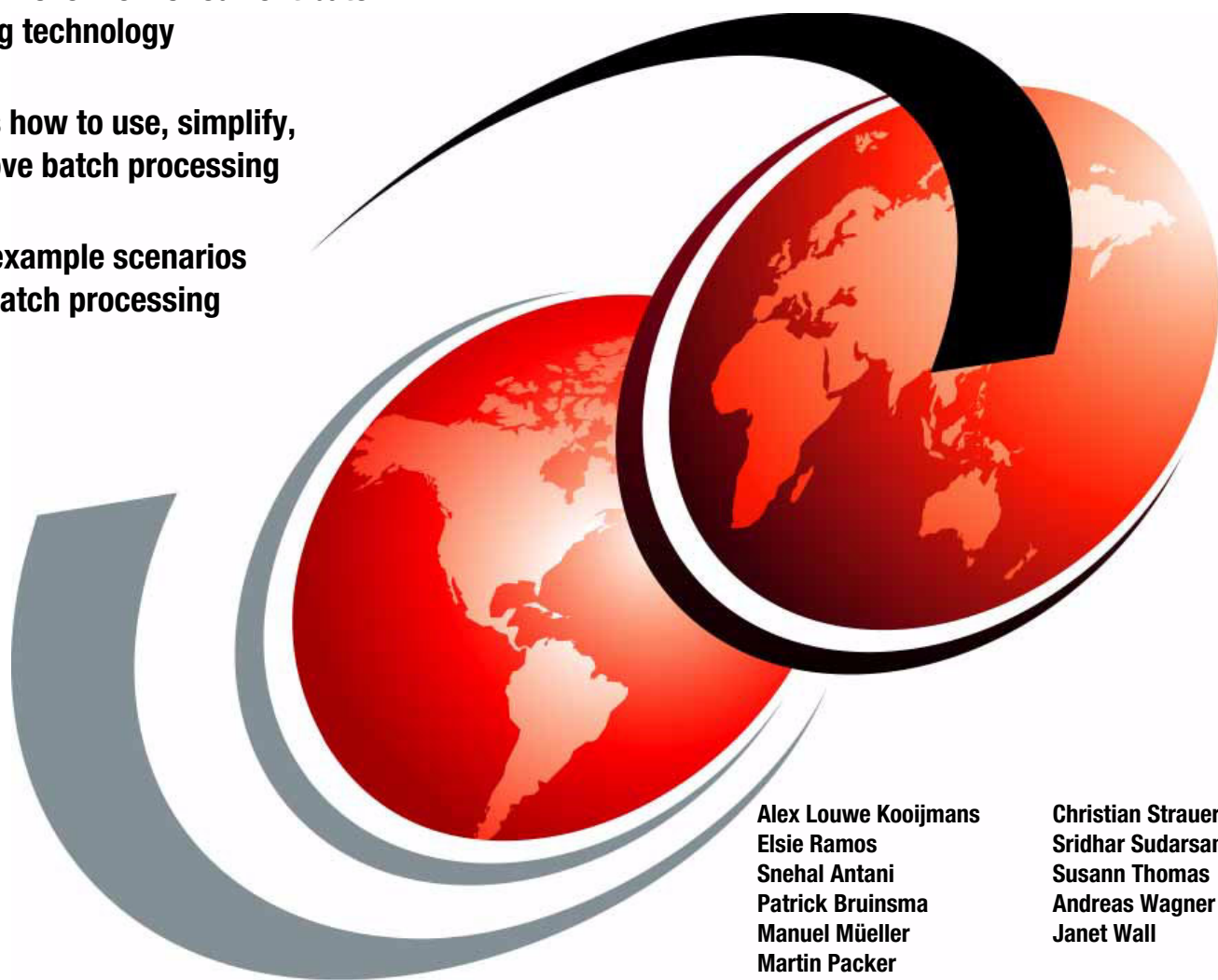


Batch Modernization on z/OS

Provides an overview of current batch processing technology

Discusses how to use, simplify, and improve batch processing

Includes example scenarios that use batch processing



Alex Louwe Kooijmans
Elsie Ramos
Snehal Antani
Patrick Bruinsma
Manuel Müller
Martin Packer

Christian Strauer
Sridhar Sudarsan
Susann Thomas
Andreas Wagner
Janet Wall

Redbooks



International Technical Support Organization

Batch Modernization on z/OS

July 2012

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

Second Edition (July 2012)

This edition applies to the following software levels:

- ▶ z/OS Version 1 Release 9
- ▶ IBM 31-bit SDK for z/OS Java 2 Technology Edition, V5
- ▶ IBM 64-bit SDK for z/OS Java 2 Technology Edition, V5
- ▶ PHP Version 5.1.2 for z/OS
- ▶ DB2 Version 9.1
- ▶ IMS Version 10
- ▶ WebSphere XD Compute Grid Version 6.1
- ▶ WebSphere Transformation Extender Version 8 Release 2
- ▶ Tivoli Workload Scheduler Version 8 Release 5 Modification 0

© Copyright International Business Machines Corporation 2009, 2012. 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

Notices	ix
Trademarks	x
Preface	xi
The team who wrote this book	xi
Now you can become a published author, too!	xiv
Comments welcome	xv
Stay connected to IBM Redbooks	xv
Summary of changes	xvii
July 2012, Second Edition	xvii
Chapter 1. Executive overview of batch modernization on z/OS	1
1.1 Do you still need batch?	2
1.2 Is replacing batch with OLTP an option?	2
1.3 Strengths of z/OS for batch	3
1.4 Batch modernization on z/OS	4
1.5 New technologies can help	5
1.6 Conclusion	6
Part 1. Overview of batch processing	7
Chapter 2. Introduction to batch modernization on z/OS	9
2.1 Differences between OLTP and batch processing	10
2.1.1 Reasons for using batch	11
2.2 Taking advantage of z/OS features for batch processing	12
2.2.1 Centralized computing model	12
2.2.2 Security	13
2.2.3 Manageability	13
2.2.4 Workload management	13
2.2.5 Reliability	14
2.2.6 Scalability	14
2.2.7 Availability	14
2.2.8 Batch processing environment	15
2.3 Drivers for change	15
2.3.1 Existing programs are not adequate for new requirements	16
2.3.2 Necessary skills to maintain and use the current technology are no longer available	17
2.3.3 The batch window needs to be shortened or made more efficient	17
2.3.4 Running batch at any time	18
2.3.5 Maintaining the actual batch programs is too complex	18
Chapter 3. Bulk processing reference architecture	19
3.1 Why do we need a reference architecture?	20
3.2 Overview	20
3.3 Bulk reference architecture	21
3.3.1 Infrastructure services	22
3.3.2 Data access management services	22
3.3.3 Bulk application container	23

3.3.4	Invocation services	24
3.3.5	System management and operations	25
3.3.6	Bulk application development	26
3.3.7	Analytics	27
3.4	Building up the bulk processing reference architecture	28
Part 2.	Serving new functional requirements in z/OS batch	31
Chapter 4.	Implement new functionality using traditional languages	33
4.1	Why use traditional languages for new functionality?	34
4.2	XML support in COBOL and PL/I	36
4.2.1	Using built-in XML support in COBOL.	36
4.2.2	Using built-in XML support in PL/I.	38
4.2.3	Using the XML Toolkit for z/OS.	39
4.2.4	Using z/OS XML System Services	39
4.2.5	Solving the “XML problem” by combining XML technologies	41
4.2.6	Using pureXML capabilities in DB2 9 for z/OS	45
4.2.7	Summary.	47
4.3	Implementing new functionality in C/C++	47
Chapter 5.	Introduction to Java on z/OS	49
5.1	The basics of Java	50
5.2	Special Java APIs for batch processing on z/OS	51
5.3	Data access with Java on z/OS	52
5.3.1	Summary.	54
5.4	Encoding issues	54
5.5	Java Interoperability with COBOL and PL/I.	55
5.5.1	Enterprise COBOL	55
5.5.2	Enterprise PL/I	56
Chapter 6.	Implement new functionality using Java in traditional containers	57
6.1	Java in CICS	58
6.2	Java in IMS	58
6.2.1	Introduction to IMS databases	58
6.2.2	Object mapped access to hierarchical data	60
6.2.3	Java applications in IMS	60
6.3	Java in DB2 for z/OS.	64
6.3.1	Java interoperability with other languages	64
6.3.2	Configuring the system environment.	65
6.3.3	Development of the sample application	67
Chapter 7.	Implement new functionality using stand-alone Java.	77
7.1	Running Java with the BPXBATCH or BPXBATSL utilities.	78
7.2	Running Java with JZOS.	79
7.3	Interoperability with other languages	80
7.4	Development tools	80
7.5	Sample stand-alone Java batch application	82
7.5.1	Creating invoice data with COBOL	83
7.5.2	Generating a PDF in Java.	84
Chapter 8.	Implement new functionality using Java in WebSphere XD	
Compute Grid	93
8.1	Java and Java Platform, Enterprise Edition	94
8.2	The Java EE runtime environment on z/OS	95

8.3	WebSphere XD Compute Grid overview	97
8.4	Quality of service in a WebSphere environment on z/OS	103
8.4.1	Security	104
8.4.2	High availability and scalability	104
8.5	Interoperability with other languages	106
8.6	Batch programming using WebSphere XD Compute Grid	106
8.7	Developing a WebSphere batch application	109
8.7.1	Setting up the environment	110
8.7.2	Creating a batch application using the BDS Framework	114
8.7.3	Testing the batch application	124
8.7.4	Running Echo batch application on WebSphere XD Compute Grid z/OS	144
8.7.5	Debugging your application in the Unit Test Server	150
8.7.6	Reusing the Echo application for huge data processing using BDS	153
8.7.7	Conclusion	155
8.8	Summary	156
	Chapter 9. Implement new functionality using PHP on z/OS	157
9.1	Introduction to PHP for z/OS	158
9.2	PHP Interoperability with other languages	160
9.3	Development tools	160
9.4	Sample application for using PHP in stand-alone batch	161
9.4.1	Eclipse setup	162
9.4.2	Importing and customizing the sample project	164
9.4.3	Implementing and deploying the PHP application	168
	Chapter 10. Summary of new functional requirements in z/OS batch	173
	Chapter 11. Batch environment enhancements in z/OS V1R13	177
11.1	Introduction to z/OS Batch Runtime	178
11.1.1	Java COBOL with DB2 interoperability	178
11.1.2	JZOS Batch Launcher enhancements	180
11.2	JES2 batch modernization	181
11.2.1	Instream data in PROCs and INCLUDEs	181
11.2.2	Support for JOBRC (job return code)	183
11.2.3	Spin and SPIN data set	185
11.2.4	Evict a job on a step boundary	185
	Part 3. Implement agile batch	187
	Chapter 12. Create agile batch by optimizing the Information Management architecture	189
12.1	Data Warehousing on System z	190
12.2	ETL	191
12.2.1	Overcoming operational inefficiency with ETL	193
12.2.2	ETL Accelerator with DataStage	200
	Chapter 13. Create agile batch by optimizing DB2 access	207
13.1	Data caching	209
13.2	Optimizing data access using SQL functionality	210
13.2.1	Joining tables	210
13.2.2	Using SELECT FROM INSERT/UPDATE/DELETE/MERGE	211
13.2.3	Multi-row processing	212
13.2.4	Explain SQL statements	212
13.3	Optimizing data access using system functionality	212

13.3.1	Optimizing using DB2 for z/OS functionality	213
13.3.2	Optimizing using I/O features	218
13.3.3	Optimizing using JDBC functionality	219
13.3.4	Checkpoint and restart functionality	221
Chapter 14.	Create agile batch by implementing trigger mechanisms	223
14.1	Job submission with native Java technology	224
14.1.1	Developing the code	225
14.1.2	Configuring WebSphere Application Server z/OS for deployment	234
14.2	Using WebSphere XD Compute Grid trigger mechanisms	234
14.2.1	The Job Management Console	235
14.2.2	The command-line interface	240
14.2.3	The WSGrid command-line utility	242
14.2.4	Web services and EJB interfaces for the Job Scheduler	249
14.3	Exploiting enhanced features of Tivoli Workload Scheduler for z/OS	263
14.3.1	Strengths of Tivoli Workload Scheduler for z/OS	263
14.3.2	Integrate Tivoli Workload Scheduler for z/OS with WebSphere XD Compute Grid	266
14.4	Triggering a DB2 stored procedure	270
14.4.1	Using WebSphere MQ to trigger a DB2 stored procedure	270
14.4.2	DB2 as a Web service provider.	279
Part 4.	Improve batch efficiency	299
Chapter 15.	Approaches and techniques to reduce the batch window	301
15.1	Project Management techniques.	302
15.1.1	Gantt charts	302
15.1.2	Dependency diagrams	303
15.1.3	Critical Path Analysis	304
15.2	Seven key strategies.	304
15.2.1	Ensuring the system is properly configured	305
15.2.2	Implementing data in memory.	305
15.2.3	Optimizing I/O	305
15.2.4	Increasing parallelism	305
15.2.5	Reducing the impact of failures.	306
15.2.6	Increasing operational effectiveness.	306
15.2.7	Improving application efficiency	306
15.3	Non-window batch.	306
Chapter 16.	Increasing concurrency by exploiting BatchPipes	309
16.1	Basic function	310
16.2	Implementation	312
16.2.1	Setting up the Pipes subsystem	313
16.2.2	Implementing Individual Pipes	313
16.3	New Pipes connectors	314
16.4	Additional Pipes functions	314
16.4.1	BatchPipePlex.	314
16.4.2	BatchPipeWorks	314
Chapter 17.	Batch application design and patterns in WebSphere XD Compute Grid	319
17.1	The Strategy pattern as the foundation for designing batch applications	320
17.1.1	The Batch Data Stream Framework and its implementation of these patterns.	324
17.1.2	Sharing business services across batch and OLTP	325

17.2 Conclusions.....	335
Chapter 18. Java performance best practices	337
18.1 Java performance in common.....	338
18.1.1 Garbage collection	338
18.1.2 Profiling.....	338
18.2 Stand-alone Java batch	339
18.2.1 JVM startup cost.....	339
Chapter 19. Increasing batch efficiency by using performance instrumentation... ..	341
19.1 System-level and WLM workload SMF	342
19.2 DB2 Subsystem-level instrumentation	343
19.2.1 DB2 Statistics Trace	344
19.2.2 DB2 Catalog	344
19.2.3 SMF 42-6	344
19.2.4 Putting Statistics Trace, DB2 Catalog and SMF 42-6 together.....	345
19.3 Batch suite instrumentation.....	345
19.4 Job-Level SMF	346
19.4.1 Data Set OPENS And CLOSEs.....	346
19.4.2 DB2 job-level Accounting Trace and deeper.....	348
19.4.3 DFSORT	351
19.4.4 BatchPipes/MVS.....	351
19.4.5 DFSMSHsm functional statistics	352
19.5 Other job-level instrumentation	352
19.5.1 SYSIBM.SYSCOPY for DB2 utility jobs	352
19.5.2 Tivoli Workload Scheduler information	352
19.5.3 Step-Termination Exit	353
19.5.4 System Log	354
Part 5. Reduce batch complexity.....	355
Chapter 20. Reduce batch complexity using a Business Rules Management System	357
20.1 Introduction to Business Rule Management	360
20.2 Overview of IBM WebSphere ILOG WebSphere BRMS	360
20.3 Using ILOG BRMS on System z	364
20.3.1 Option 1: ILOG JRules on System z using Rule Execution Server.....	365
20.3.2 Option 2: IBM WebSphere ILOG Rules for COBOL	367
20.4 Using ILOG BRMS in batch	370
Chapter 21. Reduce batch complexity using middleware for transformation logic ..	371
21.1 WebSphere Transformation Extender: Enabling universal transformation	373
21.2 Business value of WebSphere Transformation Extender	374
21.3 Sample using WebSphere Transformation Extender in z/OS batch	375
21.3.1 Creating the mapping file	377
21.3.2 Transferring files to z/OS	393
21.3.3 Running the job on z/OS to transform input data	394
Chapter 22. Reduce batch complexity by eliminating custom file transfer logic ...	397
22.1 Using WebSphere MQ FTE to perform managed file transfers.....	398
22.2 Initiating file transfer using a Java job	401
22.3 Initiating file transfer using an Ant job	403
22.4 Summary.....	404
Chapter 23. Reduce complexity by exploiting DFSORT / ICETOOL.....	405

23.1 Invoking DFSORT functions	406
23.2 Data that DFSORT can process	406
23.3 Beyond sorting	406
23.3.1 Record selection	407
23.3.2 Record reformatting	408
23.3.3 Parsing	409
23.3.4 Report writing	410
23.3.5 Record combining	410
23.3.6 Checking data	411
23.3.7 IFTHEN conditional processing	412
23.4 Symbols	414
23.4.1 Converting COBOL copybooks to DFSORT symbols	416
23.5 Invoking DFSORT from Java with JZOS	417
23.5.1 JZOS invoking DFSORT sample	417
Part 6. Appendixes	421
Appendix A. DB2 configuration	423
Data Definition Language for Java stored procedure	424
Data Definition Language for stand-alone Java application	424
DB2 SQL Insert statements for sample data	426
Appendix B. Source code	429
Java stored procedure to generate PDF files	430
Java PDF creator	437
PHP PDF creator	442
Dynamic batch Web application	445
JCL for running WebSphere Transformation Extender transformation in batch mode	448
Triggering Java Stored procedure to generate PDF files	450
Appendix C. Additional material	453
Locating the Web material	453
Using the Web material	453
System requirements for downloading the Web material	454
How to use the Web material	454
Related publications	455
IBM Redbooks publications	455
Other publications	455
Online resources	456
How to get IBM Redbooks publications	457
Help from IBM	457
Index	459

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](http://www.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®	Geographically Dispersed Parallel	QualityStage®
BatchPipes®	Sysplex™	RACF®
CICS®	HiperSockets™	Rational®
DataMirror®	IBM®	Redbooks®
DataPower®	ILOG®	Redbooks (logo)  ®
DataStage®	IMS™	RMF™
DB2®	InfoSphere®	System z10®
developerWorks®	Language Environment®	System z®
DRDA®	MVS™	Tivoli®
DS8000®	NetView®	WebSphere®
FICON®	OS/390®	z/OS®
GDPS®	Parallel Sysplex®	z10™
	pureXML®	zSeries®

The following terms are trademarks of other companies:

Intel, Itanium, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

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

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

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

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

Hibernate, Interchange, 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.

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

Preface

Mainframe computers play a central role in the daily operations of many of the world's largest corporations, and batch processing is a fundamental part of the workloads that run on the mainframe. A large portion of the workload on IBM® z/OS® systems is processed in batch mode. Although several IBM Redbooks® publications discuss application modernization on the IBM z/OS platform, this book specifically addresses batch processing in detail.

Many different technologies are available in a batch environment on z/OS systems. This book demonstrates these technologies and shows how the z/OS system offers a sophisticated environment for batch. In this practical book, we discuss a variety of themes that are of importance for batch workloads on z/OS systems and offer examples that you can try on your own system. The book also includes a chapter on future developments in batch processing.

The audience for this book includes IT architects and application developers, with a focus on batch processing on the z/OS platform.

The team who wrote this book

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

Alex Louwe Kooijmans is a project leader with the ITSO in Poughkeepsie, NY, and specializes in SOA technology and solutions on System z®. He also specializes in application modernization and transformation on z/OS. Previously, he worked as a Client IT Architect in the Financial Services sector with IBM in The Netherlands, advising financial services companies on IT issues such as software and hardware strategy and on demand. Alex has also worked at the Technical Marketing Competence Center for zSeries® and Linux in Boeblingen, Germany, providing support to customers starting up with Java and WebSphere® on System z. From 1997 to 2000, Alex completed a previous assignment with the ITSO, managing various IBM Redbooks projects and delivering workshops around the world in the area of WebSphere, Java, and e-business technology on System z. Before 1997, Alex held a variety of positions in application design and development, product support, and project management, mostly in relation to the IBM mainframe.

Elsie Ramos is a Project Leader at the International Technical Support Organization, Poughkeepsie Center. She has over 30 years of experience in IT supporting various platforms, including System z servers.

Snehal Antani works for the SOA Technology Practice within IBM Software Services for WebSphere as a senior managing consultant. He leads the consulting practice for the WebSphere XD suite of products, which includes WebSphere Virtual Enterprise, WebSphere XD Compute Grid (WebSphere batch technologies), and WebSphere eXtreme Scale. His focus is on grid, HPC, and middleware architectures and design, serving as a technical advisor to key customers, driving clients towards production, and speaking at industry conferences worldwide. Prior to joining IBM Software Services for WebSphere, Snehal worked in product development for both WebSphere Application Server for z/OS and WebSphere Extended Deployment. He has worked closely with many large distributed and z/OS clients around the world, helping them achieve production with WebSphere products as well as influence their technical strategies. Snehal has disclosed numerous patents and technical publications in the domains of enterprise application infrastructure and grid

computing. He earned a BS in computer science from Purdue University and will complete his MS in computer science from Rensselaer Polytechnic Institute (RPI) in Troy, NY with a thesis in the area of quantifying and improving the resiliency of middleware infrastructures.

Patrick Bruinsma is a certified Senior IT Specialist with 10 years of advanced experience on z/OS, OS/390®, DB2®, MQ Series, WebSphere MQ Workflow, Blaze Advisor, CICS®, Workload Manager, Java, WebSphere Application Server for z/OS, z/OS UNIX (UNIX System Services), and SAP on z/OS, as well as general installation and implementation expertise on nearly all z/OS software. He is proficient in teaching technical education and has co-authored several IBM Redbooks publications. In January 2007, Patrick was appointed System z Software IT Architect. For the last 4 years, he has focused on the design and integration of large-scale commercial computing environments, mainly in the mainframe computing arena. He supports the System z sales team and provides expertise on platform positioning and solution design.

Manuel Mueller is an IT Specialist with IBM Software Group Services. He joined IBM in 2003 as a member of the IMS™ Technical Sales team. His work centers on integration techniques of mainframe data and applications into a heterogenous landscape. He has authored an IBM Redbooks publication about WebSphere Information Integrator and has taught several workshops on System z, covering Data Propagation and Information Integration of IMS, DB2, VSAM, and others. After he joined the Software Group IM Services team, he worked on a variety of projects that are all clustered around data storage in databases and data intensive applications. This work includes the acquaintance with several products, most prominent among which are z/OS, UNIX, IMS, DB2, and SAP. He has a BS in Information Technology Management from the University of Cooperative Education (Berufsakademie) Stuttgart, Germany, and the Open University London, U.K., in a partnership program with IBM.

Martin Packer is a Senior IT Specialist working for IBM in the U.K. and Ireland. He has 24 years of mainframe performance experience, having consulted with dozens of customers in that time. He has contributed to a large number of IBM Redbooks publications over the past 20 years, has presented at many conferences, and has built a number of tools for processing mainframe performance instrumentation, most notably in the areas of batch window reduction, Parallel Sysplex® Performance, and DB2 Tuning. He holds a Bachelor's Degree in Math and Physics and a Master's Degree in Information Technology, both from the University College in London.

Christian Strauer is an IT specialist from IBM Germany. He works as a System z technical presales specialist in the Financial Services and Insurance sector. His areas of expertise are Java, Java batch, and XML solutions on System z as well as System z capacity planning. He has worked for IBM since 2002, when he started his studies at the University of Cooperative Education in Mannheim. After his studies, Christian joined the System z Field Technical Sales Support team in 2005 and was involved in many Java and XML projects on z/OS throughout Germany. Christian has authored an IBM Redbooks publication about Java Batch on z/OS. He holds a diploma in Applied Computer Sciences.

Sridhar Sudarsan is an Executive IT Architect with Software Lab Services in IBM. He has led enterprise architecture solutions for several customers worldwide for over 10 years. His clients include large companies in the finance, public sector, automobile, and SRM industry verticals. He has consulted with customers to build and realize their IT strategy. He invented the batch programming model in J2EE, now included in WebSphere Compute Grid. He champions and leads the enterprise batch modernization and modern bulk processing strategy in IBM and at customers.

Susann Thomas is an IT specialist at IBM Research and Development in Boeblingen, Germany, and specializes in modern technologies such as WebSphere solutions on z/OS. She works as a lab services specialist for Worldwide Technology Practice in the IBM Software

Group, Application, and Integration Middleware Software. Her area of expertise is WebSphere Application Server for z/OS, WebSphere XD Compute Grid on z/OS, Java batch, and XML on z/OS, as well as modern application development on z/OS. From 2001 to 2004, Susann held a variety of positions in application programming and infrastructure management and project management, mostly in relation to the IBM System z platform. She has worked for IBM since 2004, when she started her studies at the University of Cooperative Education in Mannheim. After her studies, Susann joined the System z Software Field Technical Sales Support team in 2007 and was involved in enterprise modernization, application programming, WebSphere, Java, and XML projects on z/OS throughout Germany. She holds a diploma in Economics and Informatics.

Andreas Wagner is a database administrator at GAD in Germany. He has 17 years of experience on the Mainframe (including UNIX System Services) and on distributed platforms. He worked 7 years on the mainframe as a programmer on software development projects (PL/I batch and online programs with DB2 for z/OS). For the last 10 years, he has worked as a DB2 database administrator on mainframe and distributed platforms (Linux, Windows, and AIX®). His areas of expertise include Java (JDBC and SQLJ) and WebSphere applications (J2EE and DataSource administration).

Janet Wall is the Project Manager for the IBM WebSphere ILOG® Business Rule Management System (BRMS) System z solutions. Her responsibilities and focus are on the BRMS System z product strategy and managing requirements for the product directions. Janet has approximately 30 years of IT experience in both mainframe and distributed environments. In the last 10 years, she has focused on business rule management in which she has concentrated on assisting customers to understand the benefits and how to apply best practices within BRMS. Janet was a contributing author of the Business Rule Applied Book, has authored several Business Rule articles, and has presented on Business Rule Management and Information Architecture at several IT conferences. She holds a B.S. in Computer Science from Kean University and an M.B.A. in Finance from Seton Hall University.

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

Rich Conway, Mike Connolly
IBM International Technical Support Organization, Poughkeepsie Center, USA

Budi Darmawan
Project leader, IBM International Technical Support Organization, Austin Center, USA

Norman Aaronson
System z Java Project/Release Manager, IBM Systems and Technology Group, System z Platform, Poughkeepsie, USA

Philipp Breitbach
IT Specialist, Software Services for WebSphere, IBM Software Group, Germany

Yuan-Chi Chang
Manager Database Research Group, Intelligent Information Management, IBM Thomas J. Watson Research Center, USA

Michael Dewert
DB2 Performance, Data Sharing, Recovery Technical Solution Architect, DB2 for z/OS Development - SWAT TEAM, IBM Software Group, Germany

Carl Farkas
TechWorks zWebSphere Application Integration consultant and zChampion, IBM Software Group, France

Willie Favero
Data Warehouse for System z Swat Team, DB2 SME, IBM Silicon Valley Laboratory, IBM Software Group, USA

Denis Gaebler
IT Specialist, IBM Software Group, Germany

Timothy Hahn
IBM Distinguished Engineer, IBM Software Group, Rational®

Avard Hart
System z Solution Architect, IBM USA

Eugene Kuehlthau
IT Specialist, Advanced Technical Support (ATS), USA

Patricia Pettersson
WebSphere Technical Sales, IBM Software Group, Sweden

Gary Puchkoff
Senior Technical Staff Member, IBM Systems and Technology Group. System z Strategy and Architecture, Poughkeepsie, USA

Chris Vignola
Senior Technical Staff Member, WebSphere XD Architect, Java Batch & Compute Grid, IBM Software Group, Poughkeepsie, USA

Maryela Weihrauch
IBM Distinguished Engineer, DB2 z/OS Development - Solutions Architect, IBM Silicon Valley Lab, USA

Kirk Wolf and Steve Goetze
Dovetailed Technologies, LLC

Claus Weidenfeller
GAD, Germany

Frank Yaeger
DFSORT Development Team, IBM Silicon Valley Lab, USA

Now you can become a published author, too!

Here's 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

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

- ▶ Send your comments in an email to:

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

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition might also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-7779-01
for Batch Modernization on z/OS
as created or updated on July 26, 2012.

July 2012, Second Edition

This revision reflects the addition, deletion, or modification of new and changed information described below.

New information

- ▶ Added Chapter 11, “Batch environment enhancements in z/OS V1R13” on page 177



Executive overview of batch modernization on z/OS

Today, mainframe computers play a central role in the daily operations of most of the world's largest corporations. In banking, finance, health care, insurance, public utilities, government, and a multitude of other public and private enterprises, the mainframe computer continues to form the foundation of modern business. Batch processing is a fundamental part of the workloads that run on the mainframe. This book discusses the advantages of batch processing.

The IBM mainframe is designed from the ground up to co-host OLTP and batch, with advantaged features to balance and prioritize workloads, efficiently access data, and probably most importantly, to manage large workloads and possible failures. Mission-critical batch windows often run on the mainframe with minimal on-site staff and, in some cases, even fully unattended, processing large quantities of data at relatively low cost with maximum reliability.

1.1 Do you still need batch?

For almost all companies in every industry, batch processing is still a fundamental, mission-critical component. Examples of batch processes commonly used in industries today include:

- ▶ Generating reports of all daily processed data
- ▶ Printing or sending bi-weekly account statements to all customers of a bank
- ▶ Paying salaries for all employees
- ▶ Running analytics on a large Data Warehouse
- ▶ Archiving historical data at the end of month
- ▶ Optimizing databases
- ▶ Creating backups of files and databases, for Disaster Recovery purposes
- ▶ Processing files with large amounts of data from business partners

Note: Batch jobs do not always include an application program. For example, many times a utility is used to extract data, reorganize a database, create a backup copy of data, or replicate a file.

Batch processing has significant advantages when it comes to performing repetitive logic. Static data is read only once, then cached and used throughout the program. Also, for example, one SQL query to update a status field in 10,000 rows of a database table is far more efficient than 10,000 online transaction processing (OLTP) programs, each running a query to update the status field in one row at a time.

Thus, batch processing in the sense of bulk processing of massive amounts of transactions during non-office hours remains a viable and strategic option. Because organizations want results faster without waiting for overnight OLTP processing, we will the need to run batch processes at any time during the day, in parallel with the OLTP window.

1.2 Is replacing batch with OLTP an option?

Batch is sometimes referred to as *bulk processing of OLTP logic* or as just *a program that runs in the batch window*. Reasons to design an application function as a batch process include:

- ▶ Real-time processing of the business logic is not possible because there is a dependency on something else that is not available at that time. In this case, the input of the transaction is put aside in a file or a database and processed later in a batch program. The functional design phase of an application determines whether a function is designed as a batch process or OLTP.
- ▶ A large amount of repetitive processing needs to occur, for example calculating withholding tax on the employee salaries of a large company. In this case, running each calculation as a separate OLTP transaction is inefficient, because a large part of each calculation might be using the same static data elements, such as tax percentages used.
- ▶ You need to defer CPU cycles to a time of the day when the system is not in use by online users, such as when preparing the printed invoices for the orders placed during the day.
- ▶ You need to build periodic interface files with logical groupings of records to be sent to the information systems of your partners. This type of processing occurs frequently in the banking industry. Sending one big file with many records every day might be more efficient and more reliable than sending each record in real-time.

It might look like the ideal IT environment consists of OLTP only, where each user action immediately leads to making all the updates without delay and where a user immediately receives the desired output. However, there are many functions in an information system that cannot and should not be designed and run as an OLTP function. Therefore, the design for an information system needs to identify clearly the functions that are implemented as a batch function.

1.3 Strengths of z/OS for batch

z/OS can provide its unique quality of service in combination with System z hardware and Parallel Sysplex capabilities.

The generally recognized z/OS strengths for batch fall into a number of broad categories:

▶ **Centralized computing model**

z/OS provides the advantages of a better overall utilization because of resource sharing and data proximity. This flexibility in resource sharing also results in more agility in executing OLTP and batch concurrently.

▶ **Security**

z/OS provides deep security integration, from the application level down to the operating system level and the hardware.

▶ **Manageability**

z/OS is equipped with an extensive set of management tools for managing and controlling hundreds of complex simultaneously running applications, comprising batch and online components, databases, transaction managers, and so on. Many z/OS environments run large numbers of batch jobs unattended. It is not uncommon to run 5,000 to 10,000 jobs in one evening on a System z system.

▶ **Workload management**

z/OS Workload Manager makes it possible to run many parallel batch jobs at the same time, sharing resources and balanced and prioritized according to SLA specifications.

▶ **Reliability**

System z and z/OS provide an advanced combination of availability and resilience. Failures in batch programs are recovered automatically and execution of a batch program can be deferred to another LPAR or another system in case of a disaster.

▶ **Scalability**

The z/OS Parallel Sysplex configuration provides additional horizontal scalability. Peaks in batch processing are, therefore, scaled throughout the system, acquiring additional resources as needed.

▶ **Availability**

The System z and z/OS availability features include:

- Unique mainframe clustering technology for maximum up-time (Parallel Sysplex)
- The ability to deploy participating nodes in the Sysplex cluster remotely (Geographically Dispersed Parallel Sysplex™)
- The ability to replicate data real-time at remote locations (PPRC)

- ▶ Batch processing environment

The heart of the batch environment on an IBM mainframe is the combination of a job scheduler, Job Entry System (JES), and a spool and job management tools, such as the System Display and Search Facility (SDSF).

1.4 Batch modernization on z/OS

Even if the existing batch process is currently optimally implemented in terms of performance, process controlling, and monitoring, there can be requirements that require new tools and technologies. New functional and non-functional requirements might require changes in the way that batch is organized as well as require new technologies. We refer to this process of deploying new tools and technologies in batch as *batch modernization*.

A number of drivers can lead to batch modernization on z/OS:

- ▶ Existing programming languages are not adequate for new requirements.

Most of the applications on the IBM mainframe are still written in traditional procedural programming languages such as COBOL and PL/I. New functional requirements such as creating PDF documents, sending e-mails, or just the ability to access a remote system directly, might require you to consider another programming language for batch programs. In principle this would be Java, but other languages such as PHP can be an option too.

- ▶ The necessary skills to maintain and use the current technology are no longer available.

A growing problem for many companies is that skills in certain traditional technologies are more difficult to find.

Attention: It is a common misunderstanding that a skill shortage in traditional technologies such as COBOL, CICS or IMS automatically means that the mainframe is not a viable option for the future. Practically all popular standards and technologies, such as Java, Java EE, XML, Web Services, Web 2.0, and so forth, are fully supported and integrated on the mainframe.

