 11, 14, 18, 20–21, 42–44, 154, 161, 209, 211  
 ADO 230  
 AERTDLI 34  
 aggregate functions 120  
 AIB 53, 182  
 API 2, 18, 27, 29, 47, 53, 75, 96, 139, 174, 184, 200, 211, 228–229  
 APIs 5, 11, 39, 103, 113, 174, 225, 230  
 application program 27, 32–33, 50, 52, 72, 96, 112, 114, 233  
 APSB security 27, 72–73  
 ARMRST 45–46  
 AT-TLS 32

## B

Base Primitive Environment  
     *See* BPE  
 BPE 32, 43, 214  
     exit list 69, 72  
 BPECFG 43–44, 46  
 business process 2  
 business value 6

## C

CLI 228  
 client application 7, 14–15, 25, 27, 29, 61, 71, 99  
 COBOL 4–6, 9, 15, 76, 90–91, 120, 181  
 COBOL copybook 4, 94  
     additional field information 9  
 command line  
     interface 230  
     processor 230  
 Composite Information Server 138  
 configuration 25, 41, 76, 97, 141, 147, 168, 180, 209, 231  
 configuration member 32–33, 43, 171, 209  
 Connection Factory 105–106, 203  
 connection pooling 13, 37, 99, 104  
 ConnectionFactory 105–106, 200  
 Control Center 61, 231  
 Coupling Facility 21  
 CSL 14, 22–23, 25, 42, 208, 214  
     address space 23, 25, 43  
 CSLDCxxx 27, 50, 59, 180, 209  
 CSLDIxxx 49, 59  
 CSLOIxxx 47  
 CSLSIxxx 45–46  
 CSSLIB 53, 62  
 cursor 70, 181–182

## D

data sharing 180  
 data source 99, 138, 171, 229  
 data store 32, 110, 148  
 data transformation 113, 122  
 database xvii, 2–3, 5, 9, 18, 42, 46, 48, 50, 75–76, 96–97, 126, 134, 139, 154–155, 159–160, 180–181, 208, 228, 230–231, 235  
 DataPower 9  
 DataSource 105, 108, 137, 164, 232–233  
 datastore 33, 51, 171, 180  
 datastore name 52, 180  
 DATASTORE statement 61  
 DB/DC 67, 209  
 DB2 xvii, 5–6, 14, 18–19, 36, 38, 50, 74, 99, 101, 126, 137, 160, 227  
 DB2 for z/OS server 232  
 DB2Binder utility 232  
 DBCTL 3, 25, 29, 34, 209, 211  
 DCCTL 3  
 DEDB 52  
 DELETE 114, 116, 118, 157–158, 166, 191  
 Demand Environment 60–61, 211  
 dependent region 5, 28, 174–175  
 DESC 47, 114  
 destination name 86  
 DFSDDLTO 91  
 DFSERA10 214  
 DFSPBxxx 27, 72–73  
 DFSRAS00 28, 73  
 DL/I xvii, 2, 13, 18, 25, 27, 29, 96, 101, 179–181, 210–211, 214, 227  
 DL/I call 181–182, 214  
 DLIModel 5–6, 8, 39–40, 42, 75–76, 87, 114–115, 129, 148  
 DLIModel Utility 9, 11, 75, 81, 84, 88, 146  
     business goal 9  
 DLIModel utility 8–9, 39, 42, 75–76, 146, 180  
 DRA 3, 5, 19, 26, 34, 36, 38, 50, 66, 100–101, 180, 211  
 DSN 46, 48, 53, 140–141  
 dynamic SQL 109, 228

## E

ECSA 10  
 EJB 20, 36, 104–105, 162, 208  
 Enterprise Edition (EE) 4  
 enterprise information system (EIS) 8  
 exit 10, 27–28, 30–31, 47, 53, 86, 193  
     calls 32  
     routine 28, 30, 32, 53, 72  
     user 53  
 exit routine 28, 53, 71–72

**F**

Fast Path 10, 52, 172  
 feed 149  
 FMID 96  
 FOR 43, 211, 232  
 full-function product 231

**G**

GSAM 182  
 GSAM database 182

**H**

HALDB 64  
 HDAM 240  
 HOSTNAME 60–61  
 HTML 4, 166–167  
 HTTP 6–7  
 HWS 45  
 HWSCFGxx 29, 61–62, 71  
 HWSJAVA0 32  
 HWSRCORD 62  
 HWSSMPL0 32  
 HWSSMPL1 32