Note: Java development of z/OS applications can be done entirely on the workstation using Eclipse or IBM Rational tools. In the case of Java stand-alone batch, a basic understanding of JCL, TSO, ISPF, and UNIX is needed to set up the runtime environment and test on z/OS.

In the case of the WebSphere XD Compute Grid programming model, the developer does not need to work with JCL, TSO, and ISPF to deploy and test batch programs.

- ▶ The batch window needs to be shortened or made more efficient.

An indirect reason for modernizing the batch environment might be that the traditional batch window is getting shorter (because the OLTP window is getting longer). Performance optimization, increased parallelism, and moving certain batch processes into the OLTP window are options to achieve a shorter batch window. These activities might require that you deploy new technologies.

- ▶ You want to run batch at any time.

As already mentioned, just because the current batch window is getting shorter, the batch workload itself is not reduced, leading to the consideration of running a batch job either in the traditional nightly batch window or at any time during the OLTP window. Introducing

batch processes during the OLTP window might impose some challenges, especially with regards to accessing data concurrently.

- ▶ Maintaining the actual batch programs is too complex.

The number and complexity of batch programs might lead to a situation where it takes too much time to implement new business requirements, especially in the case where batch programs contain numerous lines of code for formatting and transforming data.

State-of-the-art tooling and middleware can help to make the process of maintaining complex batch applications more agile and less error prone.

1.5 New technologies can help

Introducing new technologies on the mainframe can help to address the issues mentioned in the previous section, as follows:

- ▶ Existing programming languages are not adequate for new requirements.

Java, either running stand-alone as a batch job, running inside *WebSphere XD Compute Grid* or running in a traditional environment as a DB2 stored procedure, an IMS Batch Messaging Program (BMP) or CICS, can help to address the broad arena of today's functional requirements. PHP also offers options to address functional requirements that are hard to implement in traditional programming languages.

- ▶ The necessary skills to maintain and use the current technology are no longer available.

The use of Java or PHP can also circumvent a possible skills shortage in traditional programming languages. State-of-the-art development tools on the workstation, such as Rational Application Developer, Rational Developer for System z, and Eclipse can be used to develop these types of applications. And, WebSphere technology can be used on the mainframe, to provide platform transparency for the developer.

- ▶ The batch window needs to be shortened or made more efficient.

There is generally a drive to keep the batch window as short as possible and to provide online hours on the system as long as possible. To gain time in the batch window, you can attempt to make batch smarter and better performing, but in many cases the batch window on z/OS is already optimized to the maximum.

A way to save time can be to move batch workload from the traditional batch window to the OLTP window; however, application dependencies and avoiding parallel access to master data in both OLTP and batch workload make this method a challenge. You can considerably reduce or even eliminate dependencies on master data by accessing data in a replicated data warehouse. InfoSphere® Data Warehouse, InfoSphere Replication Server, and InfoSphere DataStage® are solutions that you can use in such an architecture.

- ▶ Maintaining the actual batch programs is too complex.

The complexity of batch can sometimes become a problem. In addition to the business logic, batch programs often contain custom code for transforming data and data formats, file transfer, and ETL logic. Rather than hand coding these tasks, you can use tools that allow for more flexibility and quicker time to market:

- WebSphere Transformation Extender provides an Eclipse-based toolkit for designing and developing data transformations. These mappings are then deployed to z/OS and run as a separate program.
- WebSphere ILOG JRules provides an environment to design and maintain business rules, also in a batch environment.

- WebSphere MQ File Transfer Edition (FTE) provides an extension to the classical WebSphere MQ product to perform secure and managed file transfers to and from a z/OS batch.
- InfoSphere solutions can manage ETL tasks in batch.
- Tivoli® Workload Scheduler remains the flagship job scheduling solution on z/OS. Tivoli Workload Scheduler can work with WebSphere XD Compute Grid. Tivoli Workload Scheduler also provides a SOAP-based interface for triggering jobs as a service.

1.6 Conclusion

z/OS is a natural operating system for running large mission-critical batch workloads in a reliable way. Disaster recovery, automatic restart of broken jobs, automatic rollback of database updates to the latest sync point are all part of the z/OS solution. Hundreds of Fortune 1000 companies run mission-critical batch on z/OS. With the introduction of Java and Java-based middleware on z/OS, this quality can be put into action for a broader set of functional requirements and other technologies can help to reduce complexity or reduce the traditional batch window.



Part 1

Overview of batch processing

In this part of the book, we provide an introduction to batch processing and batch processing modernization in Chapter 2, “Introduction to batch modernization on z/OS” on page 9. We also position a framework for batch bulk processing in Chapter 3, “Bulk processing reference architecture” on page 19.



Introduction to batch modernization on z/OS

Today, mainframe computers play a central role in the daily operations of many of the world's largest corporations. While other forms of computing are used extensively in various business capacities, the mainframe occupies a coveted place in today's e-business environment. In banking, finance, health care, insurance, public utilities, government, and a multitude of other public and private enterprises, the mainframe computer continues to form the foundation of modern business.

For almost all companies in every industry, batch processing is still a fundamental, mission-critical component. In this book, we describe aspects of modern batch processing and point out why it might be necessary to change the current batch process to achieve modern business requirements.

In this chapter, we introduce the concept of batch modernization and discuss the following topics:

- ▶ Differences between OLTP and batch processing
- ▶ Taking advantage of z/OS features for batch processing
- ▶ Drivers for change

2.1 Differences between OLTP and batch processing

Data processing on the mainframe can be grouped into two main categories:

- ▶ Online transaction processing (OLTP)
- ▶ Batch processing

OLTP is triggered by a user with a direct response. To initiate OLTP, users typically complete an entry form or select other appropriate actions through a user interface application component. The user interface component then initiates the associated online transaction with the business logic in the background. When the transaction is complete, the same user interface or other user interface component presents the result of the transaction to the user. The response can be data or can be a message regarding the success or failure of the processing of the input data.

During the processing time of the online transaction, the user typically has to wait and cannot work with the user interface. Therefore, it is extremely important that the transaction is finished in the shortest time possible. Thus, the scope of online transactions and the amount of data that is processed has to be relatively small to minimize the locking of resources on the system.

In contrast, batch processes require no user activity. Most batch programs read data from various sources (for example databases, files, and message queues), process that data, and then store the result. On the mainframe, batch programs are often designed to handle very large amounts of data within a very short time (such as millions of records in an elapsed time of just a few minutes).

Figure 2-1 illustrates the difference between OLTP and batch processing.

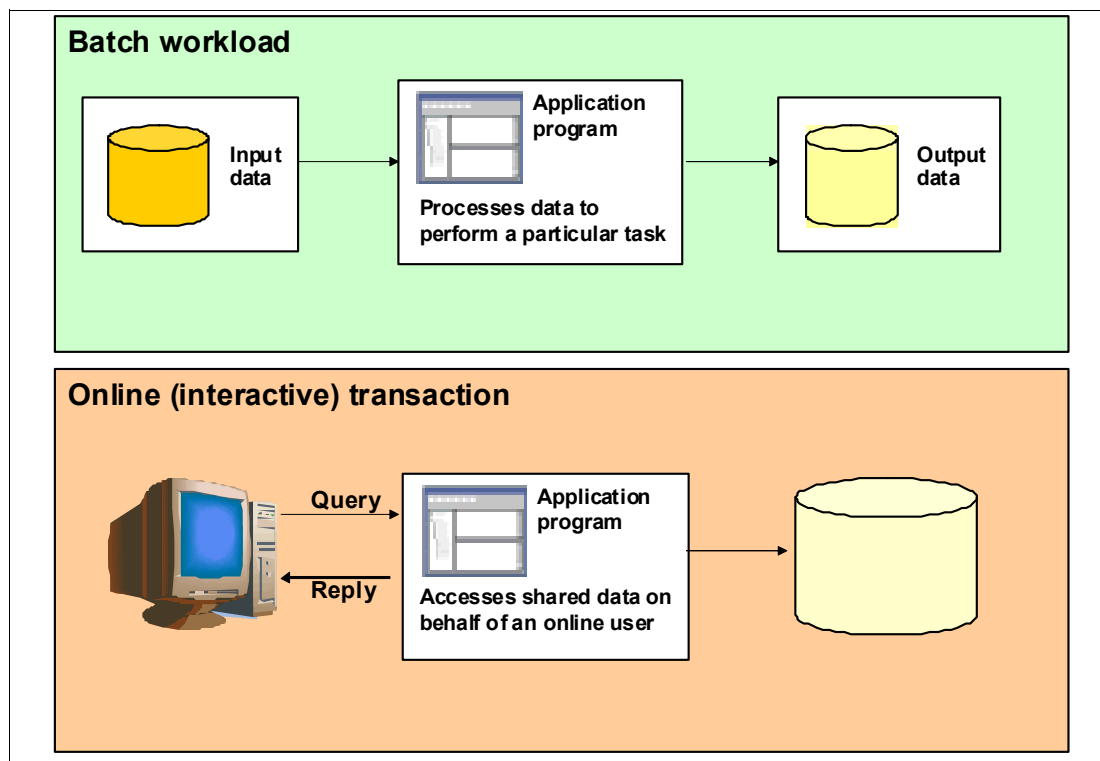


Figure 2-1 Difference between OLTP and batch processing

Generally, one batch program generates output data that then is used as input by another batch program. Because of this type of dependency, a “network” of batch jobs can grow and become quite complex very quickly. Appropriate tools, such as Tivoli Workload Scheduler, help when executing and monitoring batch programs within a large job network.

In large IT environments, batch jobs usually run overnight. During this *batch window*, online activity is restricted or even completely forbidden. Because the online capability is extremely relevant for almost all companies, the batch window must end as soon as possible or at least before the committed start time of the OLTP window. The start and times of the batch window (which can be different on different days of the week or month) are specified in the service-level agreement (SLA), which also includes additional information regarding management of errors, recovery, and so forth.

Important: The batch window can vary greatly depending on the day of the week or month. For example, most companies have different job networks on weekdays, weekends, the last workday of the week, and the last day of the month. End of week and end of month processing can make a job network significant more complex and longer to run. These factors are all included in the SLA.

Examples of batch activities include:

- ▶ Generating daily reports of processed data
- ▶ Paying salaries for employees
- ▶ Archiving historical data at the end of month
- ▶ Reorganizing data
- ▶ Processing files with large amounts of data from a partner

Note: Batch jobs do not always include an application program. For example, many times a utility is used to extract data, reorganize a database, create a backup copy of data, or replicate a file.

Batch processing provides the following advantages:

- ▶ A batch window and an online window allow balanced system usage with almost constant 100% utilization, helping to optimize IT resources and save costs. In other words, you use the mainframe during the non-office hours for work that does not need immediate response, freeing up cycles during the office hours for transactions that need immediate response.
- ▶ Batch programs are more efficient performing repetitive logic.
- ▶ Postponing transactions can help to achieve more business security, for example intervening into money transfers that might damage the business.

2.1.1 Reasons for using batch

Batch is sometimes referred to as *bulk processing of OLTP logic* or as just a *program that runs in the batch window*. Reasons to design an application function as a batch process include:

- ▶ Real-time processing of the business logic is not possible because there is a dependency on something else that is not available at that time. In this case, the input of the transaction is put aside in a file or a database and processed later in a batch program. Whether this is the case is discovered in the functional design of the application.
- ▶ A large amount of repetitive processing needs to occur, for example calculating withholding tax on the employee salaries of a large company. In this case, running each

calculation as a separate OLTP transaction is inefficient, because a large part of each calculation might be using the same static data elements, such as tax percentages used.

- ▶ You need to defer CPU cycles to a time of the day when the system is not in use by online users, such as when preparing the printed invoices for the orders placed during the day.
- ▶ You need to build periodic interface files with logical groupings of records to be sent to the information systems of your partners. This type of processing occurs frequently in the banking industry. Sending one big file with many records every day might be more efficient and more reliable than sending each record in real-time.

It might look like the ideal IT environment consists of OLTP only, where each user action immediately leads to making all the updates without delay and where a user immediately receives the desired output. However, there are many functions in an information system that cannot and should not be designed and run as an OLTP function. Therefore, the design for an information system needs to identify clearly the functions that are implemented as a batch function.

2.2 Taking advantage of z/OS features for batch processing

Generally, z/OS strengths for batch processing fall into a number of broad categories:

- ▶ Centralized computing model
- ▶ Security
- ▶ Manageability
- ▶ Workload management
- ▶ Reliability
- ▶ Scalability
- ▶ Availability
- ▶ Batch processing environment

We describe each of these features in detail in the sections that follow.

2.2.1 Centralized computing model

The *centralized computing model* is characterized by a single or few, but large, computing nodes. These nodes host a variety of applications that all share the same computing resources, such as memory, CPU, and disk and tape storage. This sharing of resources leads to an overall better utilization of those resources and less idle time. All users are directly connected to this system.

Another key aspect of the centralized computing model is data proximity. Having applications are in close proximity to the data that they use seriously reduces communication overhead, increases security, and minimizes points of failure, all of which effects batch workloads positively as such workloads typically process huge amounts of data.

2.2.2 Security

z/OS provides deep *security* integration, from the application level down to the operating system level and the hardware.

Centralized security

The Resource Access Control Facility (RACF®) provides centralized security functions, such as user identification and authentication, resource access control, and auditing for both the operating system and applications running on the system.

With the introduction of the System Authorization Facility suite of services, a centralized point was created within the operating system infrastructure through which both operating system components and applications could invoke the services of the z/OS resident security manager.

Auditing and logging

A very important feature of a centralized authentication and access control mechanism is the ability to record and analyze security information. The audit data is essential for ensuring that the customer's installation security policy is followed. The RACF option of the z/OS Security Server provides multiple ways to specify what security-relevant events are recorded in the audit stream and how that information is reduced and analyzed. RACF provides a wide variety of capabilities and tools to the auditor for data analysis.

Accountability

Accountability in z/OS is achieved by a combination of user authentication and the ability to propagate a user's credentials throughout the application. The System Management Facility (SMF) interface is designed for collecting performance and accounting information. z/OS will collect all accounting and performance data and present this back to the installation through RMF and SMF records. These records can be analyzed and aggregated in performance and capacity reports, but can also be used as a base for security analysis and accounting policies.

Network security

Networking and communications security on z/OS is provided by the Communications Server element of z/OS. The Communications Server provides networking and communication services for accessing z/OS applications over both SNA and TCP/IP networks. You can exploit these features using batch. Thus, batch can be run very secure on z/OS.

2.2.3 Manageability

The z/OS operating system is designed to manage multiple workloads in a single system image, providing the means to manage complex environments. As a consequence, z/OS is equipped with an extensive set of management tools to manage and control thousands of complex, simultaneously running applications, comprised of batch and online components, databases, transaction managers, and so on.

2.2.4 Workload management

The z/OS Workload Manager (WLM) controls the distribution of workloads over different partitions and the prioritization of work within a partition. CPU resources can be reassigned dynamically through Intelligent Resource Director (IRD). WLM and IRD working together provide unique on-demand capacity services. Furthermore, WLM can work with Sysplex

Distributor to load-balance workloads throughout the systems in the sysplex and make the network entry point into the system highly available.

These capabilities make the mainframe an ideal platform for running mixed workloads of hundreds of different business-critical applications while achieving very high resource utilization. Of course, batch workloads are also managed by WLM. Thus, it is possible to run hundreds of parallel batch jobs at the same time, which is very difficult on other platforms.

2.2.5 Reliability

System z and z/OS provide an advanced combination of availability and resilience. The combination of redundancy and virtualization delivers extraordinary levels of application availability and recoverability. The IBM mainframe architecture protects users from hardware failures as well as software failures. If an application fails, its workload can be picked up by a backup of the same application running on the same physical hardware.

Mainframe workloads can be shared by systems running up to 100 km apart, allowing for smooth disaster recovery procedures. And when the time comes to add system capacity or to replace a failed hardware module, those hardware changes can be made without interrupting customer service. Some mainframes run uninterrupted for years, adapting to changing needs, workloads, and applications while continuously in production. Because batch can be very critical, the reliability of the mainframe is a big advantage.

2.2.6 Scalability

The z/OS Parallel Sysplex configuration provides additional horizontal scalability. A sysplex can be a cluster of up to 32 z/OS images in different partitions on different System z systems (that are possibly geographically dispersed) with full data sharing and high availability and recovery.

The machines participating in a Parallel Sysplex can be physically dispersed up to 100 km from one another, which gives the capability of having a physically decentralized infrastructure that is logically centralized.

With z/OS, the horizontal scalability is not limited to specific workloads. *Any* workload can take advantage of the Parallel Sysplex scaling options. Scalability is fundamental in an on-demand world, and the IT infrastructure should be able to scale when business demands more resources.

Another key performance and scalability benefit is provided by the HiperSockets™ virtual TCP/IP network, which eliminates network delays between applications and subsystems running within a mainframe.

2.2.7 Availability

System z has many components that address availability.

Parallel Sysplex, the clustering solution for the z/OS environment, provides both scalability and availability. A failure of one image in the cluster does not affect the other images, and any specific transaction running on the Parallel Sysplex can be fully dispatched and recovered in any other image, making use of the full data sharing architecture.

For higher availability and disaster recovery purposes, a Parallel Sysplex can be configured in a Geographically Dispersed Parallel Sysplex (GDPS®) mode.

The System z and z/OS availability features are:

- ▶ Unique mainframe clustering technology for maximum up time (Parallel Sysplex)
- ▶ The ability to deploy participating nodes in the Sysplex cluster remotely (Geographically Dispersed Parallel Sysplex)
- ▶ The ability to replicate data real-time at remote locations (PPRC)

2.2.8 Batch processing environment

Another outstanding capability of the mainframe architecture, in addition to transaction processing, is the capability to run batch workloads. In a z/OS environment, dedicated subsystems (JES2 or JES3) in combination with special job scheduling software, such as Tivoli Workload Scheduler, manage batch processing.

Today, the IBM mainframe is better positioned than ever to handle batch processing. With the ability to run batch processes written in the Java language making use of lower-cost specialty processors and with the ability to access back-end data, either on z/OS or any other server, z/OS has become a very cost-effective platform for running business-critical batch applications.

Because of its nature, high performance batch processing has several requirements for the platform on which the batch workload will be run:

- ▶ Deliver superior OLTP and Batch Processing performance, often concurrently, on the same platform with dynamic workload management.
- ▶ Take advantage of proximity of data, the most important factor for high performance batch processing, because batch is inherently I/O intensive.
- ▶ Address heavy I/O workload without channel/device contention.
- ▶ Provide enterprise-wide management of system resources.
- ▶ Provide parallel execution of workloads.
- ▶ Provide automatic job scheduling.
- ▶ Deliver scalability to meet growing batch processing volume and time pressures.
- ▶ Offer extreme high availability to meet critical batch processing demands.
- ▶ Provide Lock Management to manage updates to the same record without corrupting data.

These requirements are handled extremely well with the following System z capabilities:

- ▶ Workload Management
- ▶ Powerful I/O subsystem
- ▶ Unmatched scalability, availability, and security features

2.3 Drivers for change

Today, the current IT infrastructure for large mainframe customers is getting more and more complex. In addition to the core business that often runs completely on the mainframe, some applications also process work on distributed systems. The work is often split into online and batch processes.

Even if the existing batch process is implemented optimally in terms of performance, process controlling, and monitoring, there could still be requirements that might lead to deploying new tools, processes, and technologies.

In the following sections, we discuss the key drivers for change in a batch environment on z/OS. In the remainder of the book, we continue to work with these themes.

2.3.1 Existing programs are not adequate for new requirements

Most of the applications on the IBM mainframe are still written in traditional programming languages, such as COBOL and PL/I. The examples in this section show possible requirements for business applications that are difficult or even impossible to realize with these traditional programming languages.

Generating documents in PDF format

Producing documents in a Portable Document Format (PDF) is becoming increasingly more important because PDF is an open standard that allows users to exchange and view documents on many different computing platforms. For example, using PDF allows bank customers to receive account statements electronically.

Because PDF is an open standard, you can also generate these PDF documents on the mainframe using traditional languages. However, doing so can be difficult because there are no public libraries available to easily generate PDF documents. Alternatively, with modern languages such as Java, it is fairly straightforward to create a document in PDF because there are corresponding public libraries available.

When using batch, it makes sense to create PDF files, for example for sending salary statements to employees.

Sending e-mails

The ability to send e-mails has become quite a common requirement in modern applications, for example using an e-mail confirmation for notification that a payment is processed. Although you can accomplish this requirement using a traditional programming language, it is quite complicated. Again, for a language such as Java, this requirement is easy to implement.

Processing XML

Because of its flexibility in data structure, XML is very suitable for exchanging any kind of data. In fact, XML has established itself as a new standard for data exchange. For example, with DB2 V9, you can store XML data natively in DB2 databases, allowing a user to query data using XML Query Language (XQuery). Also, other mainframe components have expanded regarding XML capabilities, including extensions to traditional languages (Enterprise COBOL and Enterprise PL/I) and the system environment (XML System Services).

Although you can now process XML data with traditional languages, this method is not always recommended because of some limitations. However, because XML is also very important for batch, it is best to evaluate the different options, including XML in all different languages, from COBOL and PL/I using C/C++ to Java.

Implementation of new industry standards

In the early days of computing, only mainframe computers existed. All standards regarding computing were developed on the mainframe and, therefore, were immediately available within mainframe programming languages.

Today, new standards are also developed on distributed platforms and become available on the mainframe through new types of middleware, such as Business Process Execution Language (BPEL), Web Services, and Web 2.0. In some cases, these new standards do not make it into traditional programming languages on z/OS.

Access to remote systems

Through the implementation of service-oriented architecture (SOA), software development has changed massively in the last years. More companies use remote systems that are integrated in the current business workflow, increasing the need to access data and business logic on remote systems.

Again, traditional programming languages might not be adequate to directly access a remote database or to perform a Remote Procedure Call (RPC) to a remote program on any platform.

Today, different technologies are available to access remote systems, such as Web Services, which is positioned as the protocol of choice in an SOA. Web Services is supported in J2EE servers as well as in Java.

2.3.2 Necessary skills to maintain and use the current technology are no longer available

A growing problem for many companies is that skills in traditional technologies are more difficult to find. New employees might not have knowledge of traditional mainframe technologies because their education normally is not mainframe minded. Typical mainframe skills include:

- ▶ Job Control Language (JCL)
- ▶ Traditional programming languages (COBOL, PL/I, and Assembler)
- ▶ ISPF user interface and TSO command-line interface
- ▶ General architecture of the mainframe
- ▶ Concepts of programming on the mainframe
- ▶ Using the ISPF editor
- ▶ Working with traditional MVS data sets

To work with a traditional batch environment on z/OS, a basic, although extensive, training is required. Also, it takes time to really become a professional. Using new standards and programming languages can “soften” the training requirement significantly, especially when WebSphere middleware and a Java batch programming model is used for batch programs. In this case, there is no need to have JCL, TSO, ISPF, and COBOL skills.

Note: Java development of z/OS applications can be done entirely on the workstation using Eclipse or IBM Rational tools. In the case of Java stand-alone batch, a basic understanding of JCL, TSO, ISPF, and UNIX is needed to set up the runtime environment and to test on z/OS.

In the case of the WebSphere XD Compute Grid programming model, the developer does not need to work with JCL, TSO, and ISPF to deploy and test batch programs.

2.3.3 The batch window needs to be shortened or made more efficient

Another reason for changing existing processes in a batch environment might be that the traditional batch window is getting smaller. In the early days of computing, batch work usually ran overnight. During this time, no online activity was possible. Today, many applications need

a longer online time, sometimes up to 7x24. As a consequence, the batch window is constantly shrinking. Reasons to extend the online workload to 7x24 include:

- ▶ Extending business to the Internet (for example Internet banking and online shopping)
- ▶ Providing application availability to global offices in many different countries

Although online activities are broadening, batch workload is still necessary. This situation causes online transactions increasingly to run parallel to batch programs. Because online and batch applications usually share the same data, the risk of performance degradation grows, which can influence online or batch processing. Thus, changing the current batch architecture might be necessary to minimize effects regarding a shrinking batch window.

2.3.4 Running batch at any time

As already mentioned, just because the current batch window is getting shorter, the batch workload itself is not reduced, leading to the consideration of running a batch job either in the traditional nightly batch window or at any time during the OLTP window. Furthermore, it might happen that you need more flexibility because your users want to have a greater influence on the execution time of some batch jobs. For these reasons, you might want to change the current batch process to allow execution of batch jobs at any time.

2.3.5 Maintaining the actual batch programs is too complex

Batch processes in large companies can be quite complex. It is not unusual that those companies run thousands of different batch jobs in their environment. In most cases, each of these jobs is built of several steps. Within these steps, the data is processed either with standard utilities or with self-written programs.

Often these jobs or steps have dependencies on each other. The sum of these dependencies can quickly result in a very complex network of different batch jobs and steps.

An inspection of the entire batch process is of crucial importance. Because of the complexity of the network of different batch jobs and steps, only tools such as Tivoli Workload Scheduler are suitable to handle such a big job network. These tools guarantee the correct submission of all jobs at the right time or, in case of errors, generate a corresponding alarm.

In addition to the complexity of the network of jobs, the batch programs themselves can become very complex due to continuous development, which results in increased complexity of programs. Alternatively, batch programs can contain a large amount of custom code, which can be eliminated completely by using adequate tools.

Thus, based on the complexity of the existing job network or batch programs, a change of the current batch process might be necessary to increase the maintainability of the program.



Bulk processing reference architecture

Recent studies in IBM on batch modernization have shown some interesting observations. We have already discussed some of them in 2.3, “Drivers for change” on page 15. In this chapter, we discuss other observations that we use to build a reference model.

There are a number of batch implementations, across industries and across geographies at enterprise and application levels. Many of these are proprietary implementations, built several years ago, to cater to specific requirements. Some of the basic requirements might be the same, but there might be need to scale. With current architectures, that might not be feasible or practical.

There are redundancies in code, data access, and operational management because of different OLTP and batch platforms, which results in additional cost to own and maintain disparate systems. Java plays an important role in OLTP systems, so it does pay to use Java in the batch applications where it makes sense.

3.1 Why do we need a reference architecture?

There is a general confusion over classification of workloads in the industry. There are a number of purviews under which batch workloads might fall, the most common and recognized one being the traditional batch process that is executed in a dedicated batch window. The dwindling skills in the bulk processing community has led to confusion over application of workload styles, such as batch; online transaction processing (OLTP); extract, transform, and load (ETL); event processing; straight through processing (STP); and so forth. For example, ETL is predominantly batch operations but is a separate workload type in itself. Similarly, analytics, scientific processing and stream processing are all different manifestations of batch or bulk processing. Adequate tools and processes do not exist to design bulk workloads, which leads to inefficiency in execution and under utilization of existing middleware, tools, management utilities, and IT infrastructure.

Before we talk about various aspects and capabilities of the products and platforms in batch processing, it is important to understand and speak the same language between all parties—business, application, and IT architects who are designing such solution; developers, database developers, and administrators; and operations teams and system administrators.

We have created a reference model for bulk processing to set all the parameters in one place, irrespective of technology or product choices. Then, we look at what capabilities exist at each component and layer.

3.2 Overview

The *bulk processing reference* architecture provides a blueprint for various kinds of batch processing that apply to a single application or across an enterprise. This architecture is based on the proven way of establishing building blocks of bulk processing, with a partially layered architecture using services and components and data flows that represent and implement bulk business processes. An architecture built using relationships between layers can help design bulk processes in the context of any architecture, which is a significant step towards making batch processing a first class workload. This architecture enables bulk business processes to be modeled as they should be, as bulk IT processes and services, to gain the economies of scale in execution. Using a reference architecture for bulk processing aligns with modern trends in computing, such as service-oriented architecture (SOA) design techniques, which help address the declining skills issues that we describe in 2.3.2, “Necessary skills to maintain and use the current technology are no longer available” on page 17.

The major capability that the bulk processing solution stack offers is reuse of existing assets. There are a large number of existing batch applications that are written in various programming languages and that run on mainframe platforms that can and should be reused. As business and IT transformation roadmaps are created and executed, in the initial stages, re-use is critical to minimize down time and to understand the approaches to re-engineer assets for the next several years.

The bulk processing reference model is an extensible and flexible architecture blueprint that provides:

- ▶ Reduced IT cost, which allows reuse of business and application artifacts (design, implementation, and system management) throughout OLTP and batch systems.
- ▶ The capability to take advantage current, highly available IT skills to design, implement, and manage bulk applications using modern programming languages and techniques.
- ▶ Optimized and efficient execution to run bulk applications on heterogeneous platforms and to take advantage of the platform capabilities.
- ▶ Flexibility in the design of bulk processes based on business requirements that can be interleaved with OLTP processes to take advantage of technology rather than being constrained by it.
- ▶ Agility to deliver applications align to business needs quicker and with lower overhead.

Layers allow a clear separation of concerns to facilitate the design of bulk processes. The bulk processing reference model defines a blueprint that can be used to then define the layers, the components within each layer, the options available at each layer, and the typical architectural decisions that need to be made.

3.3 Bulk reference architecture

The layers illustrated in figure Figure 3-1 depict a set of logical layers in the bulk processing reference architecture. This architecture follows the *Layers*¹ pattern, but we define it as *partially layered* architecture. Although we do follow a pattern where components within each layer follow the same abstraction, there might be direct dependencies between a layer and any layer below it, not just the immediate layer. For example, the invocation and scheduling services can invoke a data access management service where bulk data warehousing jobs are performed.

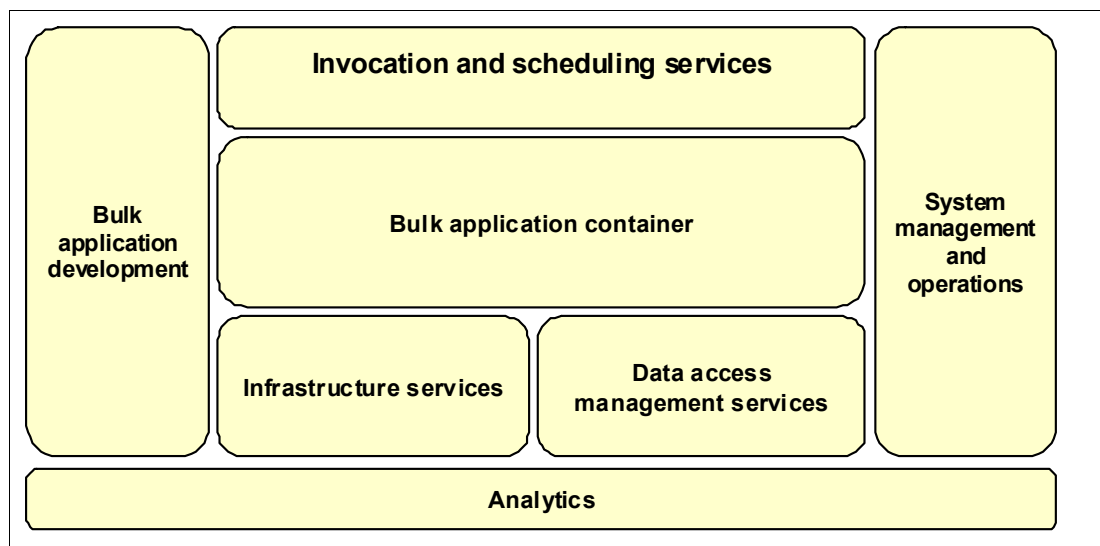


Figure 3-1 High-level bulk processing reference architecture

¹ The Layers architectural pattern helps to structure applications that can be decomposed into groups of subtasks in which each group of subtasks is at a particular level of abstraction. See Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. Pattern-Oriented Software Architecture: A System Of Patterns. West Sussex, England: John Wiley & Sons Ltd., 1996.

Sometimes, the concepts of layers and tiers are often used interchangeably. However, the important difference to note here is that a *layer* is a logical structuring mechanism for the elements that make up the software solution, whereas a *tier* is a physical structuring mechanism for the system infrastructure. In this section, we discuss only the logical structuring of the software solution elements.

All bulk applications are likely to have some manifestation of each of these layers in the implementation; however, they are not mandatory in the design. While each of the layers have a key role to play, the level of importance in a given solution can be different. The components might also differ based on a number of factors, including business and IT key performance indicators (KPI), existing applications and infrastructure, as well as time and budget constraints.

The core of the execution environment for running batch applications is in the center that comprises the Invocation and scheduling services, bulk application container, data access management services, and Infrastructure services. The surrounding layers—bulk application development, system management, and operations and analytics—are used to develop, manage, and optimize bulk applications and apply throughout the life cycle.

In the following sections, we look at each of these layers in detail and discuss the relationship between them.

3.3.1 Infrastructure services

The *infrastructure services* layer constitutes the base of the operational environment. This layer contains the hardware infrastructure, including the servers, network, data storage, and SAN, as well as the target operating system and platform on which the bulk processes execute. Generally, all available server and operating systems can be used in this layer. z/OS, Linux for System z, distributed platforms running various versions of UNIX, and Windows can be the operating system of choice in the infrastructure services.

A number of functions are available in this layer. For example, a large number of bulk applications use file data streams. To execute these bulk jobs efficiently, one step in the batch flow typically sorts files. The recommended method to sort files, when possible, is to use the functions that are provided by the operating system, such as DFSORT. This sorting of files leads to a general best practice that the processing occur as close to the data as possible to get the best performance.

Note: Taking advantage of optimized functions such as file sort and merge in the infrastructure services is key to executing efficient bulk applications.

3.3.2 Data access management services

By definition, batch or bulk applications are largely dependent on processing high data volumes. These high data volumes can be stored in a number of different types of data stores, such as relational database or files or coming in as streams or continuous feeds.

The *data access management* services layer, as shown in Figure 3-2, comprises a persistent and transient, or in-memory, data store that is used to front the persistent data. Data can be stored in any structured format in files. Then, the data can be read in chunks or one file at a time into an input data stream for further processing. Similarly, data can be accessed by reading messages from (or writing messages to) a queue. For example, this pattern of reading data from a queue can be used when trying to read streams of data coming in as a feed from a business partner.

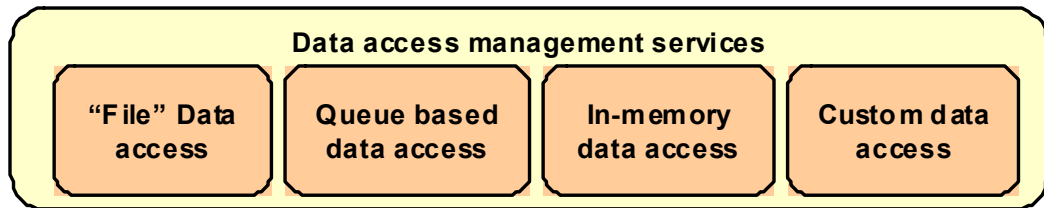


Figure 3-2 Data access management services

The “In-memory data access” in the figure represents either an in-memory database (IMDB) or an in-memory data grid (IMDG). With performance and throughput being paramount to completing jobs before a deadline, topologies using IMDB or IMDG are attractive. There are specific decision factors that go into selecting one or the other:

- ▶ An IMDB has more flexibility in storing the data in a normalized form and better querying capabilities.
- ▶ An IMDG tends to be more programming language based (for example WebSphere eXtreme Scale has a POJO-based framework) and provides more scalability. A programming language based framework can be used to store:
 - In-memory transient referential data that is used to process all records
 - In-memory chunks of input stream records to avoid an I/O for every record
 - In-memory chunks of output stream records before “flushing” them to an output I/O stream

Data can also be read from or written to queues. The queues can represent an input stream of data that is processed by the container. It also represents messages being put in queues for further consumption downstream after the unit of work processes records.

These are the most commonly used data access forms that are in play in the majority of the batch jobs. There might be some custom data access patterns in use, because of a more complex assortment of data from multiple data sources, or due to other interoperability reasons. These data access mechanisms should be used cautiously in green-field application developments weighing trade-offs between maintenance and upkeep costs with the flexibility that it provides.

All data access management services need to be highly optimized because there is a potential to invoke a service at least once for every record that is processed in bulk applications. Given volumes that are typically processed in batch jobs, this processing can affect throughput significantly.

3.3.3 Bulk application container

The *bulk application container* provides a programming model and a framework for managing the life cycle of each bulk application. It allows jobs to be submitted, canceled, suspended, or resumed. The core of a job step’s life cycle provided by any bulk application or batch container provides the following set of life cycle management steps:

```

initialize job step and data streams;
loop
  read data from the input data streams;
  process data;
  write data to an output stream;

```

```
checkpoint, if required;  
loopend  
clean up job step and data streams;
```

A *checkpoint* is one of the key distinguishing features of bulk jobs from OLTP applications, in that data gets “committed” in chunks, along with other required housekeeping to maintain recovery information for restarting jobs. An extreme example is doing a checkpoint after every record, which equates to how OLTP applications typically work. At the other extreme is doing a checkpoint at the end of the job step, which might not be feasible in most cases, because recoveries can be expensive and too many locks can be held in the database for too much time. It is generally somewhere between these two extremes. The checkpoint interval can vary depending on a number of factors, such as whether jobs are run concurrently with OLTP applications, how long locks can be held, the amount of hardware resources available, SLAs to be met, time available to meet deadlines, and so forth. Depending on the technology used, there are static ways of setting these checkpoint intervals, but ideally checkpoint intervals can be set dynamically as well.

What differentiates containers is the quality of services, including but not limited to:

- ▶ The management and operation of the jobs
- ▶ The level of sophistication in providing check-pointing mechanisms
- ▶ Ease in configuring job parameters
- ▶ Integration with schedulers invoking jobs
- ▶ Portability to run in different environments

3.3.4 Invocation services

All bulk applications need to be initiated or invoked. As shown in Figure 3-3, there are two methods of job invocation:

- ▶ Planned jobs
- ▶ Ad-hoc jobs

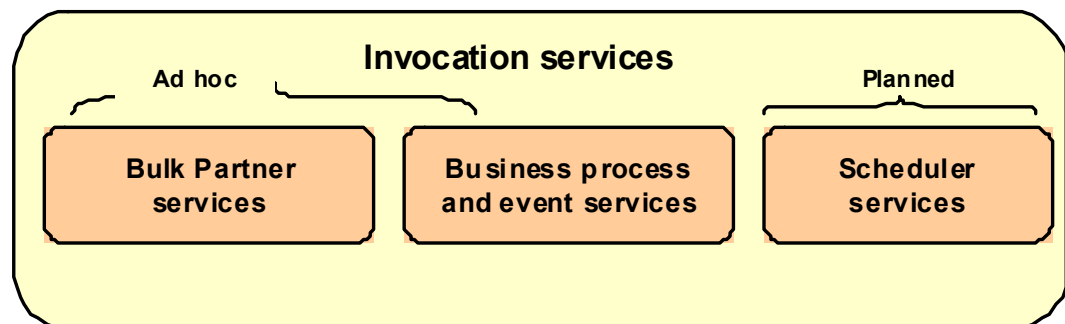


Figure 3-3 Invocation services

Planned jobs are set to be invoked based on pre-determined criteria, for example calendar date, time of the day, or completion of prior jobs based on a network of job flows defined. In the early days of computing, planned jobs were the only way jobs were invoked, and batch jobs ran when OLTP systems shut down or at night after provisioning dedicated resources to execute batch jobs. Planned jobs are in vogue in many enterprises today and, in many cases, make business sense, because there are deadlines to meet as set by regulatory boards or customers. Planned jobs provide more control to allocate resources for the workload type. Arguably, in some cases, problem determination is easier. A good example of a planned job is sending inventory status updates to the retail headquarters from a store done twice a day, at noon and midnight.

Ad hoc jobs are invoked because of a triggered event or rule and can run at any time. For example, if business partners send files of data that need to be processed, a job can be triggered to start after every n files are available, where n is a set number of files. Using the same retail inventory status update example, an ad hoc trigger rule can be set to start the status update job when 30% of the items are at 60% of original. Using ad hoc rules is better aligned with a direct business need, provides flexibility, and is a first step in running data processing when needed so that the customer gets near real-time access to information.

Combining the planned and ad hoc jobs is fairly normal and takes advantage of the strengths of both. The complexity is shifted to designing the applications and resource provisioning. In Chapter 14, “Create agile batch by implementing trigger mechanisms” on page 223, we discuss in more detail different methods to trigger ad hoc batch jobs.

3.3.5 System management and operations

All bulk processes or batch jobs, whether they are ad hoc or planned, need to be managed. Figure 3-4 illustrates the life cycle of batch jobs.

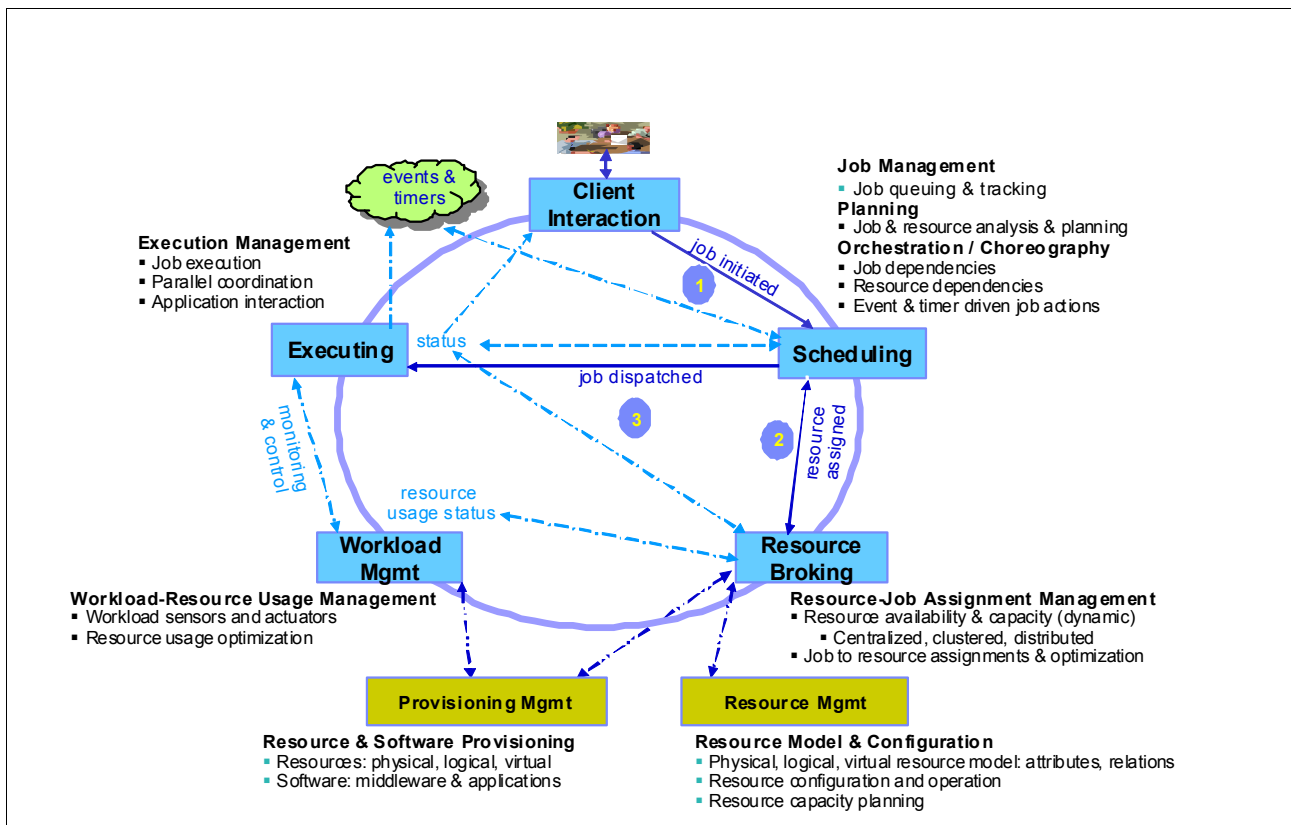


Figure 3-4 Batch application life cycle - execution and management

Figure 3-4 shows a comprehensive set of steps that are required in the management and execution of every job:

1. The client first initiates the job, either triggered by an event or at a pre-set schedule.
2. The scheduler then schedules this job.
3. The scheduler queues the job for dispatch and tracking.

This phase analyses the job and the resources available for execution and plans for the dispatch of the job. In order to plan for the resources, optionally a *resource broker* can be

used. For example, Tivoli Workload Scheduler uses a resource broker to manage the assignment of resources to a job, based on availability and capacity determined dynamically. The resource broker uses services from a resource manager that has the physical and virtual resource model and its attributes and an overview of the resource configuration. Tivoli Dynamic Workload Broker provides a number of these capabilities. A provisioning manager discovers and tracks data center resources to enable more accurate server provisioning and software deployments. It also facilitates efforts to consistently follow user defined policies and preferred configurations, in support of corporate and regulatory compliance efforts. The workload management helps with balancing the workload across available resources to optimize resource usage. A solution could be built using all or a subset of these capabilities, depending on cost and complexity of the jobs that need to be executed and managed.

4. The scheduler then dispatches the job to the execution environment where, upon completion, the status is sent back to the client.

3.3.6 Bulk application development

Bulk applications can be built from scratch as the result of green-field requirements. Alternatively, as in many cases, they can be the result of an evolutionary migration, re-engineering or re-designing a project of existing batch applications. Figure 3-5 shows the approaches that can be followed for development.

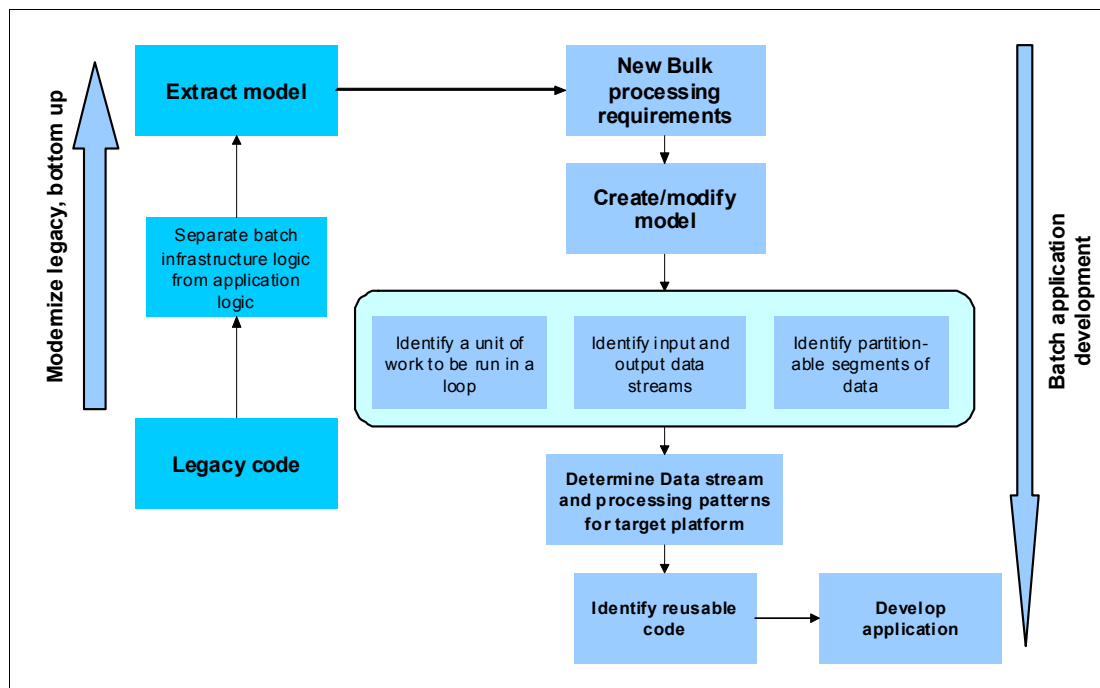


Figure 3-5 Application life cycle: Development and migration

For modernization, the approach is based on a guiding principle: *Do not perform code translations without doing a proper due diligence*. Simply performing a code translation of batch applications from, say, COBOL to Java without due diligence will probably prove expensive and unworkable. Moreover, it will produce unreadable, unmaintainable, and inefficient code. To understand what the platform-independent model for the batch jobs would be, the business logic and flows need to be abstracted, which will provide a platform independent model.

A combination of manual processes and tools can be used to create these business logic and data flow artifacts. Doing so ensures that current implementations and future requirements can be captured more accurately. The platform independent models can then be implemented in any platform specific model and programming language using a model driven approach.

For the more green-field requirements for bulk processing, the best practice is to follow a similar approach of first modeling the solution. The key elements to be identified are the data streams and the unit of work to be applied to the data in the streams. Identifying partitionable segments of data helps with optimizing the design and helps to design an efficient implementation. Depending on the target platform of choice, a number of other elements specific to each programming model and runtime environment will need to be factored in the design and implementation.

This development approach does not focus on the deployment and testing aspects of bulk applications in detail. Common best practices in OLTP applications also apply to bulk applications deployment and testing.

3.3.7 Analytics

The *analytics* component applies to every element in the bulk processing reference architecture. For example, in the invocation optimization component, you want to have insight into the ability to determine the best target application server for an incoming job request. This information can be determined by simple algorithms to some very sophisticated techniques using analytics. Some of the simplest implementations will maintain a table of the available target servers, and a server is chosen using round robin or randomly. At the other end of the spectrum of algorithms, you have a self-learning system that determines a target available server based on historical data of getting the best throughput. This determination requires collecting and analyzing data from the infrastructure, data, and application containers over a period of time.

If there are shared resources between OLTP and bulk applications, the workload manager can make decisions based on business rules that then drive the checkpoint policies. For example, WLM might determine the following conditions based on predefined SLAs:

- ▶ A batch job is negatively impacting the OLTP, and OLTP is more important
 - Dynamically reduce the checkpoint interval for the batch jobs
 - Dynamically inject pauses between the checkpoint intervals for the batch jobs
- ▶ The deadline for a particular batch job is approaching, we need to speed it up
 - Dynamically increase the checkpoint interval for the batch jobs
 - Dynamically decrease pauses between the checkpoint intervals for the batch jobs

Most of the current systems have some level of analytics built in, which we describe in the following chapters.

3.4 Building up the bulk processing reference architecture

We have now described a bit more detail about each layer, resulting in a diagram as shown in Figure 3-6.

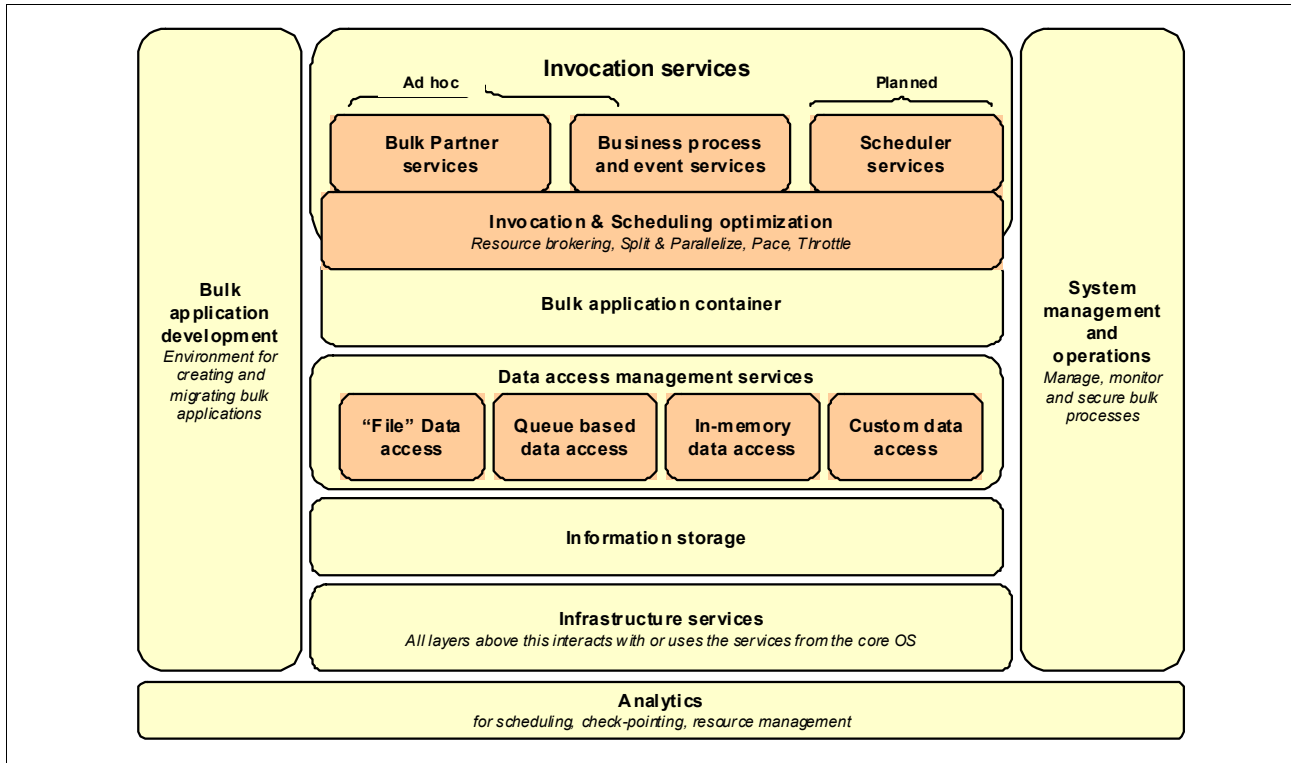


Figure 3-6 Bulk processing reference architecture

We use this reference architecture as we describe the various products that enable batch processing on z/OS in this book. We unfold and dive deeper into each layer using specific products and capabilities of the products to provide a better understanding of how the layers all can be used in conjunction to build a solution specific to your bulk processing needs.

Figure 3-7 illustrates the products that can be used in each layer. We discuss some of these products in more detail in the remainder of this book.

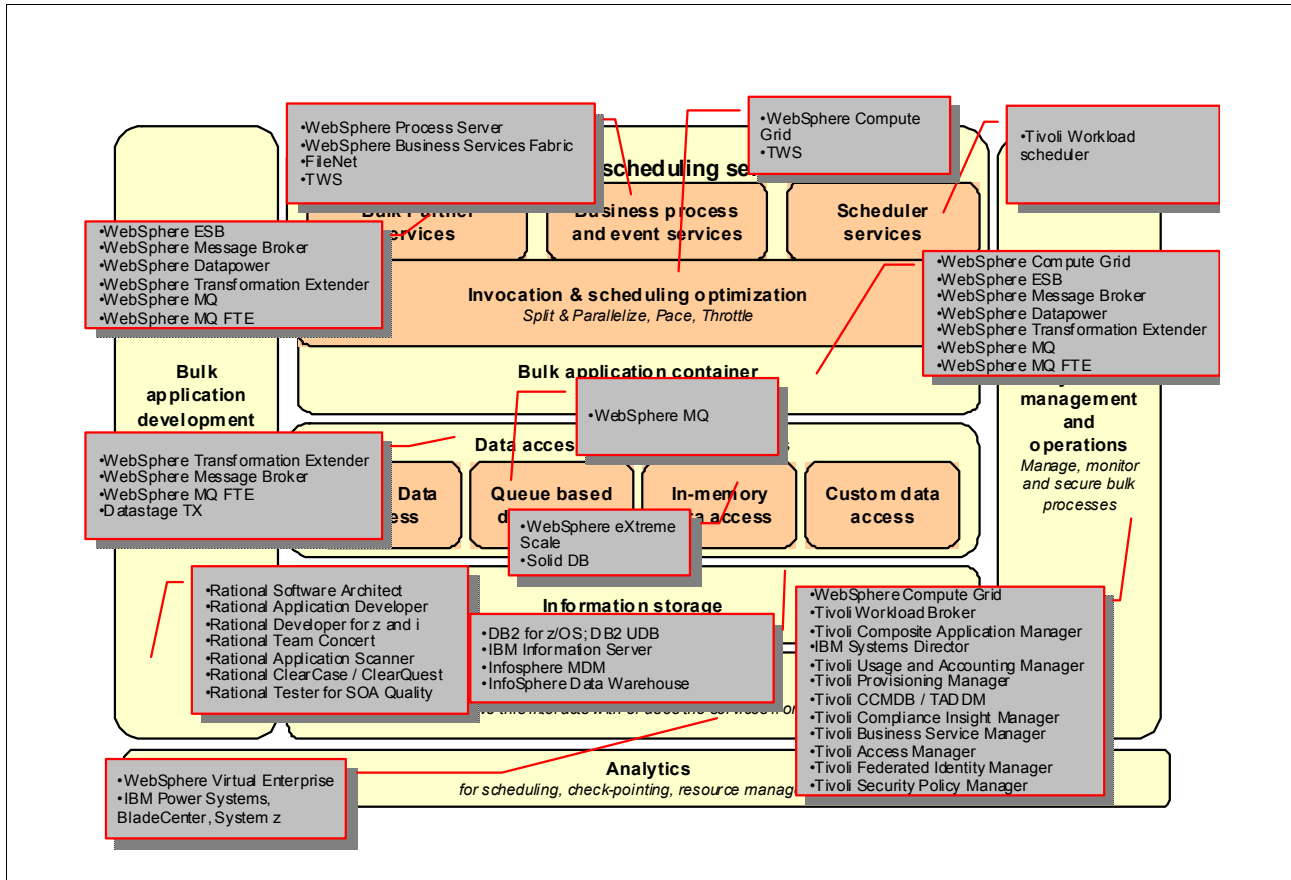


Figure 3-7 Bulk processing reference architecture with mapped products

Serving new functional requirements in z/OS batch

One of the drivers for change in a batch environment is the introduction of functional requirements that cannot be met easily with existing traditional technology. Unfortunately, it is sometimes very difficult to handle these requirements in an existing batch environment due to constraints in the technology and, especially, constraints in the programming language in use.

Examples of requirements that might require new technology on z/OS include:

- ▶ Integration with the outside world, for example through Web Services
- ▶ Expansion of print jobs to create PDF files, for example sending invoice files as a PDF to customers
- ▶ Generating electronic reports in formats such as *.xls or *.doc
- ▶ Generating diagrams for reports
- ▶ Sending e-mail
- ▶ Sending Short Messages (SMS) to recipients
- ▶ Direct remote access to heterogeneous systems using protocols such as TCP/IP, HTTP, and Web Services
- ▶ Direct remote access to heterogeneous databases on other platforms

- ▶ Imbedding functionality provided by applications from independent software vendors (ISVs) that are not written in traditional z/OS programming languages
- ▶ Using different versions of XML:
 - Parsing XML data
 - Storing and forwarding XML data
 - Generating XML data
 - Transforming XML data into other formats

As shown with these examples, these types of requirements cannot be implemented on z/OS using traditional batch technologies, such as JCL and COBOL.

Note: What is *traditional* technology actually? In the context of this book, we identify traditional technology as technologies that are widely in use on z/OS for batch processing for decades, such as JCL, MVS data sets, COBOL, PL/I, and Assembler. These technologies are well proven and provide excellent quality of service but, unfortunately, do not provide support for all today's requirements. In contrast, *modern* technologies include programming languages, frameworks, and middleware that have been added to z/OS over the past decade, are also well proven, but still not widely in use on z/OS.

In this part of the book, we discuss different new technologies with a focus on supporting new state-of-the-art requirements. The choice of the programming language is key when it comes to supporting functional requirements, and we discuss scenarios using traditional programming languages as well as new programming languages such as Java and PHP.

We also discuss different runtime environments in combination with Java. In addition to using Java stand-alone in JCL, you can also use Java in a runtime container, such as WebSphere XD Compute Grid, inside an IMS BMP, or inside a DB2 stored procedure. In all of these, cases the developer has access to a broader functionality.

We include samples that you can download from the Web site as described in Appendix C, "Additional material" on page 453.

We discuss the following technologies in this part:

- ▶ In Chapter 4, "Implement new functionality using traditional languages" on page 33, we discuss the enhanced capabilities of COBOL and PL/I and how to use the XML capabilities.
- ▶ In Chapter 5, "Introduction to Java on z/OS" on page 49, we discuss using the Java language and follow on with the following chapters that discuss using Java in different environments:
 - In Chapter 6, "Implement new functionality using Java in traditional containers" on page 57, we discuss traditional runtime environments, including CICS, IMS, and DB2.
 - In Chapter 7, "Implement new functionality using stand-alone Java" on page 77, we discuss stand-alone in JCL.
 - In Chapter 8, "Implement new functionality using Java in WebSphere XD Compute Grid" on page 93, we discuss WebSphere XD Compute Grid.
- ▶ In Chapter 9, "Implement new functionality using PHP on z/OS" on page 157, we discuss PHP in stand-alone batch.



Implement new functionality using traditional languages

This chapter provides a brief overview of how to get access to a broader functionality when using traditional languages on z/OS such as COBOL, PL/I, C/C++. We discuss the recently added XML capabilities in COBOL and PL/I.

This chapter includes the following topics:

- ▶ Why use traditional languages for new functionality?
- ▶ XML support in COBOL and PL/I
- ▶ Implementing new functionality in C/C++

4.1 Why use traditional languages for new functionality?

In today's programming, with programs written in high-level and scripting languages such as Java and PHP, it is easy to forget that a significant number of current business applications on mainframes are written using traditional languages such as Assembler, COBOL, and PL/I.

The data serving capabilities of System z provide a reliable foundation for an optimized IT infrastructure as required in today's business environments. COBOL and PL/I, with their procedural programming model, provide a huge strength for processing massive amounts of data. If you run a traditional data intensive application on z/OS, your application also benefits from the System z hardware, the z/OS operating system, and database system strengths such as fast data access, high-availability, scalability, and reliability.

Figure 4-1 shows the development cycle for traditional z/OS applications written in COBOL or PL/I, starting with creating the source code and ending with an executable module.

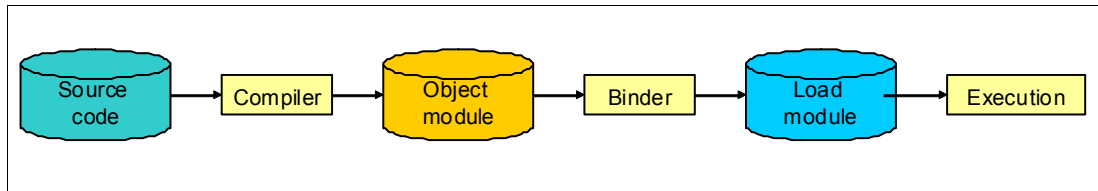


Figure 4-1 Process flow for an application using a traditional procedural programming language

The steps in this cycle are as follows:

1. In the compile step, modules of source code are translated into relocatable machine language in the form of *object code*.
2. Object code must be processed by a binder (or linkage editor or loader) before it can be executed. The *binder* resolves unresolved references in the object code and ensures that everything that the program needs is included in the final load module.
3. To execute a load module, the binder loads the module into real storage and then transfers control to it to begin execution.

If you have the requirement to enrich your existing traditional batch applications with new modern technology such as XML or Web Services or if you only want to introduce new techniques to modernize your environment, you must be aware of the following questions:

- ▶ Which programming languages on z/OS fulfill your requirements?
- ▶ Which skills are required for development?
- ▶ Are modern tools available that make development easier?

To select the programming language is one important decision in case you need to design a new application or to enrich an existing application with new techniques. You must be aware of the strengths, as well as the weaknesses, of each language to make the best choice, based on the particular requirements of the application.

A crucial factor in choosing a programming language is determining which functionality you need to implement. If COBOL or PL/I is used for most of the applications in your environment, it is likely the language of choice for the new application as well because of the existing skills in your development team and for reason of standardization. There might be a case, however, for using multiple languages to take advantage of the strengths of a particular language for only certain parts of the application. It might be a good choice to write frequently invoked subroutines in Java or C/C++, even if the standard language is COBOL or PL/I. Each programming language has its inherent strengths, and an application designer should exploit

these strengths. If a given application merits the added complication of writing it in multiple languages, the designer should take advantage of the particular features of each language. Keep in mind, however, that when it is time to update the application, multiple skills must be available. Thus, complexity in design must always be weighed against ease of maintenance.

You might find that your investment in applications written in traditional languages is reaching its end and that you have many reasons to move to a higher level language such as Java, but that you are not ready to change all your applications to Java or another modern language.

Remember that COBOL or PL/I, though not today's hottest technology, still offers a very efficient way to fulfil most enterprise computing requirements, which is operating on data. Although there seems to be a rush to code everything in Java and other Web-enabled languages, COBOL or PL/I remain a proven language to handle large volumes of data to be processed on the mainframe.

Reasons to convert from a lower-level procedural language to a higher-level language include:

- ▶ Traditional code can be difficult to maintain because of a shortage of available skills.
- ▶ Tooling for higher-level languages is more state-of-the-art and more productive.
- ▶ There is a lack of functionality in lower-level languages.

COBOL and PL/I sometimes have a lack in functionality to serve certain new requirements. However, in some areas, these languages have advanced significantly and can now be used for fulfilling the following requirements that were not possible a few years ago:

- ▶ XML support in COBOL and PL/I

The XML support available in COBOL and PL/I can be used to for certain functionality requiring the usage of XML, but both COBOL and PL/I have limitations compared to Java. Also, the syntax of COBOL or PL/I applications draws a lot of energy and attention from the developer instead of focusing on the thinking process needed for software development. However, for small and simple applications that only process XML documents it is not difficult to use the XML support available in COBOL and PL/I.

- ▶ Interoperability with C/C++

The COBOL and PL/I compilers provide compatibility for calling a C/C++ program, making it easier to decide to implement enhanced functionality in C/C++ modules and reuse those modules in COBOL and PL/I programs.

- ▶ Interoperability with Java

The COBOL and PL/I compilers provide compatibility for integrating traditional programs with Java components. (See 5.5, "Java Interoperability with COBOL and PL/I" on page 55 for more detailed information.)

- ▶ Unicode support

Unicode support for Enterprise COBOL and PL/I also helps to modernize your applications, particularly the case if your application is used across countries or regions with different languages. More information is provided in *Enterprise COBOL for z/OS Programming Guide Version 4 Release 1*, SC23-8529 and *Enterprise PL/I for z/OS Programming Guide Version 3 Release 8*, SC27-1457.

- ▶ The earlier traditional compilers for z/OS do not take advantage of the System z hardware advancements. The latest compilers take advantage of the hardware exploitation.

4.2 XML support in COBOL and PL/I

System z is a platform with large pools of structured data that are of high value to applications and middleware. XML processing is very data intensive, and introducing XML in the application usually has consequences for the performance and CPU cost, compared to not using XML.

IBM is continuously optimizing the technology to make the usage of XML as transparent as possible in terms of performance and CPU cost and currently provides the z/OS XML System Services, which is a unique XML parser written in Assembler for z/OS. z/OS XML System Services is part of the z/OS operating system. The latest version of Enterprise COBOL and PL/I, the XML Toolkit for z/OS and also DB2 9 for z/OS and CICS TS Version 4.1 are exploiters of the z/OS XML System Services parser to provide the best possible performance in XML processing on z/OS and to offload the workload on the zIIP and zAAP specialty engines. (See 4.2.4, “Using z/OS XML System Services” on page 39 for detailed information.)

If you decide to use XML processing in a COBOL or PL/I application, you can use the following technologies to implement the required functions in your traditional batch environment:

- ▶ Using built-in XML support in COBOL
- ▶ Using built-in XML support in PL/I
- ▶ Using the XML Toolkit for z/OS
- ▶ Using z/OS XML System Services
- ▶ Solving the “XML problem” by combining XML technologies
- ▶ Using pureXML capabilities in DB2 9 for z/OS

Note: For more information, refer to *XML Processing on z/OS*, SG24-7810. This book provides information about XML processing technologies on z/OS, with emphasis on z/OS XML System Services, the z/OS XML Toolkit, Enterprise COBOL and PL/I built-in XML processing support, and Java XML processing in the SDK including usage in various z/OS environments such as CICS, IMS, and Rational Developer for System z, including code samples.

In the following sections, we discuss each of these options in more detail.

4.2.1 Using built-in XML support in COBOL

The easiest way to implement XML processing in your traditional COBOL application is to use the COBOL built-in subroutine which provides a basic SAX-style parsing. However, you will notice that you will reach the limitations quickly, for example in the validation functionality. This option is recommended only for small and easy implementations.

Processing XML input

You can process the XML input in your COBOL program by using the XML PARSE statement. The XML PARSE statement is the COBOL language interface to either of two high-speed XML parsers. You use the XMLPARSE compiler option to choose the appropriate parser for your application:

- ▶ XMLPARSE(XMLSS)

This compiler option selects the z/OS XML System Services parser. This option provides enhanced features such as namespace processing and conversion of text fragments to national character representation (Unicode UTF-16) and 100% of z/OS XML System

services parsing is eligible for zAAP. See 4.2.4, “Using z/OS XML System Services” on page 39 for more information.

► **XMLPARSE(COMPAT)**

This compiler option selects the XML parser that is built into the COBOL library. This option provides compatibility with XML parsing in Enterprise COBOL Version 3.