**I**

IBM Data Server Client 228, 230–231  
 IBM IMS 14, 94  
 IMS xvii–xviii, 1–2, 7, 18, 41, 75, 95–96, 125–126, 153–154, 179, 208, 211, 227–228, 235, 246  
 IMS application 3, 5–9, 90, 97, 181  
     development 9  
 IMS Application Menu 64, 67  
 IMS asset 2  
 IMS command 47  
 IMS Connect xvii, 2, 4–5, 18, 22–24, 41–42, 75, 96, 99, 144, 148, 154, 180, 209, 211, 246  
 IMS Connect BPE 60  
 IMS Connect Extensions 5  
 IMS Connect security 30, 71  
 IMS Connectivity xviii, 60–61, 211  
 IMS Connector for Java 7  
 IMS Control Center 61  
 IMS data xvii, 2, 6, 10–11, 34, 36, 40–42, 50, 52, 76, 96, 125, 137, 146, 180, 199  
 IMS database 5, 9, 11, 18, 34, 36, 46, 48, 50, 75–76, 97, 99, 102, 147, 160, 169, 171, 180–181, 189, 208, 210  
     store XML data 9  
 IMS database resource  
     adapter 36  
     distribution 11  
 IMS DB 5, 18, 67, 113, 141–142, 160, 168, 211  
 IMS Enterprise Suite xvii, 2, 14, 39, 42, 75, 83, 97, 114–115, 126, 129, 146, 154, 160, 180  
 IMS host  
     information 34  
 IMS Java 14–15, 19, 38, 76, 79, 98, 101, 120, 174–175, 237  
 IMS Open Database 5, 11, 13, 18, 24, 29, 41–42, 81, 96,

102, 154, 161, 208, 211  
     Access 36  
     support 13, 29  
 IMS security 71  
 IMS service 3, 7  
 IMS SOA 2, 6–7, 80  
     Integration 6  
     Integration Suite 6–7  
 IMS SOAP Gateway 6–7  
     server 6–7  
     solution 7  
     Web Services 6  
 IMS SOAP gateway 4  
     Web services 7  
 IMS subsystem 11, 35, 37–38, 100  
 IMS system 3, 8, 28, 32, 45, 63–64, 75, 246  
 IMS TM 3–4, 7–8, 18, 28, 32  
 IMS TM Resource Adapter 4–8  
 IMS transaction 3–4, 6, 8, 18, 208  
 IMS Transaction Manager 10, 28, 61  
 IMS Universal DL/I driver 25, 29, 36, 38, 101, 103, 181, 184–186  
 IMS Universal Drivers 35, 38, 42, 96, 126, 129–130, 171, 177, 199, 208, 210, 212  
 IMS Universal drivers 5, 12, 29, 96, 180, 212  
 IMS Universal JDBC driver 25, 29, 36, 38, 96, 101, 174, 212  
 IMS Version  
     10 xviii  
     11 xviii, 11–12, 46, 96, 103, 126, 137, 146, 174, 209  
     9 5  
 IMSConnectionSpec 112, 180  
 IMSID 52, 171  
 IMSPLEX 43, 45, 47, 209, 211  
 IMSplex 11, 18, 25, 42–43, 45, 101, 209–210, 246  
 IMSPLEX statement 61  
 Information 19, 105, 127, 161, 200, 228  
 Informix 139, 230  
 InfoSphere MashupHub 149  
 input message 16, 61, 71, 175  
 input/output message  
     definition 4  
 installing xvii, 94, 126, 161, 230  
 integrated IMS 180  
 Internet 8, 12, 23, 249  
 IP address 37–38, 99, 154, 161, 171–172, 180  
 IPCS 64  
 ISIS 27–28, 73  
 ISPF 68  
 IVP 86–87, 97, 105–106, 154, 160, 169, 235

**J**

J2C 8, 147–148, 170  
 J2EE 4–5, 7, 19, 29, 35, 98, 104, 213, 229  
 J2EE application 104  
 J2EE Connector architecture 104  
 J2EE Connector Architecture (J2C) 8  
 Java 2, 4–6, 19, 36, 41–42, 75–76, 87, 89, 95–96, 126, 139, 141, 153–154, 179, 208, 212, 214, 227–229, 237  
 Java 2 Platform, Enterprise Edition 229

Java application 8–9, 15, 39–40, 42, 76, 98, 154, 157, 185, 212, 228, 231  
     static SQL 228  
 Java applications 5, 8, 12, 36, 102–103, 227–229, 231  
 Java classes 8, 105  
 JAVA Common Client 160, 228  
 Java development 15, 42, 154  
 Java Virtual Machine 5, 37, 99  
 java.io 166  
 JBP 5, 38, 174  
 JCA 4–5, 12, 18, 23, 36, 96, 98, 102, 147  
 JCC 160, 171, 228  
 JCL 43, 46, 48, 52, 59, 195  
 JDBC xvii, 5, 11, 13, 18–19, 35, 78, 96, 101, 125, 129, 153–154, 185, 210, 212, 227–228, 231–232  
 JDBC driver xvii, 15, 25, 29, 36, 38, 96, 101–102, 147, 174, 186, 212  
 JMP 5, 38, 174, 177  
 JVM 5, 37, 85, 99, 212