Processing XML input involves passing control to and receiving control from the XML parser. You start this exchange of control by using the XML PARSE statement, which specifies a processing procedure that receives control from the XML parser to handle the parser events. You use special registers in your processing procedure to exchange information with the parser.

Use the following COBOL facilities to process XML input:

- Use the XML PARSE statement to begin XML parsing and to identify the document and your processing procedure
- Use the ENCODING phrase of the XML PARSE statement to specify the encoding of the XML document
- Use your processing procedure to control the parsing, that is to receive and process XML events and associated document fragments and to return to the parser for continued processing
- Use the following special registers to receive and pass information:
 - XML-CODE to receive the status of XML parsing and, in some cases, to return information to the parser
 - XML-EVENT to receive the name of each XML event from the parser
 - XML-NTEXT to receive XML document fragments that are returned as national character data
 - XML-TEXT to receive document fragments that are returned as alphanumeric data
 - XML-NAMESPACE or XML-NNAMESPACE to receive a namespace identifier for a NAMESPACE-DECLARATION XML event or for an element name or attribute name that is in a namespace
 - XML-NAMESPACE-PREFIX or XML-NNAMESPACE-PREFIX to receive a namespace prefix for a NAMESPACE-DECLARATION XML event or for an element name or attribute name that is prefixed

The XML namespace special registers are undefined outside the processing procedure.

You can use the ENCODING phrase and the RETURNING NATIONAL phrase of the XML PARSE statement only when the XMLPARSE(XMLSS) compiler option is in effect.

Link-edit considerations: COBOL programs that contain the XMLPARSE statement must be link-edited with AMODE 31.

Processing XML output

You can produce XML output from a COBOL program using the XML GENERATE statement. In the XML GENERATE statement, you identify the source and the output data items. You can optionally also identify:

- A field to receive a count of the XML characters generated
- A code page in which the generated XML document is to be encoded
- A namespace for the generated document

- ▶ A namespace prefix to qualify the start and end tag of each element, if you specify a namespace
- ▶ A statement to receive control if an exception occurs

Optionally, you can generate an XML declaration for the document and can cause eligible source data items to be expressed as attributes in the output rather than as elements.

You can use the XML-CODE special register to determine the status of XML generation.

After you transform COBOL data items to XML, you can use the resulting XML output in various ways, such as deploying it in a Web service, passing it as a message to WebSphere MQ, or transmitting it for subsequent conversion to a CICS communication area.

Link-edit considerations: COBOL programs that contain the XML GENERATE statement must be link-edited with AMODE 31.

4.2.2 Using built-in XML support in PL/I

Enterprise PL/I for z/OS allows existing PL/I transactions to process inbound and outbound XML data directly within the applications. It provides a high-speed parser that enables PL/I programs to parse XML documents in EBCDIC, ASCII or UTF-16. Using the IBM PL/I Simple API for XML (SAX) parser, this XML can then be passed to other applications, even those running on other platforms, including IMS and CICS environments. With PL/I Enterprise V3.3 and later, Enterprise PL/I also supports the generation of XML. Using a built-in function, you can dump the contents of a structure as XML into a buffer. IBM PL/I provides a SAX-like event-based interface for parsing XML documents. The parser invokes an application-supplied handler for parser events, passing references to the corresponding document fragments.

The parser has the following characteristics:

- ▶ It provides high-performance, but non-standard interfaces.
- ▶ It supports XML files encoded in either Unicode UTF-16 or a single-byte code page.
- ▶ The parser is non-validating, but it does partially check for well-formed errors and generates exception events if it discovers any errors.

The PLISAXx (x = A or B or C) built-in subroutines provide basic XML parsing capability, which allows programs to consume inbound XML documents, check the syntax, and react to the content. These subroutines do not provide XML generation, which must instead be accomplished by PL/I program logic or by using the XMLCHAR built-in function.

PLISAXA and PLISAXB have no special environmental requirements. They execute in all the principal runtime environments, including CICS, IMS, and WebSphere MQ, as well as z/OS batch and TSO. PLISAXA and PLISAXB do have some important limits, they have no support for XML name spaces, no support for Unicode UTF-8 documents, and they require that the entire XML document be passed to them (either in a buffer or a file) before they do any parsing of it.

PLISAXC has no special environmental requirements except that it is not supported in AMODE 24. It executes in all the principal runtime environments, including CICS, IMS, and WebSphere MQ, as well as z/OS batch and TSO.

The new PLISAXC built-in subroutine uses the z/OS XML System Services and this workload is 100% eligible for the zAAP specialty engine. See 4.2.4, “Using z/OS XML System Services” on page 39 for more information.

Note: You can find more information regarding XML support in *Enterprise PL/I for z/OS Programming Guide Version 3 Release 8*, SC27-1457.

4.2.3 Using the XML Toolkit for z/OS

If you require more than simple XML parsing functionality in your traditional batch application on z/OS, it can be very useful to call a C/C++ program from COBOL or PL/I. The IBM XML Toolkit for z/OS is designed to provide a valuable infrastructure component to assist you in creating, integrating, and maintaining your business-to-business solutions.

The XML Toolkit for z/OS is EuroReady. It is based on a cross-platform, open source code based parser (C/C++) that is designed to be compliant with industry standards.

IBM XML Toolkit for z/OS, V1.10 continues to provide enhanced support for the XML Parser, C++ Edition and the XSLT Processor, C++ Edition. The XML Parser, C++ Edition is updated with the following support:

- ▶ Ability to optionally utilize z/OS XML System Services (see 4.2.4, “Using z/OS XML System Services” on page 39) as an underlying parsing technology when performing Document Object Model (DOM) and Simple API for XML (SAX2) based parsing operations. Support is provided for both non-validating parsing as well as validating parsing utilizing schema based on the W3C Schema recommendation. This enhancement is provided by way of a set of new z/OS-specific parser C++ classes that are similar in name to and closely mimic the existing DOM and SAX2 interfaces. Utilizing z/OS XML provides redirection to zAAP specialty processors of the portion of the XML parsing operation performed by z/OS XML and might result in significant improved raw performance as well.
- ▶ A new feature that supports importing multiple schemas with the same namespace.
- ▶ Improved source offset support, enhancing the ability to obtain information that correlates parsed output with the associated data in the input document being parsed. This new support is included in the new z/OS-specific parser classes described above.

Note: Optional usage of z/OS XML by XML Parser, C++ Edition users for non-validating parsing is also available in XML Toolkit V1.9 by using PTFs UA40707 and UA40708.

4.2.4 Using z/OS XML System Services

The “XML problem” was identified as an issue in 2003 and probably earlier. New programming models were wrapping everything in XML to enable transactions to take place in a more generic fashion throughout the network. XML is used to provide envelope information and metadata about the transaction. Traditional back-end transaction processing needed to handle the new XML-based standards, such as SOAP, efficiently or risk becoming obsolete. Users were looking at heavy additional CPU costs to support the same transaction load that they have always managed, just because the requesters of those transactions require them to be in an XML form instead of a traditional (and proprietary) form.

IBM provides the z/OS XML System Services (z/OS XML) as a new line item developed to address the soaring demand for XML processing workloads on System z. The z/OS XML parser is a system level XML parser that is integrated with the base z/OS operating system. It

is intended for use by system components, middleware, and applications that need a simple, efficient, XML parsing solution. z/OS XML can currently be accessed by a C/C++ or an Assembler programming interface. The z/OS XML parser includes the following specifications and functions:

- ▶ Is written in Assembler language.
- ▶ Is an integrated parser for z/OS and a non-Java environment.
- ▶ Provides a buffer-in, buffer-out processing model instead of the event-driven model that is common to SAX parsers.
- ▶ Natively handles a number of character encodings, including UTF-8, UTF-16 (big endian), IBM-1047, and IBM-037.
- ▶ Uses buffer spanning to handle documents of unbounded length.
- ▶ Contains minimal linkage overhead.
- ▶ Provides the ability for parsing operations to be run 100% on a System z Application Assist Processor (zAAP) when called in TCB mode or a System z10@ Integrated Information Processor (zIIP) when called in SRB mode, for example DRDA® mode.
- ▶ Provides assistive aids to the user in debugging not well formed documents.

z/OS XML is an integrated component of the z/OS base, so no additional installation is required. This integration also makes z/OS XML a high-performance parser, supporting z/OS environments where minimum overhead is a priority, such as SRB and cross-memory modes. The interface itself is simple and efficient, consisting of five Assembler-level callable services that avoid event-driven interface overhead. z/OS XML parses its XML documents to a self-defining binary form that is easy to navigate all while complying with the World Wide Web Consortium's XML specifications.

z/OS XML, like most XML parsers, can parse and check the syntax of an entire XML document. z/OS XML rises above other parsers, however, with its ability to parse incomplete documents as part of a series of fragments that make up an entire XML document. This ability saves on storage overhead when dealing with large XML documents that could be hundreds of megabytes in size. Rather than allocate one large buffer to fit a particular XML document, z/OS XML allows applications to allocate, use, and reuse a smaller buffer in memory to step through the entire document. A 100 MB document, for example, could be sequentially parsed in 100 fragments of 1 MB each, thereby greatly reducing an application's memory footprint. z/OS XML allows greater customization for storage management by supporting application-specific exits for allocating and de-allocating storage. z/OS XML also supports a string identifier exit which allows applications to further save on storage by assigning numerical identifiers to strings from the XML document. Finally, z/OS XML provides an interface to efficiently query an XML document's encoding.

You call the z/OS XML parser natively from a COBOL or PL/I application, but, for example, to handle a pointer in COBOL is not as simple, which is why Enterprise COBOL and PL/I, DB2 for z/OS, and the XML Toolkit for z/OS exploit the z/OS XML parser to reduce the application development and to increase performance.

4.2.5 Solving the “XML problem” by combining XML technologies

Traditional transaction processing on z/OS increasingly means handling XML, driving up CPU costs, which has led to several issues:

- ▶ The XML Toolkit was not performing well.
- ▶ z/OS XML is very fast but lacks the interfaces that most people use (SAX2 or DOM).
- ▶ z/OS XML also lacks a validating feature.
- ▶ IBM has high performance validating parsers, but they were not generally accessible.

Each of the individual technologies had its strengths and its weaknesses:

- ▶ XML Toolkit for z/OS

The XML Toolkit for z/OS provides the SAX2 and DOM as de-facto standard XML parsing APIs. The parser is function-rich, but performance is unacceptable. It provides an event-driven programming model where the caller and parser call one another to walk through the document. As items are identified by the parser, they are passed to the appropriate event handler. This processing model makes offload impractical and there is too much entry/exit overhead.

See Figure 4-2.

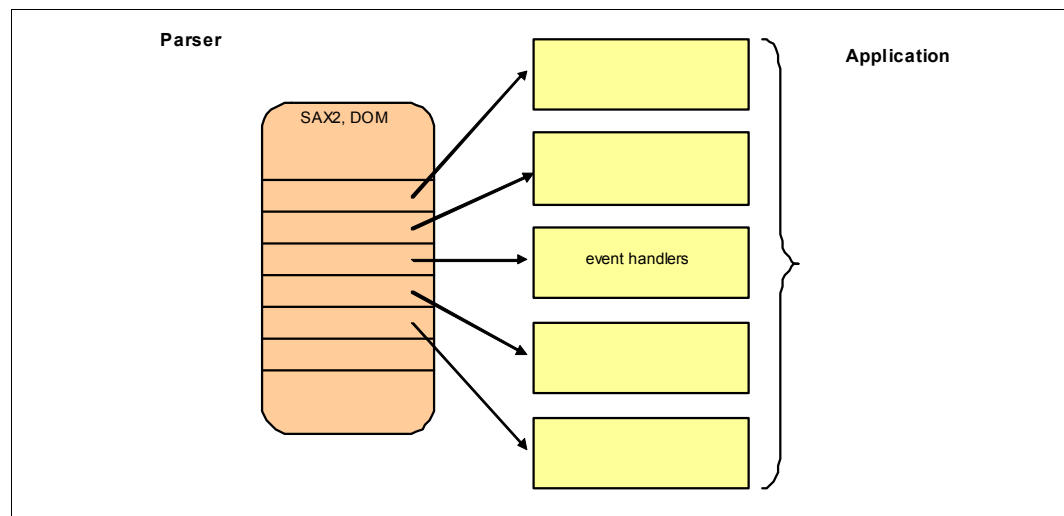


Figure 4-2 XML Toolkit for z/OS

- ▶ z/OS XML

z/OS XML is a parser for z/OS components and applications with the following key characteristics:

- Very high performance (short pathlength)
- Offloaded to specialty engines (zAAP or zIIP as appropriate)
- Limited functionality, relative to the toolkit (for example non-validating)
- Proprietary buffer-in/buffer-out interface

z/OS XML attacks the XML problem from an absolute performance standpoint as well as cost-performance through offload to assist processors.

See Figure 4-3.

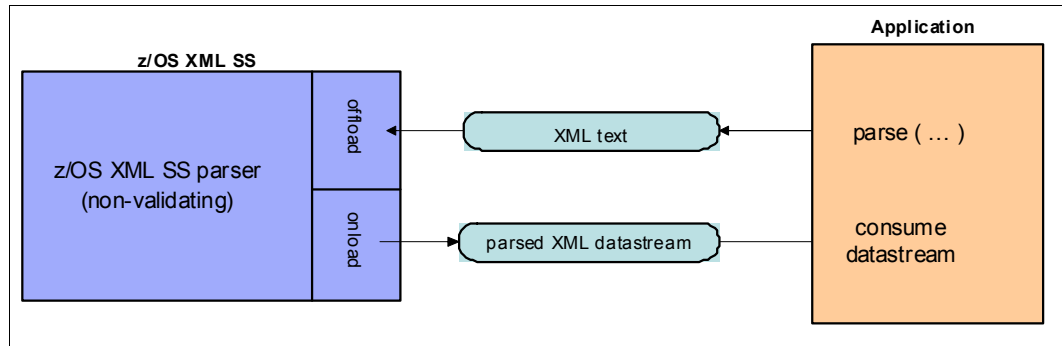


Figure 4-3 z/OS XML

► XL/XP

The XL/XP XML parser is specialized in high-performance validation and the schemas are pre-compiled to improve parse-time performance. XL/XP is a derivative of research-based, and IBM-internal XML parsers and delivers the same parsed data stream as z/OS XML.

See Figure 4-4.

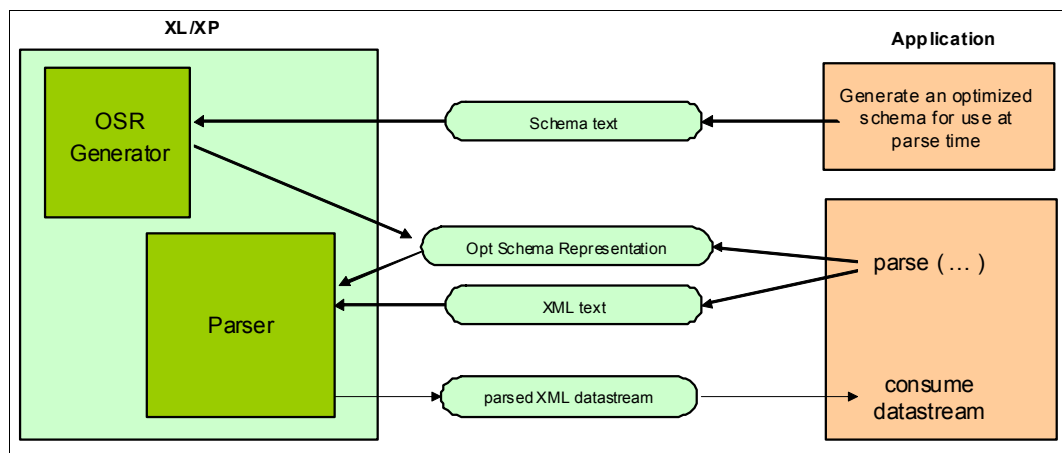


Figure 4-4 XL/XP

The requirement became to develop a high-performance, validating parsing solution with common programming interfaces that runs on assist processors.

z/OS XML was first implemented and released in V1.8 as a high-performance, system-friendly interface and a good first step in solving the “XML problem.” z/OS XML was enhanced in V1.9 with offload capabilities to a specialty engine (zAAP), a high-level language binding (C) and performance improvements. However, the validation of XML documents against a schema and a processing model that developers were already familiar with was still needed. XML processing costs were still too high, and so there was a lot of pressure to move workload off of System z. Also a system-friendly, simple call/return processing model was needed with no layers of object-oriented APIs burning CPU cycles.

All of this had evolved to a currently available solution combining the z/OS XML parser, the XL/XP validating parser and the XML Toolkit for z/OS to solve the growing issues in application development and performance on XML processing on z/OS.

Figure 4-5 provides a brief overview how the components z/OS, XLXP, and the XML Toolkit work together.

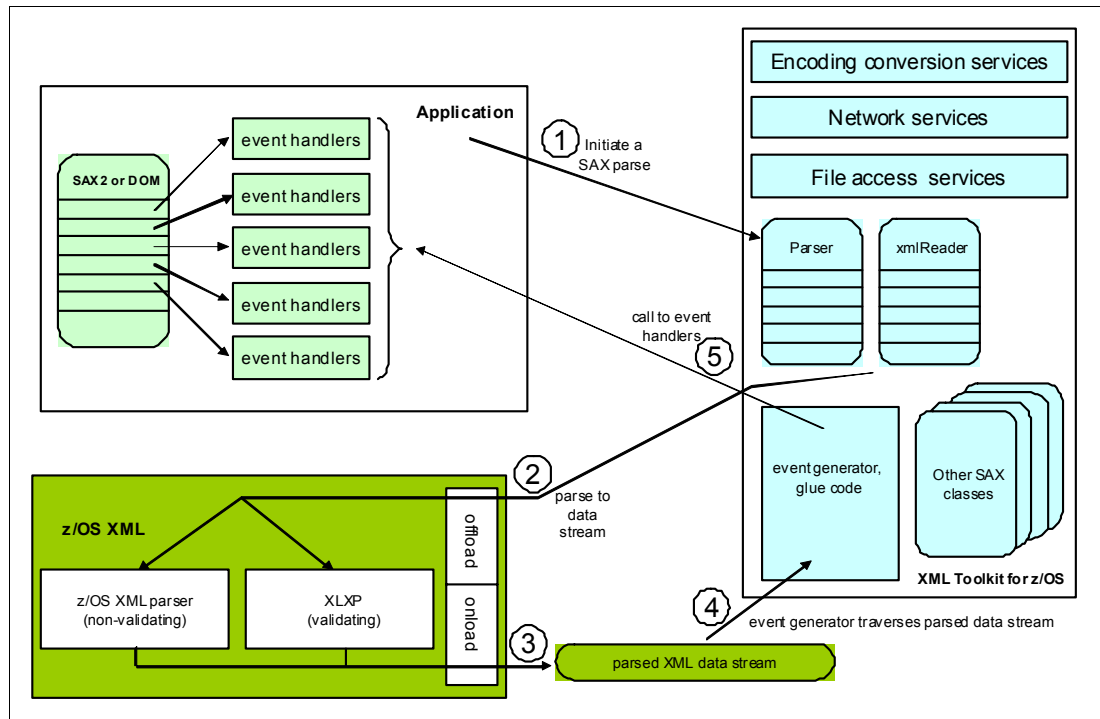


Figure 4-5 Architectural overview of using z/OS XML, XLXP, and the XML Toolkit

This solution includes the following key aspects:

- ▶ This environment is a non-Java environment. Because Java is already offloaded, any XML processing occurring within a Java application is already running on a specialty engine.
- ▶ XL/XP is an IBM internal-only parser that allows validation of XML documents against a pre-processed XML schema. It cannot be used directly at this time but is imbedded in various IBM middleware products.
- ▶ The XML Toolkit for z/OS is an open-source based collection of parser and stylesheet processors that implement the most widely accepted APIs (for example, SAX, DOM, and an XSLT processor).
- ▶ z/OS XML acts as a container and integration point for new XML processing capabilities.
- ▶ It provides an offload wrapper for both the z/OS XML non-validating parser, and XL/XP.
- ▶ It allows XL/XP validating parse functionality to be used by any software running on z/OS.
- ▶ The XML Toolkit for z/OS is enhanced to call the z/OS XML component and to request either a validating or non-validating parse.
- ▶ It provides offload capability for a large portion of the overall parse request. Some small portion of the parse request will not be offloaded, because processing still takes place within the XML Toolkit for z/OS to implement the parsing APIs it provides.
- ▶ Absolute performance improvements are being seen when running the XML Toolkit for z/OS in concert with z/OS XML. Approximately a 50% pathlength reduction for SAX parses, and 40% for DOM parse requests.

So, we use the XML Toolkit for z/OS, the z/OS XML, and the XL/XP to solve the “XML problem” with a complete set of functionality to reach high performance and cost reduction.

With the combination of these components the following problems are addressed and solved:

- ▶ Parse with schema validation
 - Integrate an existing proprietary IBM validating parser (XL/XP) with the z/OS XML parser.
 - XL/XP is already incorporated in existing IBM middleware products and z/OS XML becomes a container providing offload and an API.
- ▶ SAX2 and DOM interfaces
 - “Slide” z/OS XML under the XML Toolkit for z/OS
 - Extend existing class library to allow re-direction of parsing to z/OS XML
 - z/OS XML does the actual parse, and the XML Toolkit provides the API and external behaviors
- ▶ Additional supporting functionality
 - Offsets to locations of parsed items in the source document
 - Fully qualified names in end element records
 - Mark attribute value records generated as defaults from the DTD
 - Enhanced IPCS support
- ▶ More supported encodings
 - European country extended code pages (EBCDIC based), 19 new in total
- ▶ 100% zIIP offload
 - When caller is in enclave SRB mode, z/OS XML offloads to a zIIP
 - Will now always offload 100% of z/OS XML, and not the dialed percentage of the enclave
- ▶ New functions delivered through different mechanisms
 - z/OS releases
 - SPEs (APARs)
 - XML Toolkit for z/OS releases

DB2 is the first exploiter that currently uses both XL/XP for parsing with validation, and z/OS XML for non-validating parses in V9.

The system requirements are as follows:

- ▶ z/OS XML APARs
 - OA25903 - rollback of the validating parse function from z/OS V1.10 to V1.9
 - OA22777 - support for 19 additional European encodings (available now)
 - OA23828 - offload 100% of parser cycles on zIIP when in SRB mode
- ▶ XML Toolkit for z/OS APARs:
 - OA22700 - use z/OS XML for non-validating parse requests (for release 9 of the toolkit)

For more information, see the following resources:

- ▶ *z/OS XML System Services User's Guide and Reference*, SA23-1350
- ▶ z/OS XML System Services Web site at:
<http://www.ibm.com/servers/eserver/zseries/zos/xml/>
- ▶ XML Toolkit for z/OS Web site at:
<http://www-03.ibm.com/servers/eserver/zseries/software/xml/>
- ▶ *Using XML on z/OS and OS/390 for Application Integration*, SG24-6285
<http://www.redbooks.ibm.com/abstracts/sg246285.html?Open>
- ▶ The World Wide Web Consortium Web site at:
<http://www.w3.org/>
- ▶ Extensible Markup Language (XML) 1.0 Web site at:
<http://www.w3.org/TR/2006/REC-xml-20060816/>
- ▶ Extensible Markup Language (XML) 1.1 WQeb site at:
<http://www.w3.org/TR/2006/REC-xml11-20060816/>

4.2.6 Using pureXML capabilities in DB2 9 for z/OS

DB2 Version 9.1 for z/OS (referred to as *DB2 9* hereafter) is a hybrid data server with integrated pureXML® technology to natively manage XML data with unprecedented reliability, availability, scalability, and performance. DB2 9 pureXML exploits z/OS XML System Services for high-performance parsing with improved price and performance utilizing IBM System z specialty engines.

Figure 4-6 shows a technical overview for the XML support in DB2 9 for z/OS. You can insert your XML document and during the insert the document will be checked if well-formed and if you want you can validate the document against a registered XML schema. You can also use the functions of DB2 9 to search in the stored XML documents or generate XML documents from existing data in your database.

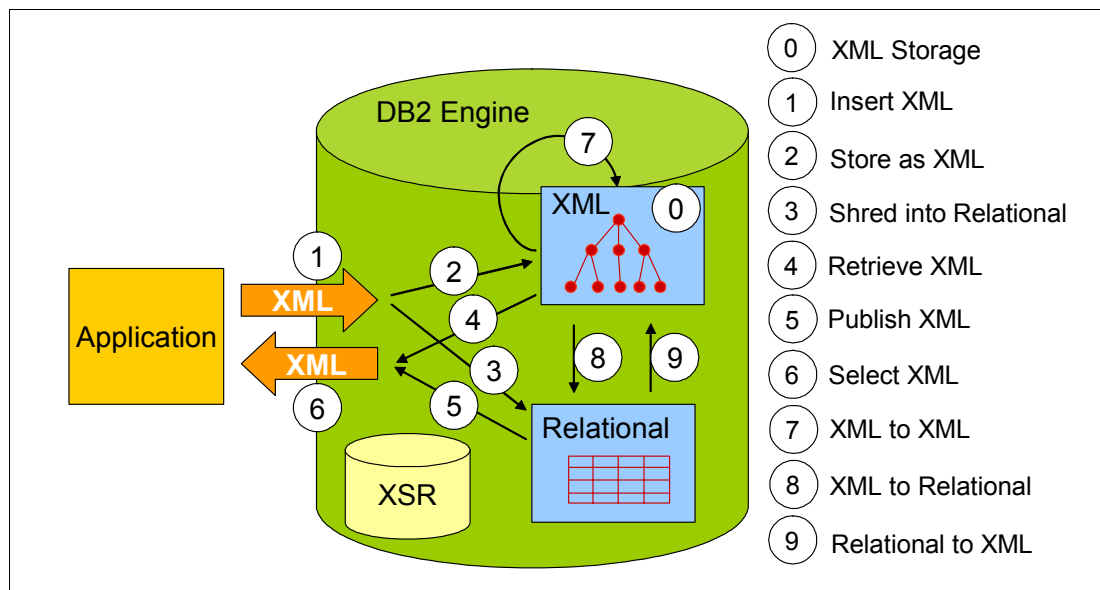


Figure 4-6 XML Support in DB2 9 for z/OS

PureXML includes the following capabilities:

- ▶ XML data type and storage techniques for efficient management of hierarchical structures inherent in XML documents.
- ▶ pureXML indexes to speed search subsets of XML documents.
- ▶ New query language support (SQL/XML and XPath) based on industry standards and new query optimization techniques.
- ▶ Industry-leading support for managing, validating, and evolving XML schemas.
- ▶ Comprehensive administrative capabilities, including DB2 utilities and tools.
- ▶ Integration with popular application programming interfaces (APIs) and development environments.
- ▶ XML shredding, publishing, and relational view facilities for working with existing relational models.
- ▶ Proven enterprise-level reliability, availability, scalability, performance, security, and maturity that users expect from DB2.

Prior to DB2 9, an XML document could have been stored in its entirety as a Character Large Object (CLOB). Alternatively, the XML Extender could have been invoked to convert and map XML data to relational data. The XML Extender used “side tables” as rudimentary indexes into the XML data and used the z/OS XML Toolkit to manipulate XML data. PureXML eliminates the need for the XML Extender and can help developers to reduce application system complexity, improve development productivity, lower maintenance costs, and develop new applications to get insight into unexploited XML data.

It is recognized that XML, which is widely adopted across industries for its flexibility and portability, is nonetheless verbose and its manipulations incur more overhead than its relational counterparts. Even though z/OS XML System Services is an efficient system level XML parser on z/OS, additional general CPU cost reduction is possible. With the proper software and hardware, local DB2 calls to z/OS XML System Services are redirected to a IBM System z Application Assist Processor (zAAP) specialty engine. Included are invocations from the attachments that run in TCB mode. DB2 already has functions that can be redirected to the System z Integrated Information Processor (zIIP) specialty engine. For a workload already on a specialty engine, the intent is to continue to run it. For a workload not running on a specialty engine, the intent is to allow the DB2 XML processing to run on a specialty engine to mitigate its cost.

There are differences between DRDA redirection and z/OS XML System Services redirection. For XML, DB2 9 XML inserts, updates or the LOAD utility invoke z/OS XML System Services and these system invocations are the only operations eligible for zAAP. A workload arriving through DRDA can have a portion of its entire XML workload redirected to the zIIP, including select, insert, update, delete, and query. A DRDA workload executes in Enclave SRB mode and none of it is zAAP-eligible.

Even though a local XML workload is 100% zAAP-eligible for its z/OS XML System Services invocation, only the z/OS XML System Services portion, not the entire execution of the XML insert/update/LOAD is redirected. DB2 actions prior to and after that call are not zAAP-eligible.

You can find more detailed information about pureXML capabilities in the following resources:

- ▶ *DB2 Version 9.1 for z/OS XML Guide*, SC18-9858, which is available at:
<http://publib.boulder.ibm.com/epubs/pdf/dsnxgk10.pdf>
- ▶ *DB2 9 for z/OS: Deploying SOA Solutions*, SG24-7663, which is available at:
<http://www.redbooks.ibm.com/abstracts/sg247663.html>

4.2.7 Summary

There are lots of possibilities to work with XML under z/OS. IBM System z offers excellent performance with z/OS XML System Services. z/OS XML and its high performance together with zAAP and zIIP speciality processors is a very cost efficient solution.

4.3 Implementing new functionality in C/C++

You might have requirements that are not easy or even impossible to implement with COBOL and PL/I but for which C/C++ is an option. C and C++ are comprehensive and powerful programming languages that allow you to produce high-performing programs providing optimized business software solutions. If you have the requirement to integrate C/C++ functions in your traditional batch environment, you can use Inter Language Communication (ILC) between C and C++ programs, and between C or C++ programs and Assembler, COBOL, PL/I or FORTRAN programs.

Inter Language Communication (ILC) under the Language Environment® covers COBOL, PL/I and C/C++, but Assembler is not a language in terms of ILC.

Java is not covered in ILC, as Java is an object oriented model providing interoperability through CLASS inheritance. Java runs under UNIX System Services and can be invoked in different ways, such as by invoking a BPXPATCH or BPXBATSL job (see 7.1, “Running Java with the BPXBATCH or BPXBATSL utilities” on page 78), or by CICS (see 6.1, “Java in CICS” on page 58) or IMS (see 6.2, “Java in IMS” on page 58).

ILC focuses on direct program CALLs between modules written in a procedural language, such as COBOL and PL/I. Enterprise COBOL and Enterprise PL/I added a lot of features (compared to what was in the earlier compilers) to make it easy to call C functions from COBOL or PL/I. In fact, it is possible to call any C function from COBOL or PL/I (and because there are also now many useful C functions on the mainframe, this is important). Some reasons for calling C/C++ modules in traditional batch applications include:

- ▶ When you need to implement a function in an existing batch program that is already available as a C/C++ module. You can then simply call that existing C/C++ module from your existing COBOL or PL/I program using ILC.
- ▶ When COBOL or PL/I do not provide the syntax and APIs to implement the function you require. An examples are using communication protocols, such as TCP/IP.
- ▶ When you do not have skills available (anymore) to continue coding in COBOL or PL/I. In that case you can consider to gradually move to C/C++ by implementing new functionality in C/C++ and keeping existing functionality in COBOL or PL/I. ILC is available to integrate.

- ▶ When you need to “bridge” from COBOL or PL/I to Java or vice versa. In that case you can use Java Native Interface (JNI) to bridge between Java and C/C++ and ILC to bridge between C/C++ and COBOL or PL/I.
- ▶ When you need to “plug in” a vendor product into your existing COBOL or PL/I batch environment. A vendor product might provide modules in C/C++ that you need to integrate in existing COBOL or PL/I batch programs, such as for printing purposes.

Note that mixing languages should not be a goal by itself, because it increases complexity and dependency on multiple programming skills. Consider mixing languages and using ILC only if there is no other option. The good thing is that with the Language Environment and ILC, z/OS provides one virtual environment for multiple programming languages.

Note: The IBM C compilers and the IBM COBOL and PL/I compilers use the same default linkage.



Introduction to Java on z/OS

Before we discuss the various scenarios with Java in subsequent chapters, we first provide an overview of Java on z/OS in this chapter. This chapter includes the following topics:

- ▶ The basics of Java
- ▶ Special Java APIs for batch processing on z/OS
- ▶ Data access with Java on z/OS
- ▶ Encoding issues
- ▶ Java Interoperability with COBOL and PL/I

5.1 The basics of Java

As traditional languages such as COBOL, PL/I, and Assembler sometimes lack functionality to serve certain new requirements (see Part 2, “Serving new functional requirements in z/OS batch” on page 31), Java can be a very interesting option to solve many of those issues.

Because Java is platform independent, it is very easy to reuse components on z/OS that originally have been written for distributed platforms. This includes Java based Open Source components as well as Java based functions and frameworks provided by ISVs. As of today, a huge variety of different Java libraries for all kind of purposes can be found and reused for your own batch processes.

Figure 5-1 shows how the platform independency is achieved by the different available IBM Java Virtual Machines (JVMs).

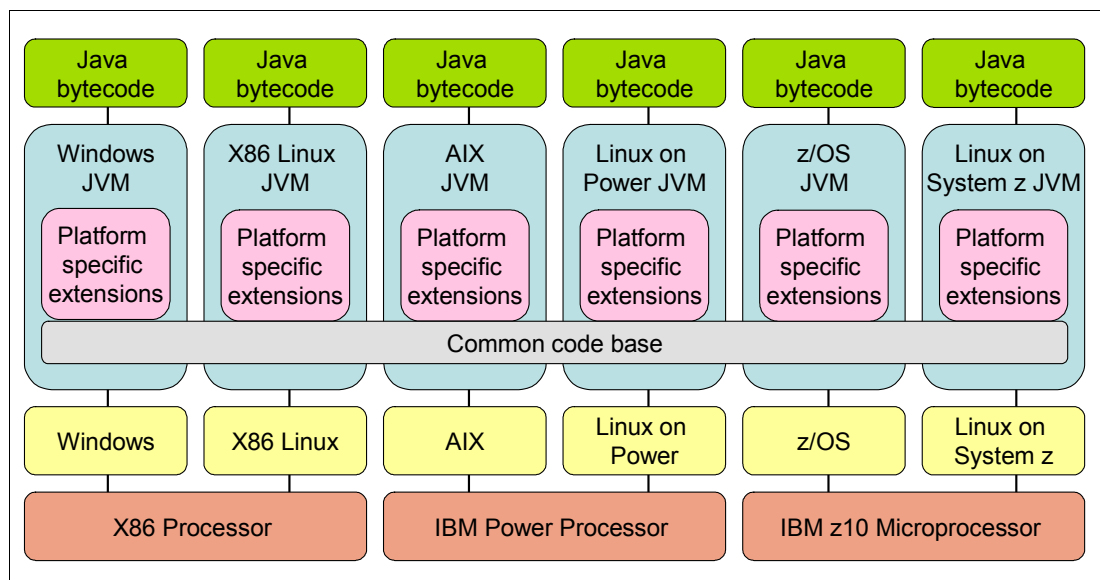


Figure 5-1 Platform independency with IBM JVMs

For different kinds of operating systems and their underlying hardware, a special IBM JVM is available. This JVM abstracts the underlying components of the OS and hardware by translating the compiled Java bytecode to OS specific instructions. As a result, every Java program can be run on all of the mentioned platforms without recompiling it.

Note: Java bytecode is not bound to the IBM JVM. It also runs in other JVMs that comply to the Java standard.

All the IBM JVMs are built on a common code base with platform specific extensions. This allows a common handling of IBM JVMs on all platforms.

In addition to the aspect of code reuse for batch modernization on z/OS, Java also offers the following advantages on z/OS:

- ▶ Easy programming model through object orientation
- ▶ Java skills are more readily available in the market
- ▶ Rich in functionality
- ▶ Very good development tools on Eclipse base available

- ▶ Can be run on System z10 Application Assist Processor (zAAP) to reduce cost
- ▶ Data proximity

Due to local connectors to data back-end systems, you can achieve high performance. For example, the JDBC Type II driver for z/OS allows Java z/OS programs to interact with DB2 z/OS based on a cross memory connection. In contrast to the TCP/IP based JDBC Type IV driver, this eliminates the cost for the network connection. Especially in a batch environment where typically much data is involved, this can be a competitive advantage.
- ▶ Multithreading support

In contrast to traditional z/OS programming languages, Java supports multithreading, which can help to achieve higher efficiency and better performance as multiple threads can run in parallel on the processors available.

These advantages can be used in nearly all run times on z/OS:

- ▶ CICS
- ▶ IMS
- ▶ DB2
- ▶ WebSphere Application Server
- ▶ WebSphere stack solutions running on WebSphere Application Server, such as WebSphere Process Server, and WebSphere Portal
- ▶ Java stand-alone under z/OS

Thus, you can use Java in batch nearly wherever you want. We discuss these run times and their usability for Java batch in subsequent chapters in more detail.

For further information about Java on z/OS, see:

<http://www.ibm.com/servers/eserver/zseries/software/java/>

5.2 Special Java APIs for batch processing on z/OS

IBM uses a common code base for the JVM on all platforms plus additional platform specific extensions. One of those extensions on z/OS is specific Java APIs for batch processing, including:

- ▶ MVS data set and VSAM access to interact with z/OS specific data
- ▶ Condition code passing for integration of Java batch jobs into z/OS job nets
- ▶ z/OS Catalog search
- ▶ Interaction with the MVS Console
- ▶ Conversion of COBOL/ASM data types to Java types
- ▶ Invoking DFSORT to effectively sort data (see 23.5, “Invoking DFSORT from Java with JZOS” on page 417 for further details)
- ▶ Access to z/OS Access Method Services (IDCAMS)
- ▶ RACF APIs to integrate Java into the z/OS security model
- ▶ Writing of Logstreams (for example as an Appender to LOG4J)
- ▶ Submission of Jobs from Java (see 14.1, “Job submission with native Java technology” on page 224 for further details)

For a description of the function, see:

<http://www.ibm.com/servers/eserver/zseries/software/java/products/jzos/overview.html>

For a description of the `com.ibm.jzos` packages, see the full Javadoc at:

<http://www.ibm.com/developerworks/java/zos/javadoc/jzos/index.html>

Note: The javadoc reflects the current version of the IBM Java SDK for z/OS. If some of these functions are not included in your SDK, update it to the latest version.

You can find samples about how to implement these APIs in the following resources:

- ▶ In the JZOS Cookbook at:

<http://www.alphaworks.ibm.com/tech/zosjavabatchtk/download>

- ▶ On the IBM Web site:

<http://www.ibm.com/servers/eserver/zseries/software/java/products/jzos/jzossamp.html>

5.3 Data access with Java on z/OS

Because data access is a very important aspect of batch, we discuss this topic in further detail here. Data on z/OS typically not only resides in UNIX System Services based files or in DB2 on z/OS. Instead, much of z/OS data is very often stored in sequential files, partitioned data sets and VSAM. To address the needs to access these z/OS specific data stores, Java APIs are available.

Data access support in Java on z/OS is provided by a complementary set of functions of the JZOS toolkit library and the Java Record I/O library (JRIO). Together, these libraries allow Java applications access to mainframe file systems that ordinary Java APIs do not support.

The JZOS toolkit provides thin wrappers for z/OS C/C++ Library functions which can be used to access MVS data sets. For a full discussion on these C/C++ functions, refer to the following IBM C/C++ publications:

- ▶ *z/OS C/C++ Run-Time Library Reference*, SA22-7821
- ▶ *z/OS C/C++ Programming Guide*, SC09-4765

Using the JZOS toolkit, Java programs can access any MVS data sets supported by the C/C++ library, including:

- ▶ Partitioned Data Set (PDS)A
- ▶ Partitioned Data Set Extended (PDSE)
- ▶ Sequential Files
- ▶ Virtual Sequential Access File (VSAM) of the type KSDS, RRDS, or ESDS

The z/OS C/C++ library supports several models of I/O when using MVS data sets:

Record mode Each read or write processes a single record of a data set.

Stream mode Data set records are presented as a stream of bytes. Each read or write reads some portion of those bytes, irrespective of record boundaries. Stream mode is further distinguished by two types:

Text (stream) mode

Data set records are converted to a stream of bytes and a “new line” record delimiter is placed in the stream between records after trailing blanks are removed.

Binary (stream) mode

Data set records are placed in the stream as is.

For example, if we define an INPUT and OUTPUT DD statement in a JCL that is calling a Java job with the JZOS batch launcher (see 7.2, “Running Java with JZOS” on page 79 later on), we can copy the INPUT data set to the OUTPUT data set in binary record mode, as shown in Example 5-1:

Example 5-1 Copying an MVS data set in binary record mode using ZFile

```
ZFile zFileIn = new ZFile("//DD:INPUT", "rb,type=record,noseek");
ZFile zFileOut = new ZFile("//DD:OUTPUT", "wb,type=record,noseek");
try {
    byte[] recBuf = new byte[zFileIn.getLrecl()];
    int nRead;
    while((nRead = zFileIn.read(recBuf)) >= 0) {
        zFileOut.write(recBuf, 0, nRead);
    };
} finally {
    zFileIn.close();
    zFileOut.close();
}
```

In contrast to binary record mode, Example 5-2 shows how to read an MVS data set in text stream mode to put it to SYSOUT. This example is reading all data defined by a DD statement called INPUT and writes it to SYSOUT.

Example 5-2 Reading an MVS data set in text stream mode using ZFile

```
ZFile zFile = new ZFile("//DD:INPUT", "rt");
try {
    String enc = ZUtil.getDefaultPlatformEncoding();
    InputStream is = zFile.getInputStream();
    BufferedReader rdr = new BufferedReader(
        new InputStreamReader(is, enc));

    String line;
    while ((line = rdr.readLine()) != null) {
        System.out.println(line);
    };
} finally {
    zFile.close();
}
```

Instead of the previous example, we can also use the JZOS FileFactory to perform the same function. This one has the benefit of being simpler and more platform portable as shown in Example 5-3.

Example 5-3 Reading text from MVS data set using FileFactory

```
BufferedReader rdr = FileFactory.newBufferedReader("//DD:INPUT");
try {
    String line;
    while ((line = rdr.readLine()) != null) {
        System.out.println(line);
    }
} finally {
    rdr.close();
}
```

In addition to binary record mode and text stream mode, the third way to access MVS data sets is binary stream mode. For example, we can define the following DD statement in a JCL:

```
//INPUT DD DSN=STRAUER.XML.IN.BIN,DISP=SHR
```

This INPUT data set contains some XML data which we parse later on with the standard Java SAXParser as shown in Example 5-4.

Example 5-4 Parsing XML from an MVS data set using ZFile binary stream mode

```
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();
MyDocumentHandler handler = new MyDocumentHandler();
ZFile zFile = new ZFile("//DD:INPUT", "rb");
try {
    parser.parse(zFile.getInputStream(), handler);
} finally {
    zFile.close();
}
```

Note: Because the SAXParser is standard Java, we do not show the source code here.

The JZOS toolkit APIs should be used for accessing all types of MVS data sets and the standard `java.io` Java library should be used for accessing UNIX System Services files (HFS/zFS).

For more information about accessing MVS data sets with JZOS, refer to the JZOS Cookbook at:

<http://www.alphaworks.ibm.com/tech/zosjavabatchtk/download>

5.3.1 Summary

All of these z/OS Java APIs allow a very efficient integration of Java into the z/OS batch environment because they provide the ability to use a lot of commonly used functions of other z/OS programming languages also to be used in Java. Combined with the additional capabilities of Java, such as object orientation or PDF creation, Java can be a very interesting option to serve new functional requirements in a z/OS batch environment.

5.4 Encoding issues

Under z/OS, the default code page for UNIX System Services is IBM-1047. As the JVM is using UNIX System Services, I/O operations are per default EBCDIC. Sometimes, this can lead to problems:

- ▶ If we are developing all Java code on distributed platforms, we have to be very careful about the encoding of some files. Some files, such as the `*.class` files, have to be uploaded to z/OS in binary mode through FTP. Other files, such as `*.properties` files, in contrast, must be converted to EBCDIC by using ASCII mode in FTP.
- ▶ Sometimes, external Java libraries from third parties contain hard coded ASCII characters in the code, which often also causes problems.

The easiest way to get around this is to use the following parameter when launching the JVM, which causes the z/OS JVM to read/write everything in ASCII and solves the problems:

```
-Dfile.encoding=ISO8859-1
```

Note: WebSphere Application Server for z/OS is using ASCII encoding as a default. Therefore, encoding should not be a problem in WebSphere Application Server.

5.5 Java Interoperability with COBOL and PL/I

Another important aspect of using Java in an existing z/OS batch environment is interoperability between the COBOL and PL/I and the Java language.

Basically, there are two ways to achieve interoperability between Java and other languages:

- ▶ Using functionalities of the runtime environment, such as CICS, IMS, DB2, WebSphere Application Server, or JES.
- ▶ Creating direct language calls.

The first option depends heavily on the selected run time. We provide further details about this option in Chapter 6, “Implement new functionality using Java in traditional containers” on page 57 and Chapter 8, “Implement new functionality using Java in WebSphere XD Compute Grid” on page 93 for CICS/IMS/DB2 and WebSphere respectively.

In contrast, the second option is not bound to a runtime environment. We discuss details about this option in the next two sections. Also refer to 4.3, “Implementing new functionality in C/C++” on page 47.

5.5.1 Enterprise COBOL

Enterprise COBOL offers the capabilities to invoke Java methods from COBOL and the other way around to invoke COBOL programs from Java. When you are using object-oriented (OO) COBOL, Enterprise COBOL allows this interaction with Java based on the Java Native Interface (JNI) technology.

Note: In Enterprise COBOL, the JVM is started only one time when OO COBOL is initiated. Thus, there is no additional cost when you call Java methods from COBOL after this initiation.

You can find further information about how to use COBOL and Java interoperability *Enterprise COBOL for z/OS Programming Guide Version 4 Release 1*, SC23-8529.

Another important aspect of COBOL and Java interaction is data conversion between Java and COBOL. Many data sets on z/OS systems contain records described by COBOL copybooks. These records contain fields that conform to the set of valid COBOL types, such as Alpha, Alpha-numeric, unscaled numeric, decimal (packed or zoned) and binary (COMP-5). As it is desirable for Java programs to read and process this kind of data set, writing the code to convert from the COBOL to Java datatypes can be tricky. Therefore, the IBM Java SDK for z/OS offers a complete set of field converter types that help to convert COBOL/ASM types to Java types.

5.5.2 Enterprise PL/I

Enterprise PL/I also allows you to invoke Java methods and Java can invoke PL/I programs. Again, this is based on the JNI technology.

Note: Each time you invoke a Java method from PL/I, it creates a new JVM. Depending on the amount of Java code to be processed, this JVM startup can lead to a relative overhead. In contrast, if you are invoking PL/I programs from Java, the JVM is only started once.

You can find further information about PL/I and Java interoperability *Enterprise PL/I for z/OS Programming Guide Version 3 Release 8, SC27-1457*.



Implement new functionality using Java in traditional containers

In this chapter, we explain the different capabilities of using Java based batch in the traditional runtime environments such as CICS, IMS, and DB2 for z/OS. This chapter includes the following topics:

- ▶ Java in CICS
- ▶ Java in IMS
- ▶ Java in DB2 for z/OS

6.1 Java in CICS

Because CICS is not intended as full batch container, we do not discuss CICS in this book. Nevertheless, you might have situations where you want to implement batch processing in CICS and also consider Java as a programming language. Refer to the extensive resources available for CICS and *Java Application Development for CICS*, SG24-5275, in particular.

6.2 Java in IMS

Using Java in IMS can be an attractive option for batch processing because it combines the benefits of Java's rich functionality and additional reliability provided by IMS through specific methods for checkpointing, restarting, and rollback.

IMS Java batch applications can access IMS databases, VSAM files, GSAM files or DB2 databases. For DB2 it might be important to know, that the connection to DB2 that is established by the IMS Java batch program accessing DB2 can be reused until the program finishes its work.

This chapter explains how Java can be used in an IMS environment for batch processing. 6.2.1, "Introduction to IMS databases" on page 58 provides a brief introduction on IMS databases and 6.2.3, "Java applications in IMS" on page 60 discusses how to use Java in an IMS batch environment.

6.2.1 Introduction to IMS databases

In the following section, we give a brief overview of IMS databases. If you are familiar with IMS principles, you can skip this section.

Hierarchical databases

An IMS Database (IMS DB) is made up of records that are composed like trees (as opposed to relational database systems, for example DB2, that store data in tables). Certain types of data might call for a specific storage structure. For example, a phone directory matching names and phone numbers fits well into a table, while a log of customer orders with several articles per order likes to dwell in a tree. Figure 6-1 illustrates the concept of storing data in a hierarchical tree.

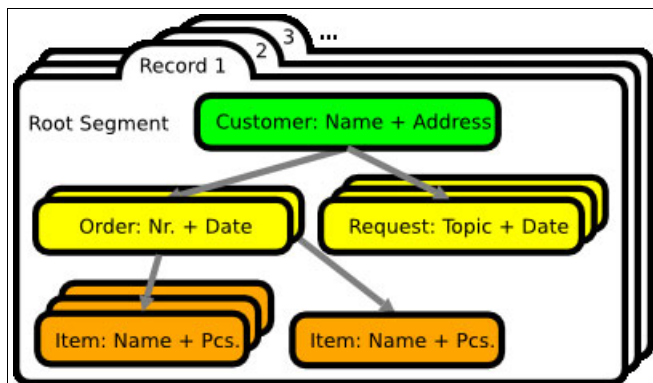


Figure 6-1 Hierarchical database

The exemplary customer database shown in Figure 6-1 stores the orders and support requests of all customers of a company. For every customer, there is one record, each with one root segment. The root segment stores the name and the address of the respective customer. Every record can contain several copies of the dependent segments order and request according to the number of orders and requests that the customer placed. For every item of an order, the product name and the number of items is noted in an extra segment. These structures allow for fast navigation through the data and to find a position with little effort.

Furthermore, the structures of IMS databases and XML documents are very similar. Special applications can store XML efficiently in new or already existing IMS databases and retrieve XML and keep the hierarchical organization of the data all the way through. In addition to using Java, using XML can be a very good enrichment of existing IMS applications.

IMS data can be accessed by navigating the hierarchical structure. The original method for accessing this data was Data Language/I (DL/I), which is roughly equivalent to SQL. DL/I uses trees instead of tables and returns single segments instead of result sets. There is also a hierarchical database interface for Java that encapsulates DL/I calls in Java methods.

To use both hierarchical and relational access to IMS databases, you must either create a Java representation of the IMS segment hierarchy and fields or use the DLIModelUtility plug-in to generate the IMSDatabaseView object from IMS DBDs, PSBs, and COBOL Copybooks.

Relational access to hierarchical data

In addition to DL/I, the same data can be accessed using Java DataBase Connectivity (JDBC). Because JDBC does not operate on trees, but on tables, the data in the tree must be mapped to a table. In the previous example, every instance of *Item* (the lowest mapped segment) results in a separate row. If several instances of *Item* have the same instance of *Order* as a parent, they share the same data for *Order_No* and *Order_Date* in the resulting table. Table 6-1 shows a mapping between hierarchical data and relational data.

Note: IMS Version 11 provides an implicit mapping called *virtual private keys*. Using this concept, an SQL INSERT no longer requires a WHERE clause (nonstandard SQL) to determine the hierarchical position of the new segment. For more information, see the following resource:

ftp://ftp.software.ibm.com/software/os/systemz/pdf/July_14_Telecon_Get_Smart_IMS_Application_with_Cobol.pdf

Table 6-1 Relational representation of hierarchical data

Cust-Name	Cust_Address	Order_No	Order_Date	Item_Name	Item_Pcs
Cust 1	Street 1	40	2009-11-14	Paper	2
Cust 1	Street 1	40	2009-11-14	Pencil	1
Cust 1	Street 1	41	2009-11-18	Eraser	1
Cust 1	Street 1	44	2009-12-04	Envelope	3
Cust 1	Street 1	44	2009-12-04	Paper	4
Cust 2	Street 8	43	2009-12-03	Paper	20
Cust 3	Street 4	42	2009-12-01	Pencil	10
Cust 3	Street 4	42	2009-12-01	Paper	8

6.2.2 Object mapped access to hierarchical data

With Java, a very popular way of persisting Java objects is to use an *object-relational mapper* to store the data that is in an Java object instance. See the following resources for examples:

- ▶ Hibernate

<http://www.hibernate.org>

- ▶ The Sun Java Persistence API (JPA)

<http://java.sun.com/javaee/technologies/persistence.jsp>

Currently, this feature is not available. You can use DB2 within IMS Java to persist objects into DB2 databases and tables (for example, by using Hibernate as a Java persistence framework).

6.2.3 Java applications in IMS

As described previously, you can access IMS data using two methods. Both access methods, JDBC or the IMS hierarchical database interface for Java, require the IMS Base API for Java to convert the calls to DL/I calls, as depicted in Figure 6-2.

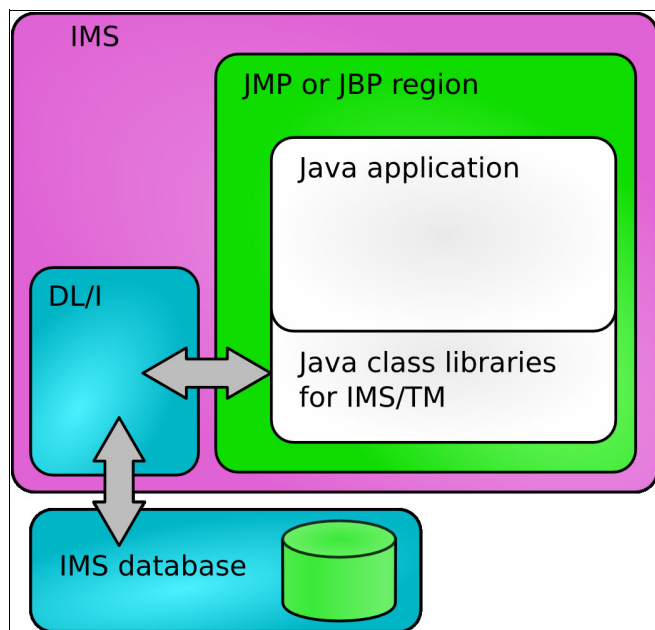


Figure 6-2 Java applications in IMS

Java Message Processing (JMP) and Java Batch Processing (JBP) applications can access DB2 for z/OS databases in addition to IMS databases. For more details about Java in IMS, see *IMS Version 7 Java Update*, SG24-6536 and the IMS Information Center.

IMS Java batch processing

Each IMS application program runs in an IMS region. Traditionally, there are the two region types:

- ▶ MPP for online processing
- ▶ BMP for batch processing

Two additional region types were introduced to accommodate IMS Java application programs:

- ▶ JMP for online processing
- ▶ JBP for batch processing

Both make use of a Java Virtual Machine (JVM).

Batch-type processing with Java in an (online) IMS environment can be achieved using a JBP region. Programs running inside JBP regions can access the IMS message queues for output but not for input. They can be started using JCL (or using TSO).

If the application does not need to run in an IMS dependent region, a JBP is not necessary. Nevertheless, running an application in the JBP container provides transactional services and checkpoint restart support.

Starting a JBP application

JBP applications are either started from TSO or by submitting a job with JCL. Example 6-1 shows a sample JCL. For a description of the parameters, see:

http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.im.s10.doc.sdr/ims_dfjsbp_procedure.htm

Example 6-1 Sample JCL to start a JBP

```
//JBPJOB JOB 1,IMS,MSGLEVEL=1,PRTY=11,CLASS=K,MSGCLASS=A,REGION=56K
// EXEC DFSJBP,
// IMSID=
// PROC MBR=TEMPNAME,PSB=,JVMOPMAS=,OUT=,
// OPT=N,SPIE=0,TEST=0,DIRCA=000,
// STIMER=,CKPTID=,PARDLI=,
// CPUTIME=,NBA=,OBA=,IMSID=,
// PREINIT=,RGN=56K,SOUT=A,
// SYS2=,ALTID=,APARM=,ENVIRON=,LOCKMAX=,
// XPLINK=N
// *
//JBPRGN EXEC PGM=DFSRR00,REGION=&RGN,
// PARM=(JBP,&MBR,&PSB,&JVMOPMAS,&OUT,
// &OPT&SPIE&TEST&DIRCA,
// &STIMER,&CKPTID,&PARDLI,&CPUTIME,
// &NBA,&OBA,&IMSID,
// &PREINIT,&ALTID,
// '&APARM',&ENVIRON,&LOCKMAX,
// &XPLINK)
//STEPLIB DD DSN=IMS.&SYS2.SDFSJLIB,DISP=SHR
// DD DSN=IMS.&SYS2.SDFSRESL,DISP=SHR
// DD DSN=IMS.&SYS2.PGMLIB,DISP=SHR
// DD DSN=CEE.SCEERUN,DISP=SHR
// DD DSN=SYS1.CSSLIB,DISP=SHR
//PROCLIB DD DSN=IMS.&SYS2.PROCLIB,DISP=SHR
//SYSUDUMP DD SYSOUT=&SOUT,
// DCB=(LRECL=121,RECFM=VBA,BLKSIZE=3129),
// SPACE=(125,(2500,100),RLSE,,ROUND)
```

Data safety: IMS checkpoints

Data changes are bundled into transactions, which are either executed completely or not at all to ensure data integrity. For example, a customer's order and the amount that is billed is entered simultaneously. Thus, either the order is accepted and the customer is billed or the order is rejected and the customer is not billed.

In batch processing, transactions are established by checkpoints. IMS commits all data changes since the beginning of the batch process or since the last checkpoint when `checkpoint()` is called.

Should a batch process not have completed its work (abnormal end), you can restart it using `restart()`.

The primary methods for checkpoint and restart are:

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

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

Example

Example 6-2 shows a JBP application that performs the following tasks:

- ▶ Connecting to a database
- ▶ Making a restart call
- ▶ Performing database processing
- ▶ Periodically making checkpoints
- ▶ Issuing a final checkpoint
- ▶ Disconnecting from the database at the end of the program

Example 6-2 JBP application

```
public static void main(String args[]) {  
  
    conn = DriverManager.getConnection(...);    //Establish DB connection  
  
    IMSTransaction.getTransaction().restart(); //Restart application  
                                              //after abend from last  
                                              //checkpoint  
  
    repeat {  
  
        repeat {  
            results=statement.executeQuery(...); //Perform DB processing  
            ...  
            MessageQueue.insertMessage(...);    //Send output messages  
            ...  
        }  
  
        IMSTransaction.getTransaction().checkpoint(); //Periodic checkpoints  
                                                    // divide work  
    }  
  
    conn.close();                               //Close DB connection  
    return;  
}
```

On an initial application start, the `IMSTransaction().restart()` method notifies IMS that symbolic checkpoint and restart is to be enabled for the application. The application then issues periodic `IMSTransaction().checkpoint()` calls to take checkpoints. The `IMSTransaction().checkpoint()` method allows the application to provide a `com.ibm.ims.application.SaveArea` object that contains one or more other application Java objects whose state is to be saved with the checkpoint. Should the application fail, the state of these objects can then be restored during restart, which is different from the traditional IMS batch that allows to store deserialized Java objects in the checkpoint area of an IMS JBP instead of copybooks or DSECTs.

If a restart is required, it is initiated by the application. The checkpoint ID is provided either with the `IMSTransaction().restart()` call (similarly to providing the ID to the `XRST` call in IMS) or within the `CKPTID=` parameter of the JBP region JCL. The `restart()` method returns a `SaveArea` object that contains the application objects in the same order in which they were originally checkpointed.

If, for whatever reason, the data changes that the application made are not written to the database, `rollback()` is called instead of `checkpoint()`, as follows:

```
IMSTransaction.getTransaction().rollback(); //Roll back DB updates
```

For more detailed information about the programming model for symbolic checkpoint and restart, see JBP application with symbolic checkpoint and restart at:

<http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/topic/com.ibm.ims10.doc.apg/ie0j1bas999243.htm#symbckpt>

If you need transactionality, this IMS feature can be a big advantage compared to stand-alone Java batch applications, which we discuss in Chapter 7, “Implement new functionality using stand-alone Java” on page 77.

Java interoperability with other languages

In addition to the common ways of communication, IMS Java applications can interact with applications written in other languages using IMS queues such as C. C can then be used to call other modules, for example Assembler. Special cases in this area are COBOL and PL/I, which have compiler and API provided support to interoperate with Java.

For COBOL, procedural mixed mode COBOL programs in addition to object-oriented (OO) COBOL applications can run in the same regions as Java applications and interact with them using JNI. JNI is used to call the C-language DL/I routines. The COBOL compiler generates Java classes which use the compiler-generated native methods. Thus, COBOL procedures and methods can call Java methods and OO COBOL methods can be called by Java methods (see 5.5, “Java Interoperability with COBOL and PL/I” on page 55). Also, an OO COBOL method can call a procedural COBOL module.

IMS Java development

All current development environments for the mainframe support IMS Java applications.

As an example, development is possible with pure Eclipse, but you need to download the `imsjava` JAR files. These files are available from the HFS path where the IMS Java HFS was installed. In addition, you can use Ant scripts and the FTP Spool API that are available with z/OS to automatically create a JAR file, upload it to the z/OS host, execute the IMS Java Batch, and return the output job log into the Eclipse workspace.

If you want to write and test Java applications that access IMS data, you can use the sample application for IMS solutions for Java development. You can download this sample application from the IMS Web site. It provides a working sample that is run against the Dealership

database and helps application developers build Java applications. Jobs to build the Dealership database comes with the IMS installation. For more information about the IVP jobs and tasks for the IMS sample application for XQuery support in IMS, see the *IMS Version 10: Installation Guide* at:

<http://www-306.ibm.com/software/data/ims/imsjava/>

For details about the classes that you use to develop a JBP application, see the Java API specification for IMS at:

<http://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/index.jsp?topic=/com.ibm.im.s10.doc.apr/ims10javadoc.htm>

6.3 Java in DB2 for z/OS

In addition to an IMS BMP, you can use DB2 for z/OS as a Java batch run time by using DB2 Java stored procedures. If Java programs are executed in a DB2 stored procedure, they can benefit from DB2 functionalities. From a batch point of view, these benefits include:

- ▶ Transactional processing

Because stored procedures are integrated in the DB2 environment, they can fully participate in the current transaction flow, thus allowing a Java program to use the same transaction context as the program that is calling the stored procedure (including two-phase commit support).

- ▶ Java Virtual Machine (JVM) instantiated only once

Compared to stand-alone Java Batch (see Chapter 7, “Implement new functionality using stand-alone Java” on page 77), the JVM for a DB2 Java stored procedure is started only once when it is called for first time. Because DB2 reuses the started JVM, subsequent calls to a stored procedure can be processed immediately, which can become relevant when stored procedures are called many times repeatedly, as they are typically in an OLTP environment. However, in the context of batch processing, it is more likely that a batch stored procedure gets called just once and the JVM startup cost is not that relevant.

- ▶ Reuse of DB2 thread.

To run a stored procedure, a process must first connect to DB2 and then issue a CALL statement. Thus, stored procedures use the same physical DB2 thread for processing SQL statements as the process that called the stored procedure, and there is no additional cost for building DB2 connections within the stored procedure.

- ▶ Scalability

With respect to stored procedures, scalability is ensured by the integration of DB2 for z/OS with other mainframe components.

You can run several stored procedures at the same time within one single address space. Furthermore, depending on the workload, the integrated Workload Manager (WLM) can start additional address spaces, if required, which ensures that stored procedures scale and are particularly well suited to process high workloads.

6.3.1 Java interoperability with other languages

Another important aspect of using Java in DB2 stored procedures is interoperability with other programming languages. Because the Java is running in a DB2 stored procedure address space, you can call the Java application from anywhere by just connecting to DB2 and issue an SQL CALL statement. This CALL statement is part of the SQL standard. Therefore, any

program (written in any language) that is capable of connecting to DB2 and using SQL is suitable to invoke Java batch within DB2.

6.3.2 Configuring the system environment

In this section, we describe the relevant steps to set up the system to run Java stored procedures in DB2.

Note: You can find detailed information about setting up and developing Java stored procedures in the following resources:

- ▶ *DB2 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7083
- ▶ *Application Programming Guide and Reference for Java*, SC18-9842

To set up a Java enabled stored procedure address space, we performed the steps described in the following sections.

Note: You can download all the necessary material to create this example (such as DDL and SQL statements, Eclipse workspace, and so forth) as described in Appendix C, “Additional material” on page 453.

Creating the WLM environment startup procedure for Java routines

First, you need to set up the WLM environment startup procedure for Java routines. However, the WLM address space startup procedure for Java routines requires extra DD statements that other routines do not need.

Example 6-3 shows a startup procedure for an address space in which Java routines can run. The JAVAENV DD statement indicates to DB2 that the WLM environment is for Java routines.

Example 6-3 Startup procedure for a Java stored procedure WLM address space

```
//DSNWLM PROC RGN=OK,APPLENV=WLMIJAV,DB2SSN=DSN,NUMTCB=5
//IEFPROC EXEC PGM=DSNX9WLM,REGION=&RGN,TIME=NOLIMIT,
// PARM='&DB2SSN,&NUMTCB,&APPLENV'
//STEPLIB DD DISP=SHR,DSN=DSN910.RUNLIB.LOAD
// DD DISP=SHR,DSN=CEE.SCEERUN
// DD DISP=SHR,DSN=DSN910.SDSNEXIT
// DD DISP=SHR,DSN=DSN910.SDSNLOAD
// DD DISP=SHR,DSN=DSN910.SDSNLOD2
//JAVAENV DD DISP=SHR,DSN=WLMIJAV.JSPENV
//JSPDEBUG DD SYSOUT=A
//CEEDUMP DD SYSOUT=A
//SYSPRINT DD SYSOUT=A
//JAVAOUT DD PATH='/tmp/wlmi jav.javaout',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
//JAVAERR DD PATH='/tmp/wlmi jav.javaerr',
// PATHOPTS=(OWRONLY,OCREAT,OTRUNC),
// PATHMODE=SIRWXU
```

WLM application environment values for Java routines

To define the application environment for Java routines to Workload Manager (WLM), we had to specify appropriate values (such as Application Environment Name, Start Parameters, and

so forth) on the WLM setup panels. For more information about how to do this setup, refer to *DB2 9 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7604.

Setting up the JAVAENV file

For Java routines, the startup procedure for the stored procedure address space contains a JAVAENV DD statement. This statement specifies a data set that contains Language Environment runtime options (such as CLASSPATH, DB2_BASE, JCC_HOME, and so forth) for the routines that run in the stored procedure address space. This data set should have record format VB, record length 255, and block size 4096.

The Language Environment runtime options string in a JAVAENV data set has a maximum length of 245 bytes. If you exceed the maximum length, DB2 truncates the contents but does not issue a message. If you enter the contents of the JAVAENV data set on more than one line, DB2 concatenates the lines automatically to form the runtime options string.

If the environment variable list is too long (the JAVAENV content is greater than 245 bytes), you can put the environment variable list in a separate file and use the `_CEE_ENVFILE` variable to point to that file.

Because Java programs could need many different JAR files at run time, the Language Environment runtime options string can exceed 254 bytes very quickly. Therefore, we recommend to use `_CEE_ENVFILE` to point to a file instead of setting values for CLASSPATH or other runtime options in the JAVAENV data set.

Example 6-4 shows a JAVAENV file that we used to run the Java stored procedure.

Example 6-4 Content of JAVAENV file WLMIJAV.JSPENV

```
XPLINK(ON),
ENVAR("DB2_BASE=/usr/lpp/db2/d9gg/db2910_base",
"_CEE_ENVFILE=/u/wagnera/javastp/cee_envfile.txt",
"TZ=EST5EDT",
"JCC_HOME=/usr/lpp/db2/d9gg/db2910_jdbc",
"JAVA_HOME=/usr/lpp/java/J5.0"),
MSGFILE(JSPDEBUG,, ,ENQ)
```

This file contains an environment variable called `JAVA_HOME`. With this variable, you can point to a specific directory with a Java version in your runtime environment. In our system, we used Java version 5.0.

Important: We recommend that you use the same Java compiler version in your development environment as the one configured for your DB2 Java stored procedure runtime environment.

We used the `_CEE_ENVFILE` variable to point to a separate file with further runtime options. This file is stored in the UNIX System Services file system. See Example 6-5 for an example that we used.

Example 6-5 Content of /u/wagnera/javastp/cee_envfile.txt file

```
CLASSPATH=/u/wagnera/javastp/PdfGenerate.jar:/u/wagnera/javastp/lib/iText-2.1.5.jar
```

The Java program for the stored procedure needs two different JAR files:

<code>PdfGenerate.jar</code>	Contains our self-written Java code
<code>iText-2.1.5.jar</code>	Contains classes for generating PDF files

In the next chapter, we explain in detail how to create and store the JAR files in the UNIX System Services file system.

After we completed these steps, our system environment was prepared to run a Java stored procedure.