## L

LANG 43–44, 239  
 Linux 8, 14, 80, 126, 227–228, 230–231, 250  
 local DB2  
     subsystem 232  
 logical relationships 79, 242  
 LPAR 11, 18, 20, 45, 49, 63, 96, 100–101, 209, 211, 232, 246  
 LUW Version  
     9.5 FixPack 3 228

## M

mashup 146  
 Message Format Service (MFS) 6  
 metadata 5, 8–9, 15, 18, 39, 75, 92–94, 96–97, 125, 139, 146, 154, 160, 169, 171, 180, 185, 212–213  
 MFS 4, 66  
 middleware 7  
 migration 23  
 mode 20, 64, 86, 97, 227  
 MPP 5  
 MSDB 79, 90  
 MVS 47, 210

## O

ODBA 5, 17–23, 49–50, 74, 100–101, 210–211  
 ODBA interface 25, 27, 34, 211  
 ODBC 139, 228  
     client 141  
     driver 140, 230  
 ODBM 11, 14, 23–25, 42, 99, 101, 161, 171, 180, 208, 211, 246  
 OLDS 214  
 OM 23, 25, 28, 42  
 OMEGAMON 5  
 on demand 2  
 Open Database xvii, 2, 5, 17–18, 41, 75, 81, 95–96, 125, 154, 208, 211

    enhancement 18  
     Manager 11, 14, 25, 28, 42, 48, 154, 161  
 Open Database Manager 14, 25, 37, 41–42, 48, 99, 161  
 Open Transaction Manager Access 28, 61  
 Operations 25, 41–42, 46  
   operations 39, 46, 112, 118, 182, 195  
 Operations Manager 28, 42  
 OTMA 28, 45, 61  
 output message 4, 16, 175

## P

PCB 9, 39, 76, 78, 97–98, 112, 181, 237  
 performance xvii, 2, 10, 15, 28, 32, 109, 119, 138–139, 145, 180, 208–209, 228, 233  
 PL/I 4, 6, 15, 76, 78  
 PM01374 247  
 PM04643 247  
 PM09535 247  
 PM12893 247  
 port 34, 38, 61, 71, 99, 110, 148, 161, 168, 171, 173, 180, 233  
 Practical Guide 60–61, 211  
 PROCLIB 27, 30, 43, 45–46, 59, 177, 209  
 PROCOPT 15, 78–79, 239  
 Program Call (PC) 211  
 program communication  
     block 112  
 program specification block (PSB) 9  
 programming model 6–7, 14, 23, 102, 179, 212  
 project xviii–xix, 87, 90, 92, 135, 155–156, 163, 208  
 PSBGEN 239

## Q

QUERY 46, 53–54

## R

RACF 14, 27–28, 47, 60, 101  
 RAD 4, 154  
 RAR 147, 162, 169, 200  
 Rational Application Developer 4, 8, 15, 76, 79–80, 91, 126, 154, 160  
 Rational Developer 8, 15, 76, 80, 91, 126, 153, 213  
 Rational Developer for System z 79, 86, 94, 126  
 Rational Software  
     Architect 8  
 RDz 6  
 Recovery Resource Services 37, 99  
 Redbooks Web site 249, 258  
     Contact us xix  
 RESLIB 72, 211  
 Resource 3–4, 19, 23, 50, 61, 99, 147, 160, 179–180, 210–211  
 resource xvii, 4–5, 8, 11–12, 25, 27–28, 42, 50, 96, 98, 140, 146–147, 166, 174–175, 200, 213  
     access 28, 50, 71, 98  
     adapter 8, 12, 15, 29, 36, 50, 96, 98, 146–147, 174, 213  
     name 40, 148

Resource Adapter 4–6, 19, 23, 26, 103, 160, 179, 211  
 response message 61  
 RM 66  
 RRS 19, 26–27, 50, 99, 105, 161, 210

## S

SAF 71  
 SCEERUN 62  
 SCI 20, 22, 25, 28, 42, 209  
 SDFSRESL 46, 48, 53, 62, 211  
 security xvii, 8–9, 12–13, 27–28, 47, 61, 104, 144, 208  
 SECURITY macro 28  
 servlet 166  
 SETRACF 32, 71  
 SLDS 214  
 SMP/E 14, 75, 81, 96  
 SMU 73  
 SOA environment 6–7  
     IMS capabilities 6  
 SOAP 4, 6–7, 75, 139  
 SOAP Gateway 6–7  
 SOAP message 6  
 sockets 29  
 SPOC 47, 54–55  
 SQL xvii, 2, 5, 19, 36, 39, 96, 98, 102, 126, 135,  
 163–164, 179, 208, 210, 212, 214, 227–228  
 SQLJ 126, 160, 227, 231–232  
 storage 10, 43, 45, 186  
 stored procedure 74  
 Structured Call Interface xvii, 25, 41–42, 209  
 sync point 19, 26–27, 50  
 synchronous callout 15  
 Syntax Checker 64, 67  
 Sysplex 3, 209  
 sysplex 208  
 system definition 28  
 System z 8, 14, 18, 76, 79–80, 126, 153, 160, 172, 195,  
 213, 229  
     Rational Developer 8  
     WebSphere Developer 8