Note: The user who is supposed to start the stored procedure address space must have READ authority to all configuration and JAR files that are used by the stored procedure. Note that this user is usually *not* the same user who calls the DB2 stored procedure!

6.3.3 Development of the sample application

In our example, we assume that an existing COBOL batch program needs to be enhanced. A new requirement has resulted in the ability of the program to process XML data to generate PDF files. The input data is stored in a DB2 XML column.

Because it is very easy to use Java to generate PDF files, we decided to create a Java stored procedure to implement this new requirement. We then call this new Java stored procedure from the existing COBOL program. The only modification needed in the existing COBOL program is a call to the new Java stored procedure.

In the following sections, we describe how to develop the Java stored procedure using Eclipse tooling. As mentioned previously, you can use this stored procedure to generate PDF files from XML data. As shown in Figure 6-3, the procedure processes all data with one single call.

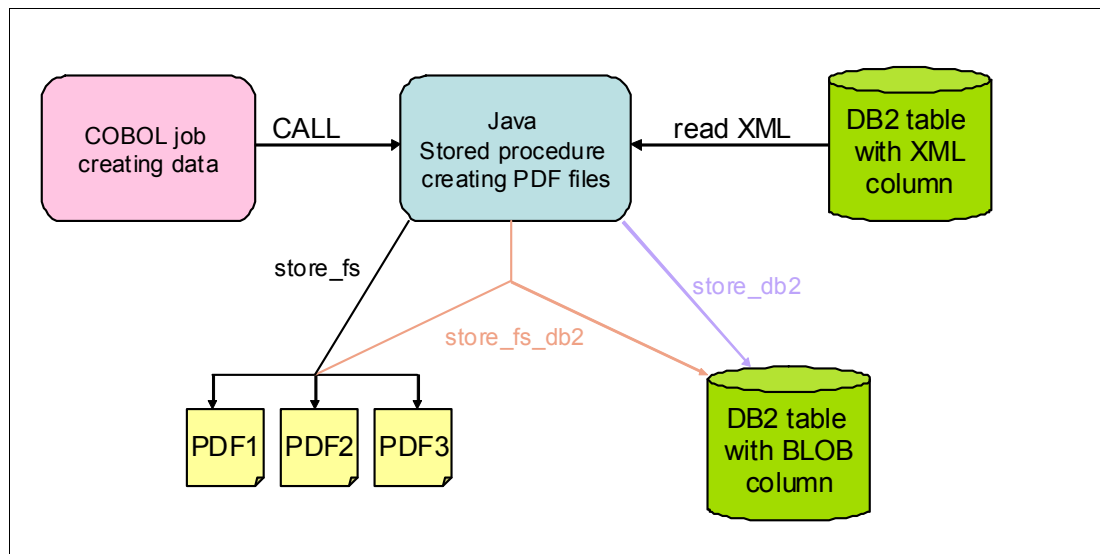


Figure 6-3 Java stored procedure

The flow of the procedure is as follows:

1. Read XML formatted data from DB2 by using XML functionality.
2. Generate files in PDF format.
3. Store generated PDF files in DB2 as Binary Large Object (BLOB) or in a UNIX System Services file system or both.

Setting up the DB2 environment

To run our example, we first create a DB2 table with one XML and one BLOB column, as shown in Example 6-6.

Example 6-6 Create table for Java stored procedure example

```
CREATE TABLE BATCH.XML2PDF
  (ID          INTEGER      NOT NULL
  ,XML_DATA    XML
  ,PDF_DATA    BLOB(1M)
  ,PRIMARY KEY (ID)
  )
```

Next, we insert test data in XML format. This XML data should contain information about a company (all employees and the departments for which they work). The easiest way to insert the data is using an SQL INSERT statement. For our test, we inserted only two rows in DB2 with data from two different companies.

Example 6-7 shows an extract of our INSERT statements. Example A-4 on page 426 lists the complete statements for this example.

Example 6-7 Insert XML test data

```
-- Insert first company
INSERT INTO BATCH.XML2PDF (ID, XML_DATA)
VALUES (1,
'<?xml version="1.0"?>
<Company>
  <Name>Big data center</Name>
  <Department>
    <DepNo>1</DepNo>
    <Description>Production</Description>
    <Employee>
      <EmpNo>1</EmpNo>
      <FirstName>Paul</FirstName>
      <LastName>Smith</LastName>
    </Employee>
    <Employee>
      <EmpNo>2</EmpNo>
      <FirstName>Hannah</FirstName>
      <LastName>Smith</LastName>
    </Employee>
  </Department>
  <Department>
    <DepNo>2</DepNo>
    <Description>Development</Description>
    <Employee>
      <EmpNo>3</EmpNo>
      <FirstName>Mia</FirstName>
      <LastName>Brown</LastName>
    </Employee>
  .....

```

Developing the Java stored procedure

After setting up the DB2 environment, we implement the stored procedure. For development, we use an Eclipse project. We start the Eclipse workbench and create a new Java project called *JavaSTP* by selecting **File** → **New** → **Java Project**.

Important: We recommend that you use the same Java compiler version in your Eclipse environment as the one configured for your DB2 Java stored procedure runtime environment.

Next, we create a new package called `com.ibm.itso.sample` in the `src` folder. Afterwards, we create the following new classes in that package:

- ▶ `GenPdf.java`
- ▶ `PdfCreator.java`

You can find the complete code for every class in “Java stored procedure to generate PDF files” on page 430.

In the `GenPdf` class, we implement only the one `runGenerate` method. Example 6-8 shows the content of this method. This method must be `static` because DB2 is calling this method when the Java stored procedure is executed. The method contains the following parameters to allow the user to influence the flow of the Java program:

<code>action</code>	<code>store_db2</code> : store PDF file in DB2 BLOB column <code>store_fs</code> : store PDF file in UNIX System Services file system <code>store_fs_db2</code> : store PDF file in DB2 and UNIX System Services file system
<code>pdfDir</code>	Directory to store PDF files in the UNIX System Services file system

From a Java stored procedure perspective, these two parameters are used only to control the flow of the Java coding. They are pure input parameters.

With DB2 stored procedures, you can also return values to the caller. You can use result sets for multiple returned rows of data. Returned parameters are called *output parameters*. It is also possible to define a parameter for input and output processing.

When defining the DB2 stored procedure you have to decide whether the parameter is input (IN), output (OU), or both input and output (INOUT).

Example 6-8 The runGenerate method in the GenPdf Java Class

```
public static void runGenerate(String action, String pdfDir)
    throws SQLException {
    PdfCreator pdfc = new PdfCreator();
    pdfc.generatePDF(action, pdfDir);
}
```

Note: DB2 passes INOUT and OUT parameters as single-entry arrays. Thus, in the Java routine, you must declare OUT or INOUT parameter as arrays. For more information, see *DB2 Version 9.1 for z/OS Application Programming Guide and Reference for Java*, SC18-9842.

Next, we code the `PdfGenerator.java` class, which reads the XML data, generates PDF files, and stores the files in DB2 or the UNIX System Services file system.

To create PDF files, we use the Open Source Java library `iText-2.1.5.jar`. To use this library, we follow these steps:

1. We first create a folder called `lib` in the Eclipse Project
2. Then, we download `iText-2.1.5.jar` from the following Web site:
<http://www.lowagie.com/iText/download.html>
3. Finally, we save this file in the `lib` folder of the Eclipse project.

Because the library `iText-2.1.5.jar` is needed to develop and run the Java program, we copy this file through FTP to the runtime environment of the Java stored procedure on z/OS. In our case, as shown in Example 6-5 on page 66, the Java environment expects this file in the UNIX System Services folder `/u/wagnera/javastp/lib`.

Then, we include the `iText-2.1.5.jar` in the Eclipse Build Path by right-clicking the Java project and selecting **Build Path** → **Configure Build Path** as shown in Figure 6-4.

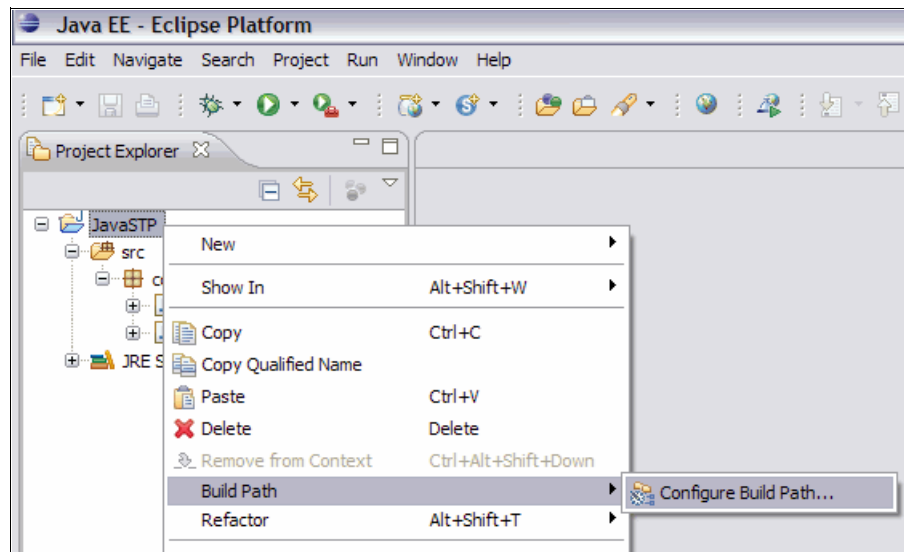


Figure 6-4 Configure Build Path in Eclipse workbench

In the next window, we select **Add Jars**, select the `iText-2.1.5.jar` file, and press **OK** twice. Now, the two classes—`GenPdf` and `PdfCreator`—compile without any errors.

Next, we take a closer look at some code of class `PdfCreator`. As shown in Example 6-9, the connection URL in our case is `jdbc:default:connection`. This URL forces DB2 to use the same physical thread that processes the `CALL` statement for the stored procedure, which is important regarding DB2 processing. For example, if DB2 uses a different thread for SQL processing within our stored procedure than the process that called our stored procedure, we cannot access data within the stored procedure because it is blocked by the process that called the stored procedure.

Example 6-9 Get Connection within Java stored procedure

```
// Get connection to the database
Connection con = DriverManager.getConnection("jdbc:default:connection");
```

In our example, we use XML data as input for our stored procedure, as shown in Example 6-10. Thereby, each row of our table contains all data from one single company.

Example 6-10 SELECT data from XML column

```
SELECT ID, EMP.*
FROM BATCH.XML2PDF
     , XMLTABLE('$X/Company/Department/Employee' PASSING XML_DATA AS "X"
              COLUMNS COMPANY  VARCHAR(50) PATH '../..Name'
                    , DEPT_NO    INTEGER    PATH '../DepNo'
                    , DEPT_DESCR VARCHAR(20) PATH '../Description'
                    , EMP_NO     INTEGER    PATH 'EmpNo'
                    , FIRST_NAME VARCHAR(20) PATH 'FirstName'
                    , LAST_NAME  VARCHAR(20) PATH 'LastName'
              ) EMP
ORDER BY ID, EMP.DEPT_NO, EMP_NO;
```

Because XML is fully integrated to the DB2 system, you can access and manage XML data using DB2 functions and capabilities. These include functions such as parsing, validation, index support for fast queries, and much more.

To receive values from DB2, we use the XMLTABLE function with XPath instructions. The XMLTABLE function returns information from XML data in the form of a DB2 table. With the SELECT statement shown in Example 6-10, we retrieve all data of all companies using one single DB2 cursor.

Note: For detailed information regarding DB2 and XML processing, see *DB2 Version 9.1 for z/OS XML Guide*, SC18-9858.

Depending on the input parameter value, we store the generated PDF data in the UNIX System Services file system, a DB2 BLOB column, or both. Example 6-11 shows the relevant Java code.

Example 6-11 Save PDF file in DB2 or UNIX System Services file system

```
// save PDF file in USS file system
FileOutputStream fOut = new FileOutputStream(pdfDir + File.separator
      + "Report" + oldId + ".pdf");
out.writeTo(fOut);
fOut.close();

// save PDF file in DB2 BLOB column
String strSQL = "UPDATE BATCH.XML2PDF SET PDF_DATA = ? WHERE ID = ?";
PreparedStatement psUpd = con.prepareStatement(strSQL);
InputStream is = new ByteArrayInputStream(out.toByteArray());
psUpd.setBinaryStream(1, is, -1);
psUpd.setInt(2, oldId);
psUpd.execute();
```

By calling the stored procedure with the value `store_fs` for the first parameter, the generated PDF file for each company is stored in the UNIX System Services file system. Using the `store_db2` value, the PDF files are saved in DB2. Using the `store_fs_db2` value, the PDF files are saved in both DB2 and the UNIX System Services file system.

Creating the DB2 stored procedure

The next step is to define the Java stored procedure in DB2 by issuing a CREATE PROCEDURE statement. To form the correct statement, you need the following information:

- ▶ IN, OUT, and INOUT parameters for the stored procedure
- ▶ Is the stored procedure returning data using DB2 result sets?
- ▶ Java method that should be issued when the stored procedure is called
- ▶ Name of the WLM Java Application Environment

Example 6-12 shows the CREATE statement for the stored procedure. There are many more parameters available. For detailed information about each parameter, refer to the SQL documentation of DB2 for z/OS.

Example 6-12 Creating the stored procedure in DB2

```
CREATE PROCEDURE BATCH.JAVASTP
  (IN ACTION VARCHAR(15)
  ,IN PDF_DIR VARCHAR(100))
EXTERNAL NAME 'com.ibm.itso.sample.GenPdf.runGenerate'
LANGUAGE JAVA
PARAMETER STYLE JAVA
NOT DETERMINISTIC
FENCED
CALLED ON NULL INPUT
MODIFIES SQL DATA
NO DBINFO
NO COLLID
WLM ENVIRONMENT D9GGWLMJ
ASUTIME NO LIMIT
STAY RESIDENT YES
PROGRAM TYPE SUB
SECURITY DB2
INHERIT SPECIAL REGISTERS
STOP AFTER SYSTEM DEFAULT FAILURES
COMMIT ON RETURN NO;
```

```
GRANT EXECUTE ON PROCEDURE BATCH.JAVASTP TO PUBLIC;
```

Of course, it is not necessary to re-create the stored procedure in DB2 only because the Java code of the stored procedure is modified. However, if you need to change one of the parameters or add another parameter needs, you must re-create the stored procedure in DB2.

Deploying the Java files to the UNIX System Services file system

Because our stored procedure runs on z/OS, we build a JAR file and copy this file to the z/OS environment. As shown in Example 6-5 on page 66, the Java environment expects the file PdfGenerate.jar in UNIX System Services folder /u/wagnera/javastp. To build and copy the JAR file, we used an Ant script in Eclipse.

To use Ant, you download the following Open Source libraries:

- ▶ The jakarta-oro-2.0.8.zip file from the Apache Jakarta Project at:
http://jakarta.apache.org/site/downloads/downloads_oro.cgi
- ▶ The commons-net-2.0.zip file from the Apache Commons Project at:
http://commons.apache.org/downloads/download_net.cgi

From these compressed files, you extract the following files:

- ▶ commons-net-2.0.jar
- ▶ jakarta-oro-2.0.8.jar

Then, save these files in a folder on the hard disk. After you save the files, use **Window** → **Preferences** in the Eclipse workbench and activate the **Ant - Editor - Runtime**. As shown in Figure 6-5, click **Add External JARs** and add the two JAR files to the Ant run time.

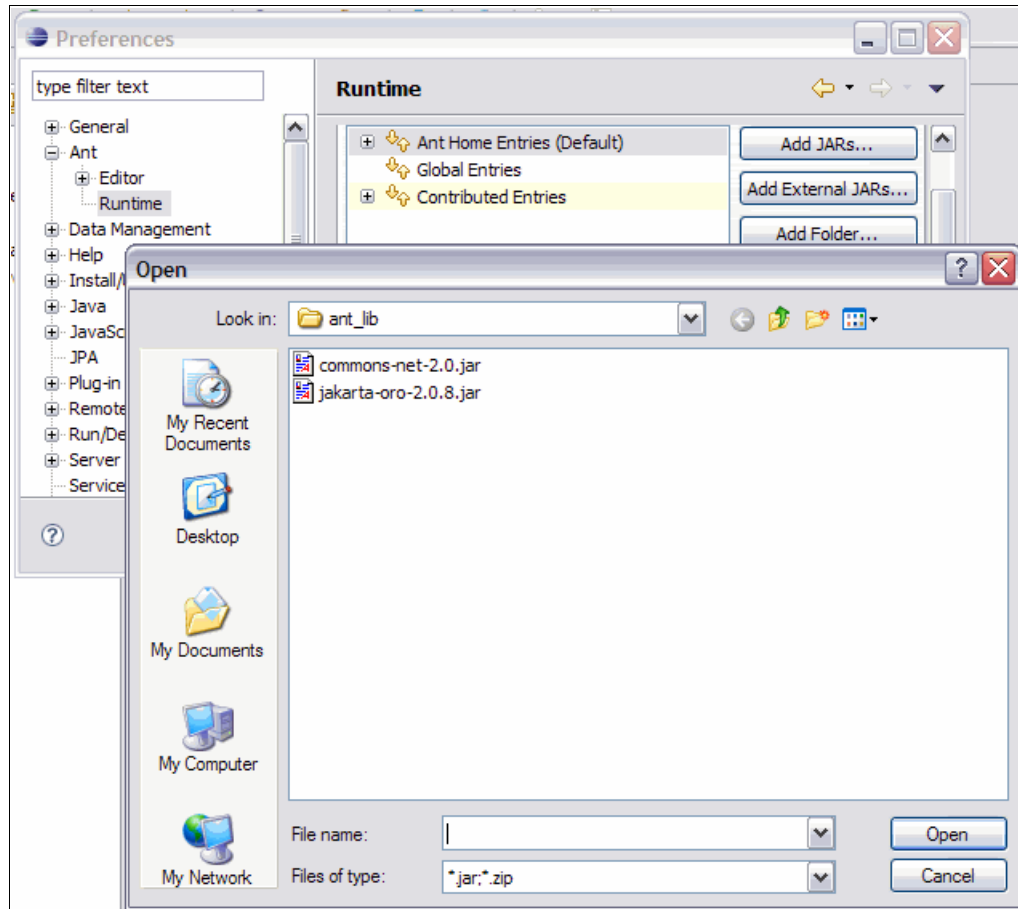


Figure 6-5 Add JAR files to Ant run time

Now, right-click the Java project JavaSTP, and select **New** → **File** to create a file called `deploy.xml`. In this file, we configure the necessary steps for the deploy process (that is compile the Java source, build the JAR file, and copy the JAR file to UNIX System Services file system). For the entire content of this file, see Example B-3.

Next, right-click the Java project JavaSTP, and select **New** → **File** to create a file called `zos.properties`. In this file we configure necessary parameters that are used as input for the `deploy.xml` script to copy the JAR file (server, user, password, JAR directory, and JAR name). See Example 6-13.

Example 6-13 The zos.properties file for the deploy process

```
# either the Eclipse workspace directory or your "home directory"
server = zos.server.dns.name
userid = uid
password = pw
# the home directory for deploying the application jar
```

```
apl.home = /u/wagnera/javastp
# jar name
jarname = PdfGenerate.jar
# time to wait for output in seconds
waittime = 0
# remote debugging
debug= no
```

The JAR file is copied to the UNIX System Services file /u/wagnera/javastp/PdfGenerate.jar. As mentioned previously, the user who starts the stored procedure address space must have READ authority to this file.

Finally, we build and copy the JAR file with the Ant script to z/OS by right-clicking **deploy.xml** and selecting **Run As** → **Ant Build**.

Refreshing the WLM Application Environment

Every time you change the Java code under Eclipse, you have to deploy the new JAR files to the UNIX System Services file system with the Ant script. However, to activate the new version, you also have to refresh the current WLM Application Environment.

The easiest way to refresh this environment is to issue the DB2 system stored procedure SYSPROC.WLM_REFRESH. This stored procedure uses the following parameters:

IN - WLM_ENV_NAME	Name of the WLM application environment
IN - SSID	DB2 subsystem ID
OUT - STATUS_MSG	Message text from system stored procedure
OUT - STATUS_CODE	Return code from system stored procedure

We recommend to use a local Java application to call this system stored procedure.

To do this, we create a second Java project called UtilSTP (click **File** → **New** → **Java Project**) in Eclipse. Then, we create the package com.ibm.itso.sample in the src folder of the UtilSTP project. In this package, we create the new Java class RefreshWLM. Example 6-14 shows the relevant code to refresh our WLM Application Environment. You can find the entire Java source code in Example B-5 on page 435.

Example 6-14 Refresh application environment

```
CallableStatement cs = con.prepareStatement("CALL SYSPROC.WLM_REFRESH(?, ?, ?, ?)");
cs.setString(1, "D9GGWLMJ");
cs.setString(2, "D9G1");
cs.registerOutParameter(3, Types.VARCHAR);
cs.registerOutParameter(4, Types.INTEGER);
cs.execute();
System.out.println("WLM_REFRESH ended with: RC=" + cs.getInt(4)
    + ", Text: " + cs.getString(3));
```

Because our local Java program connects to DB2 on z/OS, we downloaded using FTP the db2jcc.jar and the db2jcc_license_cisuz.jar files from the z/OS system (the exact folder where it is stored can be retrieved from the database administrator) into the root directory of the UtilSTP project. These two JAR files are necessary to use JDBC. We also put these two files in the build path of the UtilSTP project by selecting **Project** → **Build Path** → **Configure Build Path**.

Important: Every time you deploy a new version with the Ant script, you have to issue RefreshWLM (right-click **RefreshWLM** and select **RunAs** → **Java Application**) to activate the new version in DB2 for z/OS.

Testing the stored procedure

The easiest way to test the stored procedure is to use a local Java test application. Therefore, we created a new class called TestSTP in the Java UtilSTP project. Example 6-15 shows the relevant lines of code. You can find the entire Java source code in Example B-6 on page 436.

Example 6-15 Testing the stored procedure

```
CallableStatement cs = con.prepareCall("CALL BATCH.JAVASTP(?, ?)");
cs.setString(1, "store_fs_db2");
cs.setString(2, "/u/wagnera/javastp/pdfs");
cs.execute();
```

In our example, the PDF files are stored in the UNIX System Services directory `/u/wagnera/javastp/pdfs`. This directory must exist in the UNIX System Services file system, and the user who starts the stored procedure address space must have authority to create files in this directory.

To start the test program, right-click **TestSTP**, and select **RunAs** → **Java Application**. After the test program completes, we downloaded using FTP the generated PDF files from the UNIX System Services directory `/u/wagnera/javastp/pdfs`. Figure 6-6 shows an example PDF file.

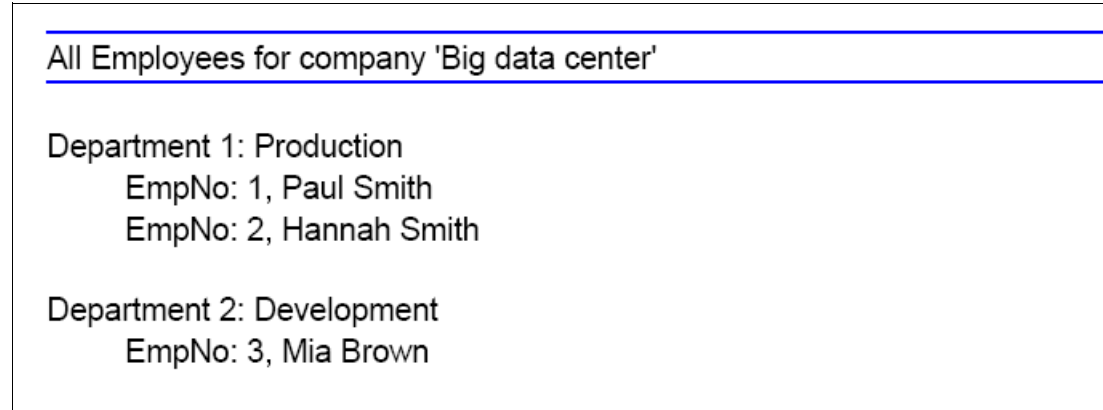


Figure 6-6 Generated PDF file

Implement new functionality using stand-alone Java

In addition to using middleware that provides a runtime environment for Java, you can run Java as a stand-alone batch job. Everything you need to run Java as a stand-alone batch job is part of the z/OS operating system. Figure 7-1 illustrates how to incorporate Java into the batch environment.

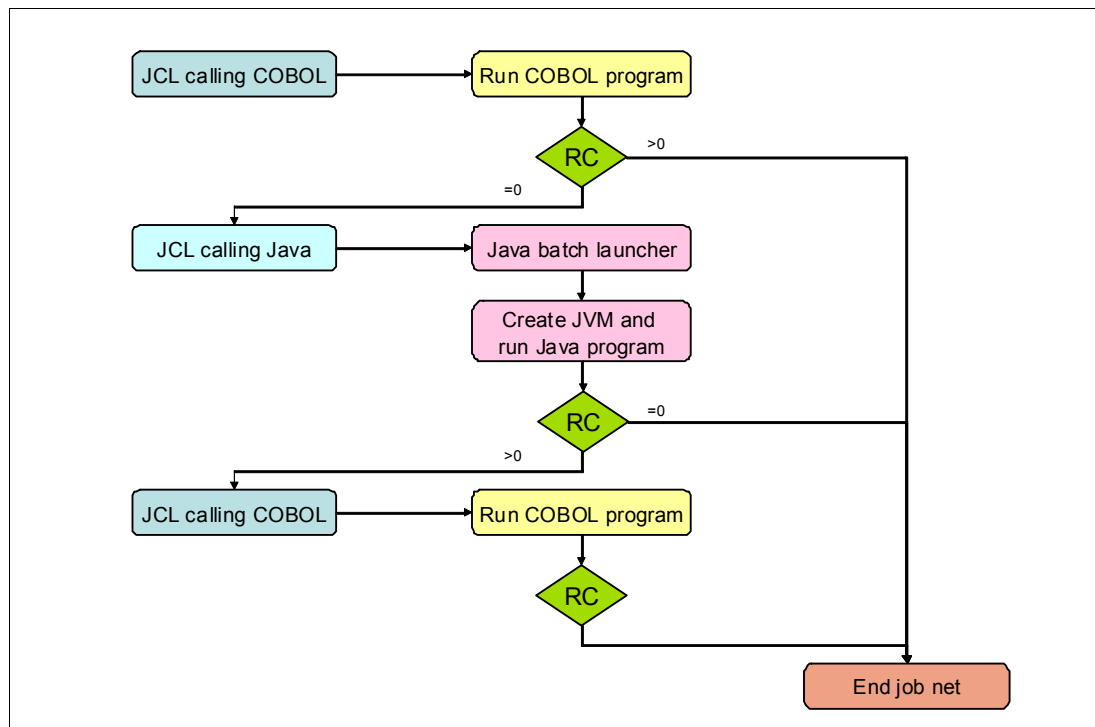


Figure 7-1 Overview of stand-alone Java batch

You can completely integrate Java into your existing batch environment. For example, after running a COBOL preparation step with a Return Code (RC) of zero, you can then use Java

to implement functionality that is difficult to implement in COBOL. In the Java step, you need a “launch” mechanism that creates the JVM and launches the Java program. The launched Java program itself can then generate RCs at the end so that the job automation starts other jobs based on the results.

In this chapter, we discuss different z/OS stand-alone Java batch launchers, introduce ways to interact from Java with other languages, talk about development capabilities in stand-alone Java batch, and give an example of how to enrich an existing COBOL job with Java functionality in a stand-alone batch environment.

This chapter includes the following topics:

- ▶ Running Java with the BPXBATCH or BPXBATSL utilities
- ▶ Running Java with JZOS
- ▶ Interoperability with other languages
- ▶ Development tools
- ▶ Sample stand-alone Java batch application

7.1 Running Java with the BPXBATCH or BPXBATSL utilities

You can invoke a Java program in a job on z/OS using one of the following alternatives:

- ▶ The BPXBATCH or BPXBATSL utility
- ▶ IBM JZOS Batch Toolkit for z/OS (JZOS)

We discuss the first option briefly in this chapter. We discuss the second option in the next section, Running Java with JZOS.

BPXBATCH and BPXBATSL are batch launcher facilities under z/OS that allow you to launch z/OS UNIX System Services programs in batch. Therefore, you can also use them to launch a Java Virtual Machine (JVM) in a batch job on z/OS. However, compared to the JZOS batch launcher, these utilities have some limitations:

- ▶ No flexible configuration of the z/OS UNIX System Services environment for each job
- ▶ No condition code that is passed from Java programs
- ▶ When launched with BPXBATCH, the z/OS UNIX System Services program runs in a separate address space and prevents the use of DD statements by the UNIX program.
- ▶ When launched with BPXBATSL, the z/OS UNIX System Services program is launched in the same address space and DD statements can be used.

You can find more information about how to use BPXBATCH and BPXBATSL for Java batch on z/OS in *Java Stand-alone Applications on z/OS, Volume 1*, SG24-7177.

7.2 Running Java with JZOS

The JZOS batch launcher is a good option to enhance an existing stand-alone z/OS batch environment with additional functionality written in Java. JZOS is included in the following versions of the IBM SDK for z/OS:

- ▶ Java 31-Bit SDK 1.4.2 SR6 for z/OS or higher
- ▶ Java 31-Bit SDK 5.0 SR3 for z/OS or higher
- ▶ Java 64-Bit SDK 5.0 SR3 for z/OS or higher
- ▶ Java 31-Bit SDK 6.0 all versions
- ▶ Java 64-Bit SDK 6.0 all versions

Figure 7-2 illustrates how JZOS works. A Job Control Language (JCL) calls the JZOS batch launcher load module which creates the JVM under UNIX System Services. The JCL contains all relevant information about how to launch the batch job:

- ▶ DD statements that are expected by the Java program
- ▶ Java main class
- ▶ Java version that is needed for this job
- ▶ Classpath
- ▶ Libpath
- ▶ Additional JVM arguments
- ▶ JVM Encoding

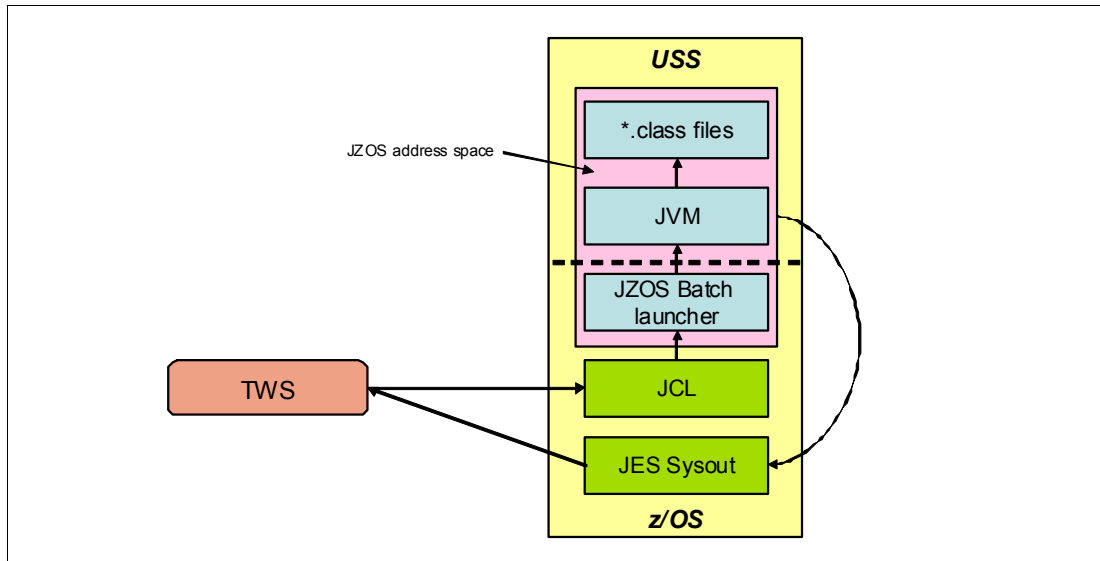


Figure 7-2 JZOS overview

The JVM called by the JZOS load module then initiates the Java program by calling the main class. While the Java batch job is running, all `System.out.println` and `System.err.println` are redirected to the `SYSOUT` and `SYSERR` DD designations, respectively, of the job.

Another advantage of JZOS is that the launcher and the JVM run in the same address space, making job accounting easier and allowing the use of DD statement. Because JZOS Java batch jobs are launched by standard z/OS JCL, it is easier to integrate Java batch jobs in workload scheduling products such as Tivoli Workload Scheduler.

Note: Every time you start a JZOS batch job, a JVM is initiated. For further information about how this might impact performance see 18.2, “Stand-alone Java batch” on page 339.

In summary, the JZOS batch launcher is a good tool to enrich your existing batch environment with Java functionality.

For further information about JZOS, see the following Web site:

<http://www.ibm.com/servers/eserver/zseries/software/java/products/jzos/overview.html>

Another excellent resource is *Java Stand-alone Applications on z/OS Volume II*, SG24-7291.

7.3 Interoperability with other languages

In addition to using native language calls as described in 5.5, “Java Interoperability with COBOL and PL/I” on page 55, you can use the functionality provided by the z/OS batch environment to establish a *mixed* environment where the Java application and the applications written in other languages are in separate *jobs* or *job steps*. These jobs are then connected afterwards by the job automation (see Figure 7-1 on page 77). The data transfer between these job steps can be realized by storing data in temporary data sets or in UNIX System Services files.

Combining programming languages provides a clear separation of the program logic. In the case of program errors, having program logic separated helps to locate the error easily. Furthermore, less interaction between the different developers is needed.

Alternatively, if we have a high frequency of switching between short Java steps and steps written in another language, the overhead for each JVM startup might become a disadvantage. In that case, direct language calls from object-oriented (OO) COBOL to Java and the other way around might be more attractive (see 5.5, “Java Interoperability with COBOL and PL/I” on page 55).

7.4 Development tools

Because Java developers very often come from the distributed world, the tooling for developing stand-alone Java batch applications becomes a very important aspect. In this section, we discuss the following development tools:

- ▶ Tooling for the z/OS UNIX System Services environment
- ▶ Tooling for developing the Java code itself

For Java developers that have never worked on z/OS, the OMVS Shell and the ISHELL might be difficult to handle. To solve this issue, the z/OS UNIX System Services environment can be enriched with some tools that are very common in the Linux and UNIX space, such as:

- ▶ bash
- ▶ vi
- ▶ viascii
- ▶ vim
- ▶ openssh

These tools, and many others, can also be used on z/OS. For more information, see the z/OS UNIX System Services Tools at the following Web site:

<http://www.ibm.com/servers/eserver/zseries/zos/unix/tools/>

For Java code development, Eclipse-based tools are very common. Such Eclipse-based tools can also be used for stand-alone Java batch development in combination with the JZOS batch launcher.

An excellent starting point for developing stand-alone Java is the JZOS Cookbook, which you can download from:

<http://www.alphaworks.ibm.com/tech/zosjavabatchtk/download>

If you want to use the Eclipse platform only for development of z/OS batch written in Java, you can find a cost free and very efficient solution described in *Java Stand-alone Applications on z/OS Volume II*, SG24-7291. This book includes a chapter that describes a solution that is based on Ant scripts, which allows you to compile and deploy a Java batch application to z/OS with only a few clicks. This solution combines the advantages of Eclipse-like syntax highlighting and code completion with an easy deployment to z/OS, including job output in Eclipse. In most cases, you will not need 3270 sessions.

If you also want to develop programs in traditional languages such as COBOL and PL/I in Eclipse with features such as syntax highlighting or code completion, IBM Rational Developer for System z is a solution. Rational Developer for System z is an Eclipse-based integrated development environment (IDE) that allows application developers to edit, compile, build, and interactively debug COBOL, PL/I, and Java programs. Syntax highlighting, build assistance, and interactive debug of applications running on remote z/OS systems are supported. In addition, users can directly access z/OS data sets, including JCL and UNIX System Services files, and can view VSAM data using the IDE. For Java developers who are familiar with an IDE, Rational Developer for System z can ease interaction with z/OS and raise productivity.

You can find further details about how to use Rational Developer for System z for z/OS application development in *Topics on Version 7 of IBM Rational Developer for System z and IBM WebSphere Developer for System z*, SG24-7482.

Note: you can also combine the advantages of the Ant script and Rational Developer for System z.

7.5 Sample stand-alone Java batch application

In this section, we describe a sample of how to modernize an existing batch environment with stand-alone Java with the purpose of getting access to more functionality in batch.

Figure 7-3 shows a job net that originally consists of two jobs managed by Tivoli Workload Scheduler:

- ▶ A COBOL stand-alone batch job that inserts data for invoices into DB2 on z/OS based on input from a sequential data set.
- ▶ A COBOL job that creates invoices based on the DB2 data and sends those invoices to a physical printer.

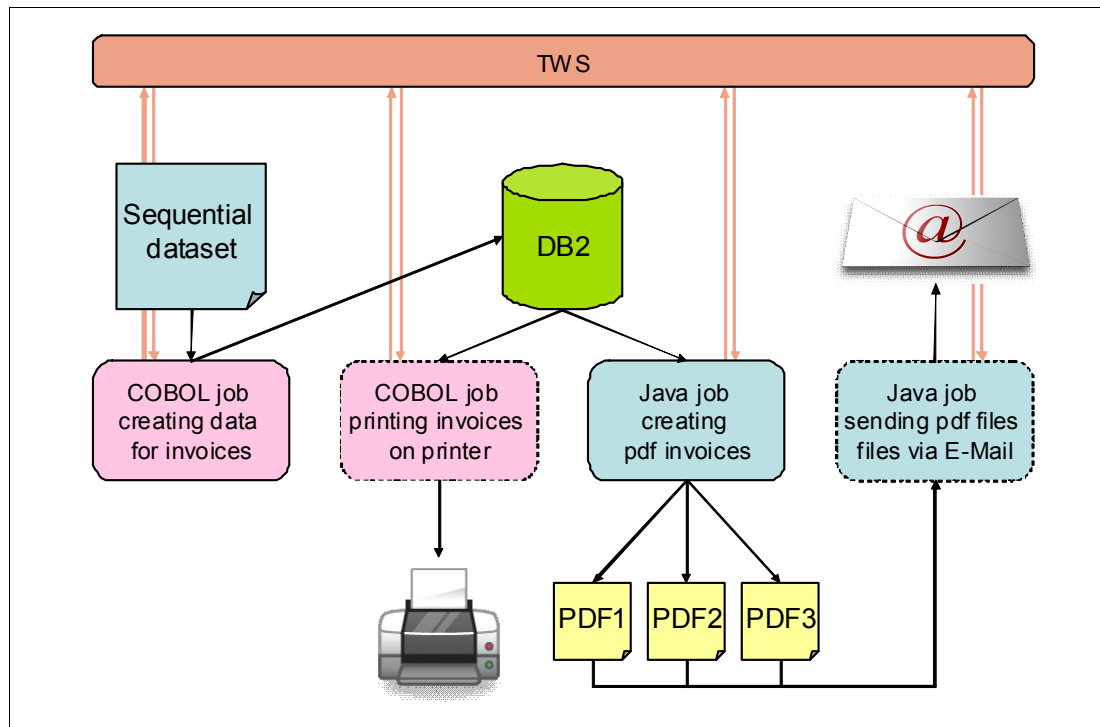


Figure 7-3 Stand-alone Java batch sample overview

Because printed invoices cost money (in the form of printer cost, paper, and postage), invoices based on PDF files are an alternative. However, because creating PDF files in COBOL and PL/I is more difficult and time consuming, we created a third job in this sample that is written in Java. This Java job uses the same DB2 input data as the COBOL printing job, but instead of real printed invoices, it creates PDF files using a Java PDF library. To send those invoices directly to the recipient, we create another job in Java that sends these files using e-mail to their recipients. All of the jobs and their dependencies are managed by Tivoli Workload Scheduler.

Note: We do not show how to implement the COBOL print job and the e-mail send job because most z/OS users will understand this process. Furthermore, sending e-mails with Java is quite easy and, therefore, we do not need explain it further here. Our purpose is to show how to use Java to implement a new piece of functionality and integrate it within the batch flow, not to explain all the possible functionality of Java.

This scenario demonstrates the easy integration of Java functionality into an existing z/OS batch landscape. The Java job behaves as all other jobs:

- ▶ The job is submitted by JCL.
- ▶ JES takes care of the job management.
- ▶ The job can be managed by a workload scheduler such as Tivoli Workload Scheduler.
- ▶ The job is accountable similar to a COBOL or PL/I job.
- ▶ WLM manages the job priorities.

The advantage of using Java in this scenario is that Java provides an easy way to solve the functionality requirements. For e-mail or PDF creation, Java includes libraries that only need a few lines of code to implement the job.

7.5.1 Creating invoice data with COBOL

As a first step, we create a COBOL job that reads invoice data from a sequential MVS data set and inserts this data in a sorted way into DB2 on z/OS.

Figure 7-4 shows the Entity Relationship Model (ERM) of the DB2 database that we use. The COBOL job uses a flat file as input and transforms it into the corresponding tables.

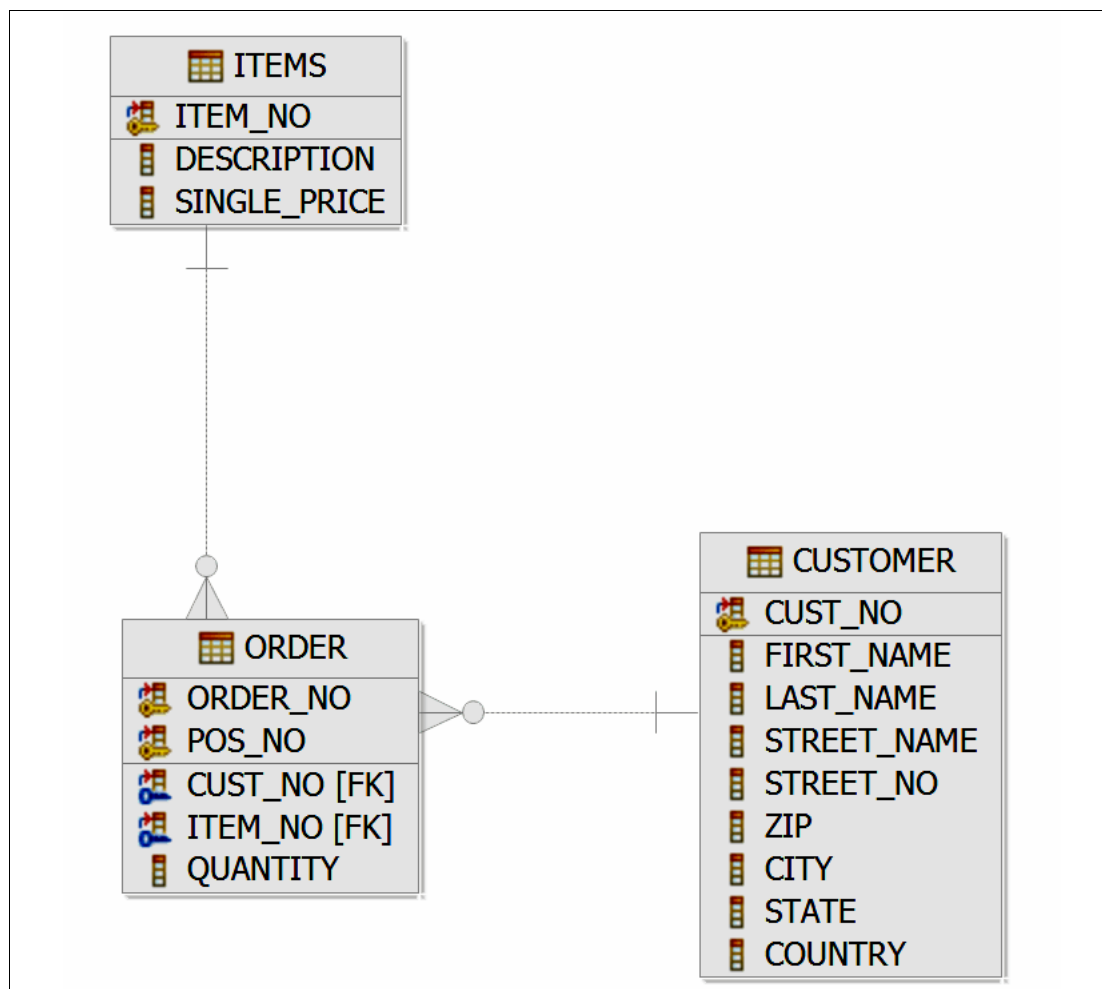


Figure 7-4 Invoice data for the Entity Relationship Model

You can find the Data Definition Language (DDL) and SQL statements to insert this sample in Appendix A, “DB2 configuration” on page 423 as well as in the Eclipse project file that is included in the additional material for this book. You can find the COBOL source code and the flat file in Appendix C, “Additional material” on page 453.

7.5.2 Generating a PDF in Java

After the invoice data is inserted into DB2, we create a new job in Java that connects to DB2 and reads the data for PDF generation. This job is based on an Eclipse project. You can find the entire project in Chapter C, “Additional material” on page 453.

To create this new job, we first implement the HelloWorld sample as described in Chapter 4, “Java development and job management with Eclipse” in *Java Stand-alone Applications on z/OS Volume II*, SG24-7291. Then, to create the PDF files, we follow these steps:

1. Use the Open Source Java library iText-2.1.5 from:

<http://www.lowagie.com/iText/>

Downloaded the iText-2.1.5.jar file and save this file in the deploy folder of the Eclipse project.

2. Next, download using FTP the db2jcc.jar and db2jcc_license_cisuz.jar files from the z/OS system into the root directory of the Eclipse folder. (You can obtain the exact folder name where these files are stored on the z/OS system from the database administrator.) These two JAR files are necessary to use JDBC.
3. Select the project in the Project Explorer, and press F5 to refresh it. The directory structure looks similar to that shown in Figure 7-5.

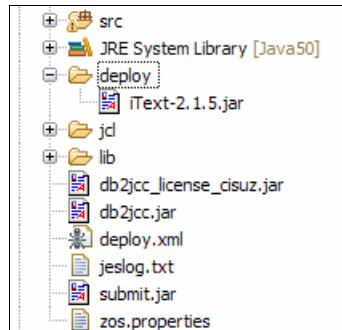


Figure 7-5 Eclipse directory structure

- Next, include these three JAR files in the Eclipse Build Path. Right-click the project, and select **Build Path** → **Configure Build Path** as shown in Figure 7-6. In the window that opens, select **Add Jars**.

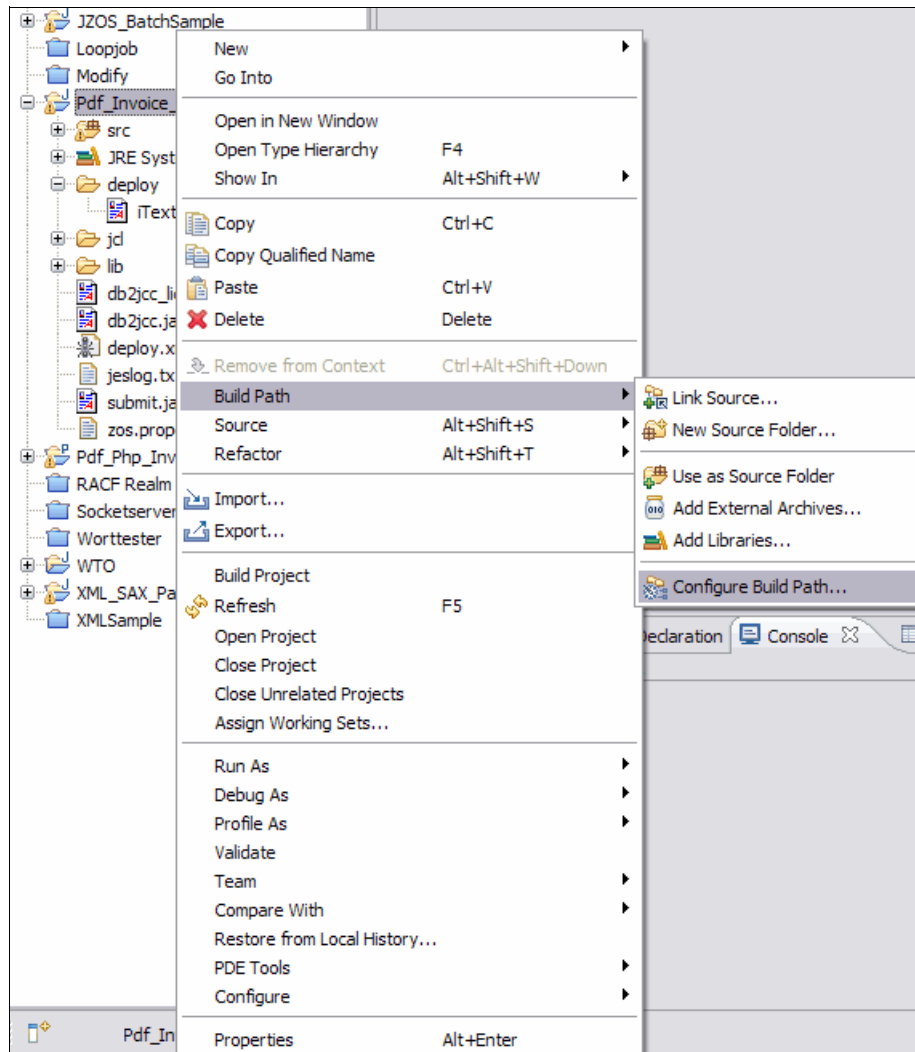


Figure 7-6 Configuring the Java Build Path

5. Next, select the DB2 and iText JAR files as shown in, Figure 7-7 and press **OK**.

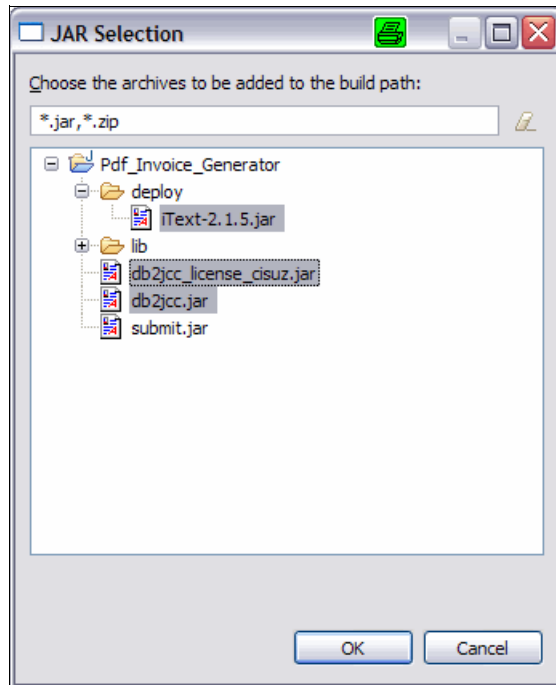


Figure 7-7 JAR file selection

6. Figure 7-8 shows the build path for our test environment. Select **OK**.

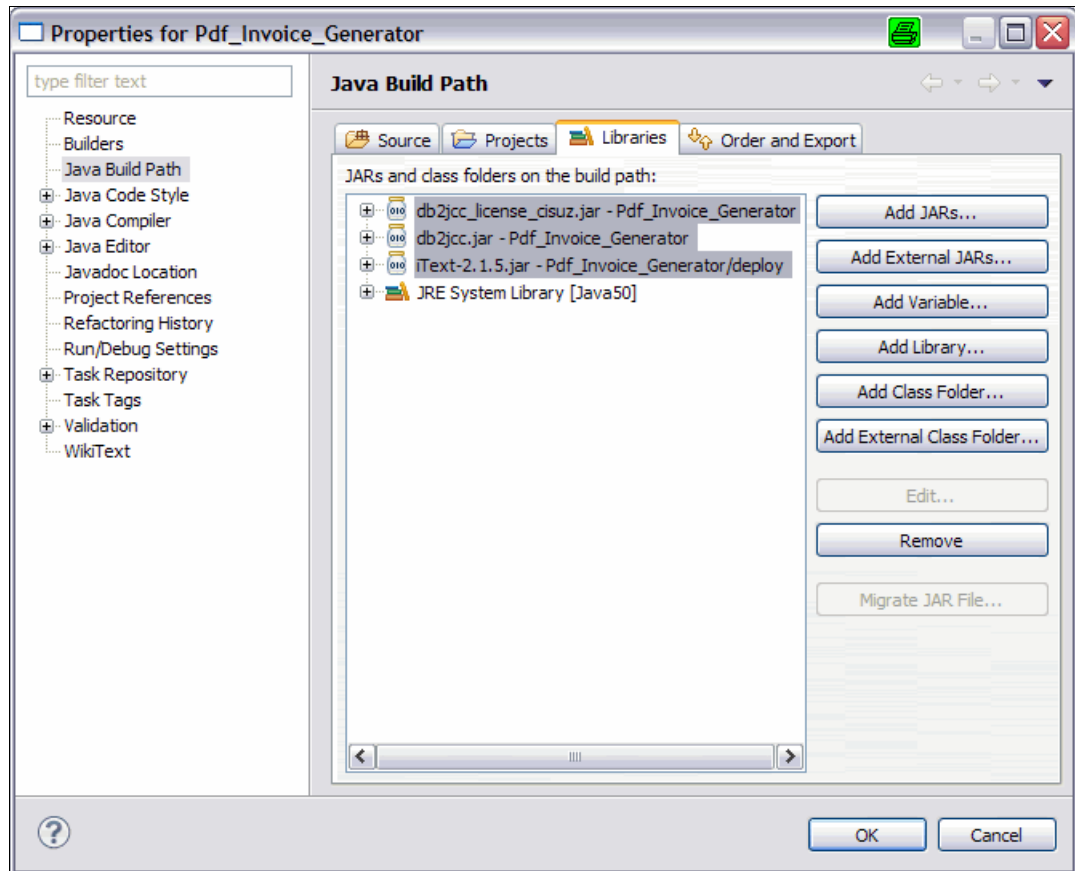


Figure 7-8 Java Library Path

7. Open the zos-properties file and enter a new JAR file name:

```
jarname = pdfinvoice.jar
```

8. Now, to code the program, we create a new package called `com.ibm.itso` in the `src` folder. Then, we create the following new classes in that package:

- InvoiceCreator.java
- PdfCreator.java

In both classes, we add the Java code to get the required invoice data from the DB2 tables using JDBC and the iText classes to generate the PDF files. You can find the complete code in “Java PDF creator” on page 437.

Note: Because we use only the JDBC Type II Driver for the cross-memory data access between DB2 and z/OS, the two JAR files, `db2jcc_license_cisuz.jar` and `db2jcc.jar`, are not required to be in the Java Build Path of Eclipse. Nevertheless, in our example, we include them for cases where we might need DB2 specific functions, such as a Type IV connection test with Java.

9. To launch the Java job with the JZOS batch launcher, we modify the HelloWorld JCL in the jcl folder of the Eclipse workspace to look like the sample shown in Example 7-1.

Example 7-1 JCL to launch Java PDF Creator

```
//STRAUERA JOB
/*JOBPARM SYSAFF=SC48,L=9999
//PROCLIB JCLLIB ORDER=SYS1.IBM.PROCLIB
//HOLD OUTPUT JESDS=ALL,DEFAULT=Y,OUTDISP=(HOLD,HOLD)
//JAVA EXEC PROC=JVMPRC50,
// JAVACLS='com.ibm.itso.sample.InvoiceCreator',
// ARGS='/u/strauer/jzos/pdf'
//STEPLIB DD DISP=SHR,DSN=DB9G9.SDSNLOAD
//          DD DISP=SHR,DSN=DB9G9.SDSNLOAD2
//STDENV DD *
#This is a shell script which configures
# any environment variables for the Java JVM.
# Variables must be exported to be seen by the launcher.

. /etc/profile
export APPL_HOME=/u/strauer/jzos
export JAVA_HOME=/usr/lpp/java/J5.0
export DB2_HOME=/usr/lpp/db2/d9gg/db2910_jdbc/classes

export PATH="$PATH":{"JAVA_HOME}"/bin:

LIBPATH="$LIBPATH":{"JAVA_HOME}"/bin
LIBPATH="$LIBPATH":{"JAVA_HOME}"/bin/classic
LIBPATH="$LIBPATH":"/usr/lpp/db2/d9gg/db2910_jdbc/lib"
export LIBPATH="$LIBPATH":

# Customize your CLASSPATH here
# Add application home directory and jars to CLASSPATH
for i in "${DB2_HOME}"/*.jar; do
    CLASSPATH="$CLASSPATH":"$i"
done
for i in "${APPL_HOME}"/*.jar; do
    CLASSPATH="$CLASSPATH":"$i"
done
export CLASSPATH="$CLASSPATH":

# Configure JVM options
IJO="-Xms16m -Xmx128m"
# Uncomment the following line if you want to debug the application
#IJO="$IJO -Xdebug -Xrunjdpw:transport=dt_socket,server=y,address=8000"
# Uncomment the following if you want to run with Ascii file encoding..
IJO="$IJO -Dfile.encoding=IS08859-1"
IJO="$IJO -Ddb2.jcc.ssid=D9G1"
export IBM_JAVA_OPTIONS="$IJO "

export JAVA_DUMP_HEAP=false
export JAVA_PROPAGATE=NO
export IBM_JAVA_ZOS_TDUMP=NO
//
```

The most important changes to this JCL include:

- Changing the JAVACLS to point to the main class `com.ibm.itso.sample.InvoiceCreator` and specifying the UNIX System Services directory where the PDF files are stored as `ARGS`
- Because we use the JDBC Type II driver to connect to DB2 with high performance cross-memory operations, including the `SDSNLOAD` and `SDSNLOD2` member in the `STEPLIB`
- Creating a new UNIX System Services variable, `DB2_HOME`, that points to the directory of the JDBC driver
- Because the JDBC Type II driver needs a native `*.so` file, adding the JDBC lib directory to the `LIBPATH`
- Adding a script that adds all JAR files in the `DB2_Home` directory to the class path
- If not specified by a properties file, adding the JVM `-Ddb2.jcc.ssid` option

Note: The specific directories shown in our examples might be different on your system. Consult with your system administrator and DB2 administrator to determine the appropriate values for `STEPLIB`, `LIBPATH`, and `CLASSPATH`.

10. Finally, deploy and submit the job with the Ant script by right-clicking **deploy.xml**. Then, select **Run As** → **Ant Build** as shown in Figure 7-9.

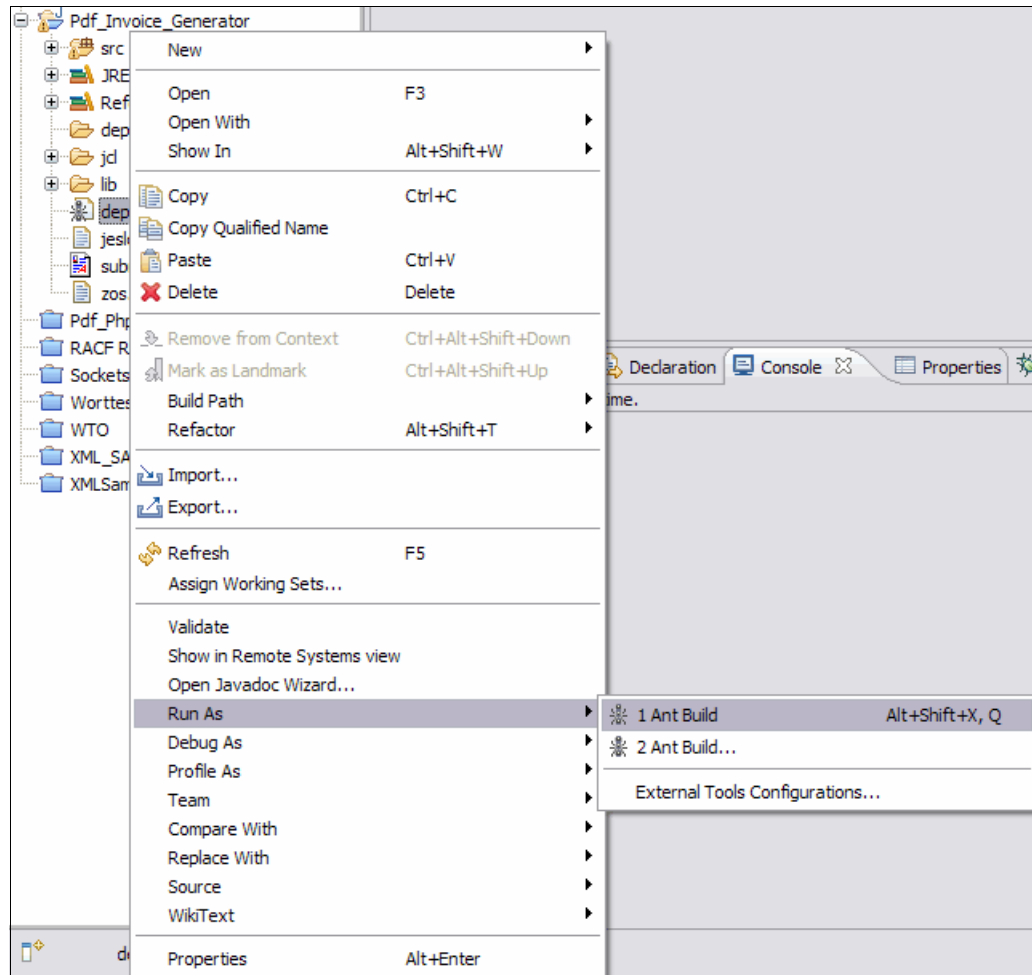


Figure 7-9 Eclipse Ant Deploy

The Ant script performs the following steps:

- a. Compiles all the Java files on the local workstation.
- b. Build a JAR file that contains all the created *.class files.
- c. Uploads the necessary JAR files in binary mode using FTP to z/OS UNIX System Services.
- d. Uploads the JCL that we customize before in Eclipse to z/OS in ASCII mode.
- e. Submits the job using FTP.
- f. Retrieve the output using FTP.

The output on the system console looks similar to that shown in Example 7-2.

Example 7-2 Java batch job output

```
[...]
JVMJZBL1023N Invoking com.ibm.itso.sample.InvoiceCreator.main()...
[java] JVMJZBL1024N com.ibm.itso.sample.InvoiceCreator.main() completed.
[java] JVMJZBL1021N JZOS batch launcher completed, return code=0
[java] !! END OF JES SPOOL FILE !!
[java] Connecting to DB2
[java] ... connected.
[java] Starting to create pdf file /u/strauer/jzos/pdf/Invoice_No_1.pdf...
[java] ... invoice pdf file created.
[java] Starting to create pdf file /u/strauer/jzos/pdf/Invoice_No_2.pdf...
[java] ... invoice pdf file created.
[java] Starting to create pdf file /u/strauer/jzos/pdf/Invoice_No_3.pdf...
[java] ... invoice pdf file created.
[java] Starting to create pdf file /u/strauer/jzos/pdf/Invoice_No_7.pdf...
[java] ... invoice pdf file created.
[java] Starting to create pdf file /u/strauer/jzos/pdf/Invoice_No_8.pdf...
[java] ... invoice pdf file created.
[java] !! END OF JES SPOOL FILE !!
[...]
```

11. Now, check the results and open one of the created PDF files. Figure 7-10 shows one of the batch-created PDF files from our test environment.

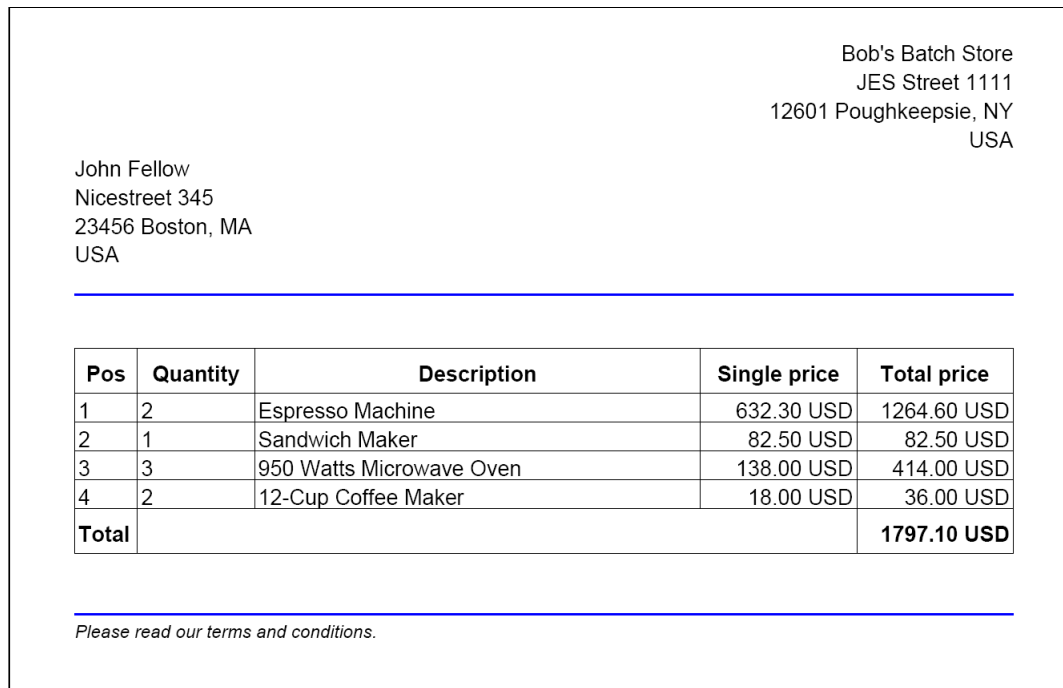


Figure 7-10 PDF invoice created in Java

12. Finally, we can implement another job in Java that takes these PDF files and sends them to the appropriate recipients using e-mail.

In summary, this sample application shows an easy and effective method to use Java with the JZOS batch launcher to enrich an existing batch environment with new functionality.



Implement new functionality using Java in WebSphere XD Compute Grid

Most of the batch workloads running on z/OS today are written in traditional languages such as COBOL or PL/I. Many of these workloads contain complex business logic and process a very large amount of input and output data.

Today, we can deploy Java batch applications on z/OS in a traditional batch run time, such as IMS or DB2, or as stand-alone batch, for example with JZOS as described in Chapter 7, “Implement new functionality using stand-alone Java” on page 77. However, depending on the environment, there can be disadvantages, such as a JVM is created for every batch job, which increases the CPU usage, consumes additional resources, and increases the overall cost of implementing a Java batch application as opposed to implementing the same application in a more traditional language.

One of the ways to deploy a Java batch application is in the UNIX System Services shell. The z/OS UNIX System Services does support shell scripts and a *cron* facility, which you can use to set up scheduled background processing. However, this facility lacks robust monitoring and administration capabilities like Tivoli Workload Scheduler. This situation gets much better in the case where a Java batch program is run inside a JCL managed by a workload scheduler, such as Tivoli Workload Scheduler. However, with WebSphere Application Server for z/OS container server services, Java fits very well into the On Line Batch Processing paradigm on z/OS. WebSphere Extended Deployment (XD) Compute Grid for z/OS, a batch container for Java EE applications, extends the existing WebSphere core functions and is easily accessible using the same administration console as WebSphere Application Server for z/OS.

With the Java EE platform as the core foundation for WebSphere Application Server for z/OS, WebSphere XD Compute Grid delivers the following improved qualities:

- ▶ Business flexibility delivers the core capabilities to more efficiently and effectively support both Java batch and OLTP workloads concurrently.
- ▶ Processing enables centralized management for a heterogeneous IT infrastructure that consists of both WebSphere and non-WebSphere application servers.
- ▶ Extended manageability offers simpler and improved management of complex system operations with real-time visualization tools, application versioning, and gradual, controlled implementation of autonomic capabilities such as health management. This manageability helps to reduce the cost and complexity of managing WebSphere's IT resources.
- ▶ Dynamic operations enable the Java EE application environment to support a focused configuration of WebSphere resources, and helps to increase the speed at which a company can adapt to business change.
- ▶ High-performance computing optimizes the performance of business-critical applications to support near-linear scalability for high-end transaction processing. Improves customer service levels while also taking advantage of existing Java skills and resources.

This chapter includes the following topics:

- ▶ Java and Java Platform, Enterprise Edition
- ▶ The Java EE runtime environment on z/OS
- ▶ WebSphere XD Compute Grid overview
- ▶ Quality of service in a WebSphere environment on z/OS
- ▶ Interoperability with other languages
- ▶ Batch programming using WebSphere XD Compute Grid
- ▶ Developing a WebSphere batch application
- ▶ Summary

8.1 Java and Java Platform, Enterprise Edition

The Java Platform, Enterprise Edition (Java EE) standard, as implemented by today's application server products, has emerged as a very popular model and environment for developing and deploying modern, enterprise-class business applications, especially OLTP applications. As a result, the Java EE skill base is rapidly growing in the industry, making the Java language and Java EE application servers an interesting and obvious target to host other types of enterprise business workloads, such as batch and compute-intensive applications.