## T

TCP 5–6, 11, 14, 23–24, 28–29, 42, 45, 59, 61, 63, 67,  
 96, 99, 171, 180, 210, 230  
 TCP/IP 4–5, 10, 12, 14, 24, 28–29, 34, 45, 60–61, 96,  
 99–100  
     client 37, 99  
     port 37  
 TCP/IP clients 5  
 TIMEOUT 211  
 timeout 29, 71, 110–111  
 TM Resource Adapter 4–7  
 TMBER 60–61  
 TMRA 4, 8  
 Transaction Manager (TM) 3, 6  
 TRCLEV 43  
 triggers 138  
 TSO 54, 86  
 tuning 5

two-phase commit 11, 29, 37, 39, 50, 98–99, 210

## U

UK55059 247  
 UK56453 247  
 UML 76  
 Universal Driver 97, 129, 154, 209, 212  
 UNIX System Services 79, 232  
 UOR 50  
 UPDATE 46, 53, 58–59, 79, 114, 116, 118, 157–158,  
 166, 202, 214  
 URL 110–111, 130–131, 168, 171, 180  
 user exit 30–32, 53, 72  
 user exits 53, 62, 208  
 user message exit 32

## V

V9.5 FP3 229  
 VIEWDS 29  
 VIEWHWS 29  
 VIEWPORT 29  
 VSAM 242

## W

Web 2.0 6, 146  
 Web browser 249  
 Web page 163  
 Web Service 10  
 Web Services 3–4, 7–8, 126, 208  
     Description Language 7  
 Web Services (WS) 7  
 WebSphere 4, 7–8, 20, 36, 38–39, 50, 74, 76, 79,  
 100–101, 146–147, 160–161, 200, 210–211, 213, 229,  
 233  
 WebSphere Application Server 5–6, 8, 13, 35–36,  
 38–39, 101, 103, 147, 160–161, 168, 211, 213  
 WebSphere Application Server for z/OS 14, 38  
 WebSphere Integration Developer 8  
 WebSphere Integration Developer (WID) 4  
 WebSphere Process Server 8  
 WID 4  
 workload balancing 209  
 WSDL 6–7

## X

XCF 21, 45, 63, 209  
 XIB 60  
 XIBAREA 60  
 XML descriptions 76  
 XML 6, 9, 76, 113, 126, 134, 146, 186, 227  
 XML data 9, 76  
 XML schema 9, 87  
 XQuery 5, 135, 227, 230

## Z

z/OS 4–5, 8, 11–12, 18, 20, 27, 45, 47, 76, 79, 96–97,  
 160, 208–209, 211, 227–228

z/OS server 232  
z/OS subsystem 232





To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(->Hide:)>Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine.fim still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.



# IMS 11: The Open Database

(0.5" spine)  
0.475"<->0.873"  
250 <-> 459 pages

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(->Hide:>Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine:fm still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.





# IMS 11 Open Database



## Install IMS Open Database and its prerequisites

## Implement Java client access to IMS and DB2 data

## Integrate Mash up Center with IMS Open Database

IMS Version 11 continue to provide the leadership in performance, reliability and security expected from the product of choice for critical online operational applications. IMS 11 also offers new functions to help you keep pace with the evolving IT industry.

The introduction of the new IMS Enterprise Suite allows application developers with minimal knowledge of IMS Connect to start developing client applications to communicate with IMS..

With Open Database, IMS 11 also provides direct SQL access to IMS data from programs running on any distributed platform unlocking DL/I data to the world of SQL application programmers.

This IBM® Redpaper Redbooks® publication documents the steps for installing the new IMS components and their prerequisites and shows scenarios of how your client applications can take advantage of SQL to access IMS data.

We describe the installation of prerequisites such as IMS™ Connect and the Structured Call Interface component of Common Service Layer address space and document the set up of the new IMS 3 drivers:

- ▶ Universal DB resource adapter
- ▶ Universal JDBC driver
- ▶ Universal DLI driver

Our scenarios use the JDBC driver for type 4 access from Windows® to a remote DL/I database as well as DB2® tables and extend it to use IBM Mashup Center to provide an effective Web interface and integrate with Open Database.

## INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

## BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**  
[ibm.com/redbooks](http://ibm.com/redbooks)