In general, it is comparatively less expensive to achieve an equivalent level of application development competency with Java as with older languages. The object oriented nature of the language enables rapid development cycles. In addition ISVs provide a large portfolio of powerful Java applications and frameworks. Furthermore, JVM performance is continually increasing. This marks Java as the language of choice for the development of new business applications or a target language for the re-engineering of existing applications.

Additionally, the Java EE standard as implemented by today's application server products has emerged as a very popular model and environment for developing and deploying modern, enterprise-class business applications, especially OLTP applications. As a result, the Java EE skill base is rapidly growing in the industry.

8.2 The Java EE runtime environment on z/OS

The Java EE environment on z/OS as implemented by WebSphere Application Server for z/OS provides a natural home for deploying enterprise-class business applications written in Java. Figure 8-1 illustrates a simplified overview for OLTP workload in a WebSphere Application Server for z/OS environment, where Java EE-based applications are facilitated and managed as transaction-oriented applications. The user sends a request from a client over HTTP or IIOp, the business logic written in Java is invoked, and a response is sent to the user. All the transactional processing is done by WebSphere Application Server for z/OS.

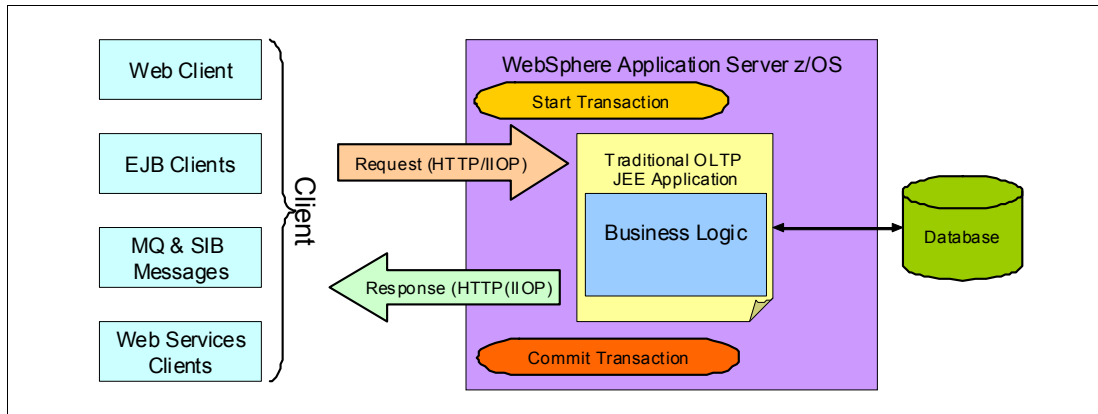


Figure 8-1 OLTP workload in a Java EE runtime environment on z/OS

WebSphere provides many of the overall system services (that is container services) that are required by complex business applications. Some typical container services are security, transaction support, Web services, session pooling, connection pooling, caching, and messaging, just to name a few. All this leads to a simplified Java application design, accelerates development time, simplifies management, and increases the application's overall security, availability, performance, and stability.

Unfortunately, the Java-based business logic residing within WebSphere Application Server for z/OS is not efficiently accessible from traditional batch. Figure 8-2 shows that traditional runtime environments, such as IMS, CICS, and TSO, exist in addition to the OLTP environment that is based on WebSphere Application Server. Integration between the traditional runtime environments and WebSphere Application Server is based on connectors and messaging.

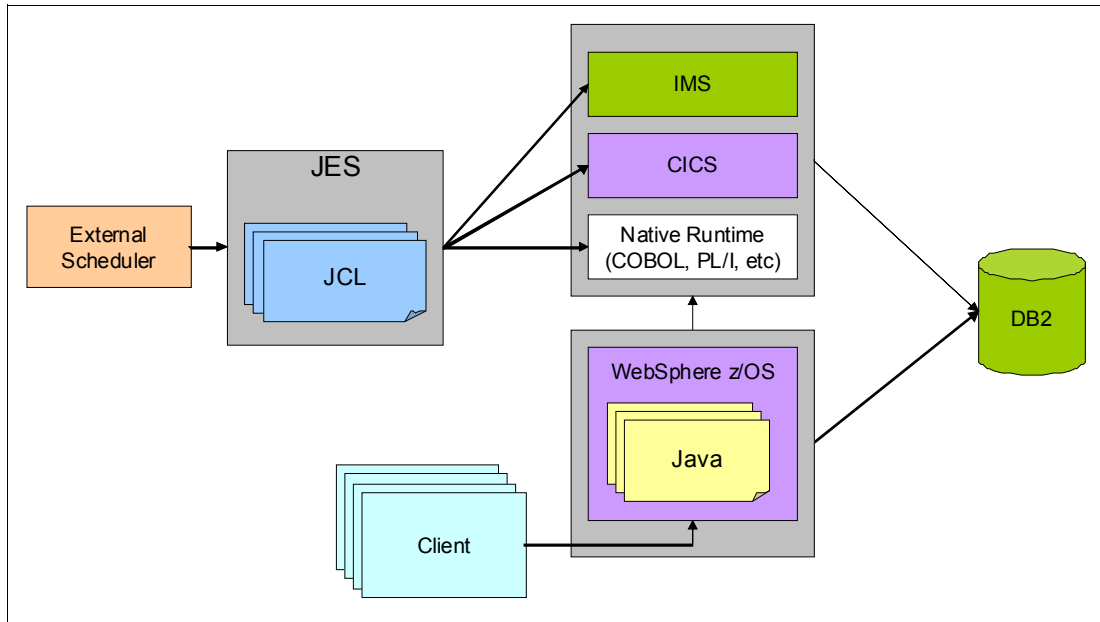


Figure 8-2 Traditional batch and OLTP workload in WebSphere Application Server for z/OS today

As mentioned previously, WebSphere XD Compute Grid introduces a new type of workload to the Java EE world: Java EE-based batch workloads that can run concurrently similar to the way this is done in traditional batch on z/OS. This solution expands on existing z/OS support for WebSphere, such as virtualization, goal-directed policy, and the capability to run both Java EE-based transactional and the Java EE-based batch workloads.

With WebSphere XD Compute Grid as a batch container, the business logic in WebSphere Application Server for z/OS, that is Java EE applications, is accessible by the traditional batch world on z/OS as shown in Figure 8-3 on page 97. Java-based business logic residing within WebSphere can be seamlessly integrated into the traditional batch execution environment and so the business logic can be shared across both the batch and OLTP paradigms.

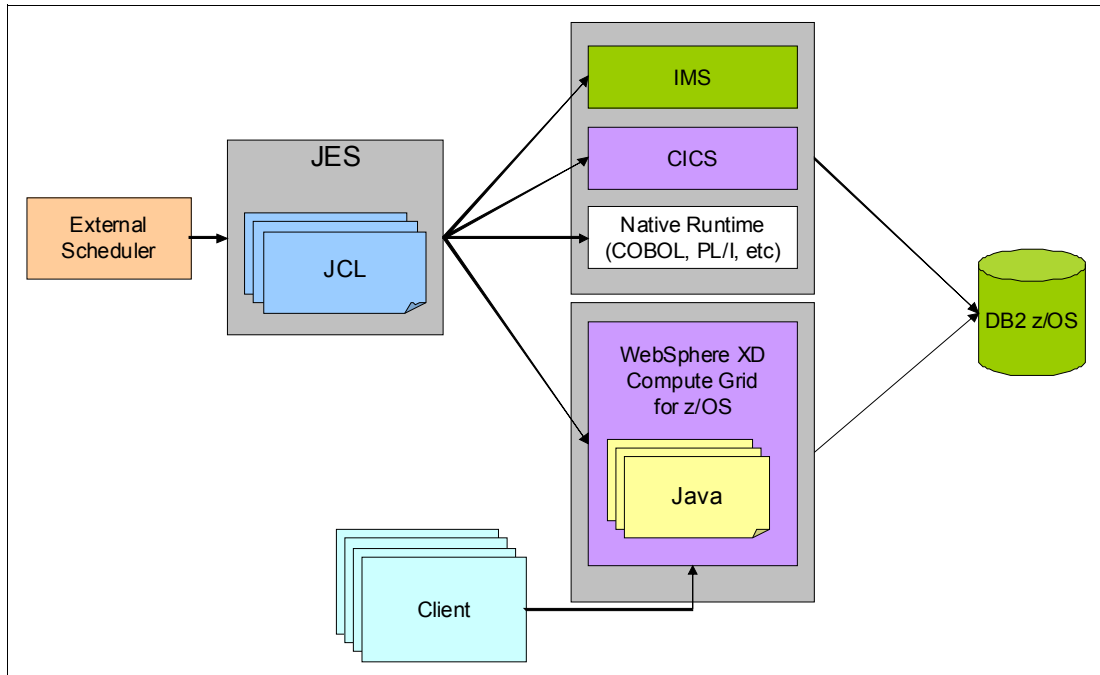


Figure 8-3 Batch and OLTP workload in WebSphere XD Compute Grid on z/OS

In contrast to the real-time, request-and-response behavior of typical transactional WebSphere applications, batch workloads require dedicated computing resources for longer periods of time. Traditionally, the processing of Java EE based batch workloads had to occur either in a separate, isolated environment, or executed during a specific time period (such as off hours), or both, so that it did not negatively impact the real-time processing of transactional requests. This separation of transactional and batch workloads resulted in a costly duplication of resources, with limited ability to share resources during the same periods. WebSphere XD Compute Grid supports two general types of long-running workloads in a Java EE application server:

- ▶ Batch
- ▶ Compute-intensive

With WebSphere XD Compute Grid for z/OS, however, these workloads can be dynamically balanced with transactional workloads by exploiting Workload Management (WLM).

This is the idea of a new kind of batch modernization using container managed batch where we can use the Java programming model for batch. Furthermore, with WebSphere XD Compute Grid for z/OS, you can make use of existing batch assets such as applications that are written in COBOL or PL/I, as well as the workload scheduler, JCL, and so forth. In addition, WebSphere XD Compute Grid for z/OS allows you to use the same business logic in both OLTP and batch contexts.

8.3 WebSphere XD Compute Grid overview

WebSphere XD Compute Grid for z/OS builds on top of the existing WebSphere Java EE programming model and container services by providing a job scheduler, an execution environment, and additional features that are designed specifically to provide for the execution and management of long-running batch applications. Just as WebSphere Application Server Network Deployment (ND) for z/OS provides a more natural environment

for hosting Java enterprise applications, specifically OLTP applications, WebSphere XD Compute Grid provides a more robust environment for deploying Java batch applications as shown in Figure 8-4.

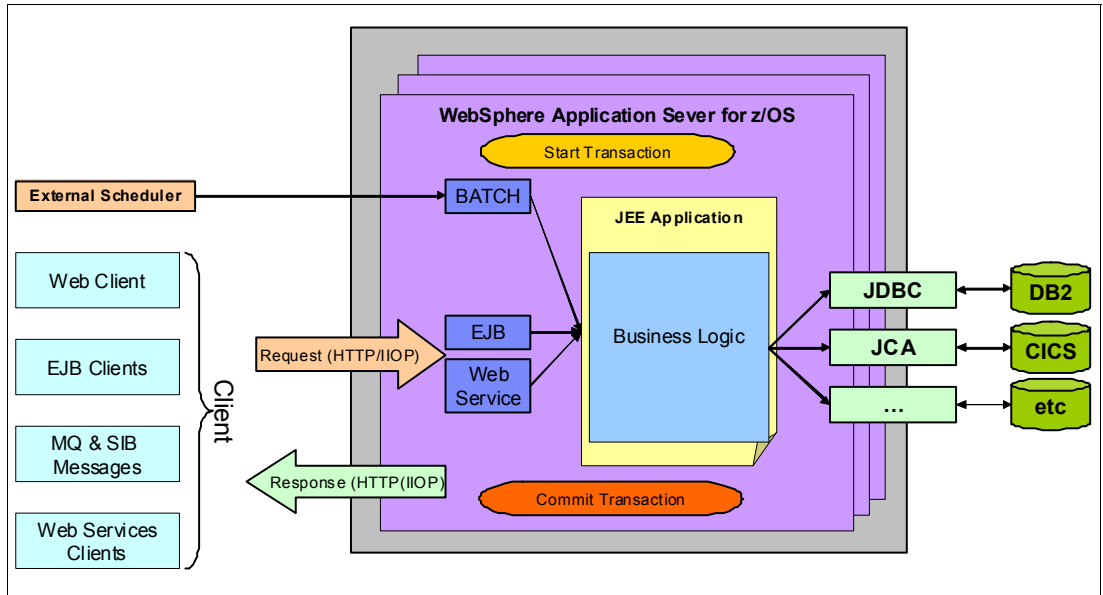


Figure 8-4 Batch and OLTP workload in a WebSphere environment

A WebSphere XD Compute Grid environment (as shown in Figure 8-5) is modeled after the z/OS Job Entry Subsystem (JES), which uses components such as Job Control Language (JCL), a job dispatcher, and job executors (JES initiators).

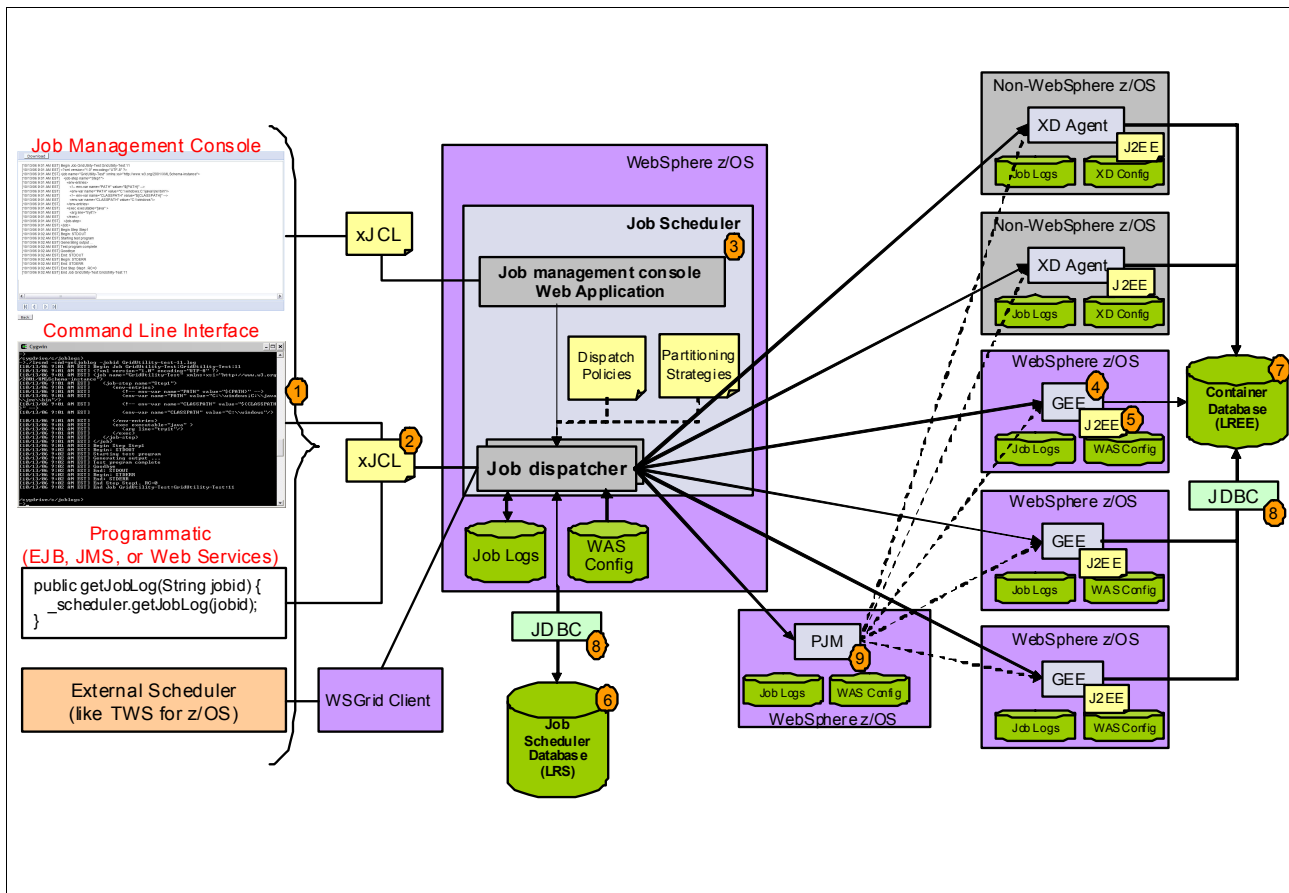


Figure 8-5 Container managed batch environment overview

WebSphere XD Compute Grid has an XML-based job description metadata called xJCL, a job dispatcher called the Job Scheduler (JS), and multi-threaded job executors called Grid Execution Endpoints (GEE). In addition, a Parallel Job Manager (PJM) component partitions and governs the execution of parallel batch jobs. Each of these WebSphere XD Compute Grid components are Java EE applications that can be placed in one JVM or in a fully clustered environment, that allow you to use WebSphere Application Server for z/OS security, scalability, high availability, and life cycle management best practices. The components of such a WebSphere XD Compute Grid environment are:

► Web, Shell, API

The job scheduler exposes the following API types to access its management functions:

- The Job Management Console (JMC), which is a Web interface
- The `trcmd` shell command
- APIs that are available as either Web Services and EJBs (see 14.2, “Using WebSphere XD Compute Grid trigger mechanisms” on page 234)

However, jobs can also be submitted to the system using an enterprise scheduler like Tivoli Workload Scheduler for z/OS (see 14.3.2, “Integrate Tivoli Workload Scheduler for z/OS with WebSphere XD Compute Grid” on page 266).

- ▶ xJCL

Jobs are described using a *job control language*. Compute Grid jobs use an XML based job control language. The job description identifies which application to run, its inputs, and outputs.
- ▶ Job scheduler

The job scheduler (JS) is the job entry point to XD Compute Grid that provides all job life-cycle management functions, such as submit, cancel, restart, and so forth, and the monitoring functionality. It maintains a history of all jobs, including those waiting to run, those running, and those that have already run. It also maintains usage data for jobs that have run. The JS dispatches workload to either the PJM or GEE and also hosts the JMC.
- ▶ Batch container

The batch container is called the Grid Execution Endpoint (GEE) component that executes the actual business logic of the batch jobs. Java EE-based batch applications run inside the WebSphere batch container. Native execution applications run inside a separate container, which is described in the following section. A WebSphere cell can have any number of batch containers.
- ▶ Java EE batch application

Java EE batch applications are regular WebSphere Java EE applications, deployed as an Enterprise Archive (EAR) file, that contain implementations of one or more Java batch applications. These Java batch applications follow either the transactional batch or compute-intensive programming models.
- ▶ Scheduler tables

The job scheduler uses a relational database (LRS) to store job information. It can be any relational database supported by WebSphere Application Server. If the job scheduler is clustered, the database must be a network database, such as DB2.
- ▶ Container tables

The batch container uses a relational database (LREE) to store checkpoint information for transactional batch applications. The database can be any relational database supported by WebSphere Application Server. If the batch container is clustered, the database must be a network database, such as DB2.
- ▶ JDBC

JDBC is standard JDBC connectivity to the scheduler and container tables, as supported by the WebSphere Application Server connection manager.
- ▶ Parallel Job Manager

The Parallel Job Manager (PJM) breaks large batch jobs into smaller partitions for parallel execution and provides job life-cycle management (submit, stop, cancel, and restart) for the single logical job and each of its partitions. The PJM component is not a required component in a WebSphere XD Compute Grid environment.

All of this means that a WebSphere XD Compute Grid environment supports today's batch processing needs, including:

- ▶ 24x7 batch processing, where batch can be executed concurrently with online transaction processing (OLTP)
- ▶ Sharing business services across batch and OLTP, where a service can be executed in multiple execution environments without sacrificing efficiencies, such as bulk-data processing

- ▶ Parallel-processing and caching features, where large problems can be partitioned, governed, and processed in parallel across a collection of server resources while hiding the complexities of multi-threading and management
- ▶ Container-managed batch qualities-of-service, such as checkpoint algorithms, restart mechanisms, multi-threading, and threshold policies, so the developer can focus on business logic
- ▶ Use application design patterns for building agile applications, where object-oriented design and service-orientation allow emerging middleware technologies, such as persistence and caching, to be adopted easily.
- ▶ Take advantage of the qualities-of-service of WebSphere Application Server z/OS, such as security, thread-pooling, connection-pooling, scalability and z/OS integration.

The platform includes also:

- ▶ Runtime components for submitting, dispatching, monitoring, and governing the execution of batch applications across a collection of resources
- ▶ Workload-management integrations for goals-oriented execution, where batch jobs can be throttled, paced, and reprioritized to meet batch and OLTP service-level agreements
- ▶ Operational control, external scheduler integration, and management, with reduced operational complexity for parallel processing, disaster recovery, and high availability
- ▶ End-to-end application development tooling for lightweight, POJO-based development, development frameworks and libraries, unit test, and application deployment.

Finally, the solution meets the requirements of heterogeneous enterprise environments by supporting:

- ▶ Platform-neutral batch applications that allow the location of the application's data to dictate the application deployment platform
- ▶ Standardized batch application architecture across platforms
- ▶ Standardized operational control and job management for all platforms, where the enterprise scheduler, in conjunction with WebSphere XD Compute Grid, can provide a common infrastructure for operational management; archiving, auditing, and log management; and scheduling for batch running on z/OS and non-z/OS platforms.

Compared to Java stand-alone applications, the WebSphere XD Compute Grid environment delivers a modern batch-processing platform for Java EE based applications. The JZOS technology delivers two technologies, the JZOS Launcher and the Java z/OS APIs. (See 7.2, "Running Java with JZOS" on page 79 for more information.) JZOS allows a stand-alone Java program to be invoked using JCL and also plays a role in a WebSphere XD Compute Grid environment. You can use the APIs, which we discuss in 5.2, "Special Java APIs for batch processing on z/OS" on page 51, from WebSphere batch applications. The APIs provide a strong integration point for Java and traditional z/OS as shown in Figure 8-6.

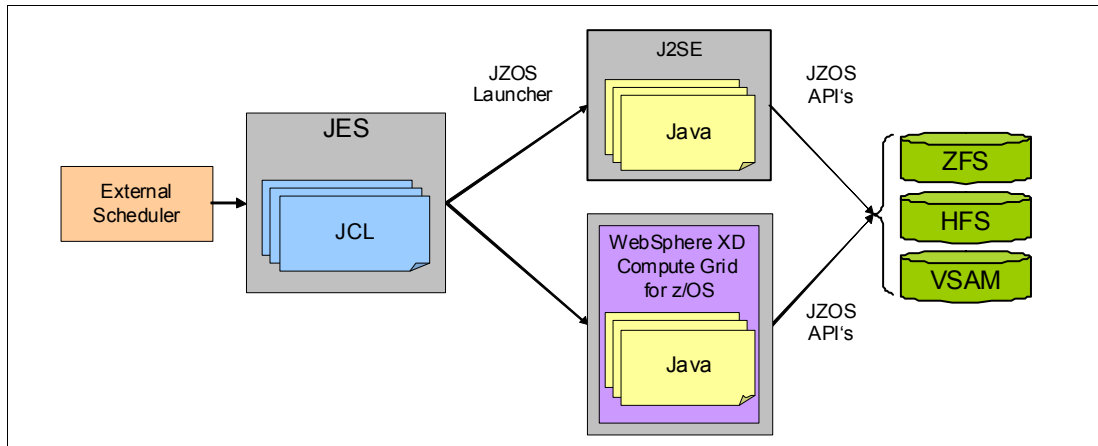


Figure 8-6 The role of JZOS in container managed batch

However, the JZOS launcher is not efficient for thousands of batch jobs that are run within a batch window, because a J2SE JVM lacks the following capabilities:

- ▶ No security, transaction, or connection management facilities
- ▶ No checkpoint or restart facility for batch jobs (which must be implemented by the Java program itself)
- ▶ No inherent high availability or other quality of service (QoS) that WebSphere Application Server for z/OS provides
- ▶ The JVM is not persistent or reusable (because each job step that uses Java needs to reload the JVM)

Compared to JZOS, WebSphere XD Compute Grid is built on WebSphere Application Server for z/OS and takes advantage of all necessary QoS and services provided by the WebSphere Application Server for z/OS run time such as:

- ▶ Security, transaction, and connection management
- ▶ Thread pooling
- ▶ High availability
- ▶ Persistent and reusable JVM and execution container

WebSphere XD Compute Grid is a simple Batch Programming Framework that allows developers to:

- ▶ Focus on business logic in batch applications, not on infrastructure support
- ▶ Can be used to incrementally modernize the existing batch environment
- ▶ Allows for co-locating batch program execution with OLTP execution (distributed) to better utilize application infrastructure and run batch and OLTP concurrently

WebSphere XD Compute Grid also supports a robust batch application infrastructure with the following capabilities:

- ▶ Application availability
- ▶ Prioritization
- ▶ Integration, particularly with z/OS WLM to enhance job execution and management

The WebSphere XD Compute Grid environment delivers the following tools for executing transactional Java batch applications:

- ▶ Container management checkpoint and restart capabilities
- ▶ Batch Data Stream (BDS) Management
- ▶ Parallel Job Execution (PJM)
- ▶ Operational control
- ▶ External scheduler integration (such as Tivoli Workload Scheduler for z/OS)
- ▶ SMF records for batch
- ▶ z/OS WLM integration

Using WebSphere for batch applications provides the following advantages:

- ▶ Batch modernization
Pursuing batch modernization projects has been of particular interest of z/OS customers. The purpose of these projects is to migrate from a native z/OS batch run time, typically developed in programming languages such as C, C++, PL/I, and COBOL, to Java.
- ▶ Highly parallel batch jobs
Taking advantage of the dispatcher-worker architecture of WebSphere XD Compute Grid to execute highly parallel batch jobs. A highly-parallel batch job is defined as a single, large batch job that can be broken into discrete chunks; each chunk can be executed concurrently across a grid of resources.
- ▶ Dynamic OLTP and batch run time
On z/OS, WebSphere XD Compute Grid integrates with WebSphere for z/OS, and the underlying z/OS operating system, and uses its inherent dynamic run time. For distributed platforms, WebSphere XD Compute Grid integrates with WebSphere XD Operations Optimization to deliver a virtualized and goals-oriented infrastructure.
- ▶ Batch as a service
WebSphere XD delivers features for resource usage accounting and reporting, on z/OS, these features integrate with the features already available on the platform (workload management, RMF, SMF, and so on).
- ▶ Replace existing batch frameworks
IT Operations, prior to the introduction of enterprise batch run times like WebSphere XD Compute Grid, took it upon themselves to build their own batch solutions. These infrastructures are being replaced with XD Compute Grid in an effort to eliminate code maintenance and execution costs.
- ▶ Sharing business logic across OLTP and batch
Business services, defined as discrete tasks responsible for executing some business function, can be shared across both the batch and OLTP run times.

8.4 Quality of service in a WebSphere environment on z/OS

WebSphere XD Compute Grid is designed to deliver a modern batch-processing platform for the enterprise. It supports today's batch processing needs, including 24x7 batch processing, shared business services across batch and OLTP, parallel-processing and caching features, and container-managed batch QoS. An important aspect of the WebSphere batch environment is also that it uses application design patterns for building agile applications and the QoS, such as security, thread-pooling, connection-pooling, high availability and scalability, and z/OS integration. We provide further detail in the sections that follow.

8.4.1 Security

Because of the rapid growth of e-business and the integration of different organizations, securing and managing (IT) infrastructures has become very complex and demanding. Protecting sensitive and confidential data from malicious intruders, deadly viruses, and worms is not an easy task. It requires constant monitoring of the daily IT business operations and deploying the latest security technology.

WebSphere XD Compute Grid is based on a WebSphere Application Server for z/OS environment. Thus, WebSphere XD Compute Grid uses the WebSphere Application Server for z/OS security and underlying operating system infrastructure security which can provide customers running enterprise applications with secure and reliable services, which are consistently managed by a security product such as RACF.

The security infrastructure of the underlying operating system provides certain security services to the WebSphere security application, including the file system security support to secure sensitive files in WebSphere product installation. The WebSphere system administrator can configure the product to obtain authentication information directly from the operating system user registry. On z/OS, WebSphere LocalOS registry uses the System Authorization Facility (SAF) implementation. SAF is a common set of APIs that allow a z/OS pluggable z/OS security manager, such as RACF or a third-party equivalent, to manage authentication and authorization needs on z/OS. These security products contain information about users, groups of users, resources, and user or group access to those resources. These products provide authentication and access control for the z/OS environment.

Note: You can find more information about WebSphere security at Version 7 in general in *WebSphere Application Server V7.0 Security Guide*, SG24-7660.

For more information about security in WebSphere on z/OS at Version 6.1, refer to *Security in WebSphere Application Server V6.1 and J2EE 1.4 on z/OS*, SG24-7384.

WebSphere XD Compute Grid security is based on the following techniques:

- ▶ WebSphere authentication for access to job scheduler interfaces. Users defined to the active WebSphere security registry can authenticate and gain access to the job scheduler's Web, command line, and programmatic interfaces.
- ▶ Role-based security for permission rights to job. Authenticated users must be assigned to the appropriate roles to perform actions against jobs. There are two roles:
 - `lrsubmitter`
Users in the `lrsubmitter` role can submit and operate on their own jobs but on no others.
 - `lradmin`
Users in the `lradmin` role can submit jobs and operate on their own or anyone else's jobs.

WebSphere XD Compute Grid roles are assigned through the job scheduler configuration page in the WebSphere administrative console.

8.4.2 High availability and scalability

High availability in this context means the ability of all WebSphere Application Server for z/OS components to be available nearly continuous at 99,999% for 365x24x7. If your workloads are predictable, system resources can be fairly static, but if workloads fluctuate and system resources are needed on-demand, then your environment should also be very scalable. Also,

the types of workload that are running on the system influence the system setup. If you want to run highly parallel jobs, if you have many concurrent jobs, or if you have a high submission rate of jobs, the Parallel Job Manager (PJM) is needed.

If high availability is needed, clustering of Compute Grid components is needed. Both the job scheduler and the batch container should be deployed to, and operated on, clusters to provide high availability. For all these requirements you have to set up a high available and very scalable WebSphere Application Server for z/OS batch environment for your Java EE based batch applications as shown in Figure 8-7.

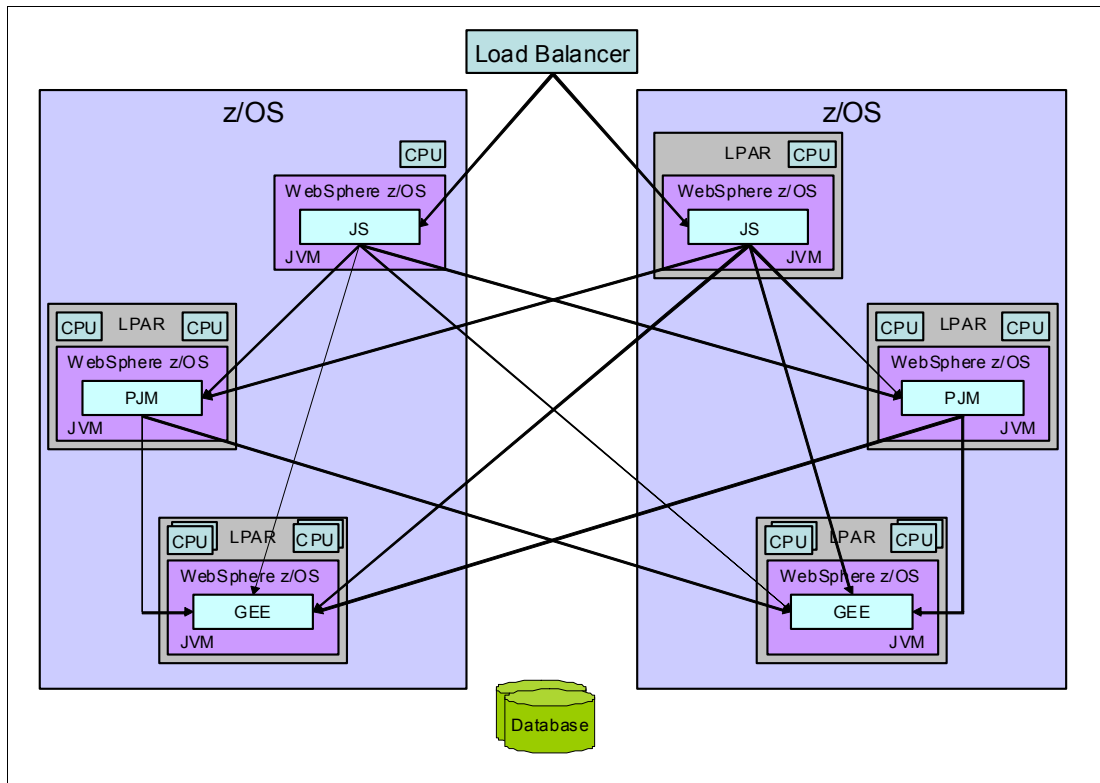


Figure 8-7 High availability and scalability in a WebSphere batch environment

Typical application clustering techniques should be used with the job scheduler to ensure it is highly available. The job scheduler (JS) supports multiple methods of access to its APIs, such as Web application, command line, Web service, and Enterprise JavaBean (EJB). Ensuring highly available network access to a clustered job scheduler depends on the job scheduler API access method. These choices include using the WebSphere plug-in or the On-Demand Router (ODR) from the WebSphere XD Operations Optimization feature. The batch container is made highly available simply by deploying it to a cluster. The job scheduler automatically recognizes the batch container is clustered and takes advantage of it to ensure a highly available execution environment for the batch jobs that run there.

The most available and scalable WebSphere Application Server for z/OS batch environment has redundant job scheduler, parallel job manager, and grid execution endpoints. Each component is clustered across two data centers. If you have a clustered job scheduler you have multiple active job schedulers and your jobs can be managed by any scheduler in your cluster environment. If your grid execution endpoints are clustered your batch applications are hosted also on clusters. Also, database sharing and shared file systems are necessary for a high available and scalable WebSphere Application Server for z/OS batch environment.

8.5 Interoperability with other languages

WebSphere Application Server for z/OS batch applications are Java EE-based applications. Thus, for interoperability with other languages such as COBOL and PL/I, you need to use native languages calls (as described in 5.5, “Java Interoperability with COBOL and PL/I” on page 55) or the functionality that is provided by the z/OS batch environment to use a mixed environment of Java and other languages in a Java EE-based batch application. See Figure 8-8.

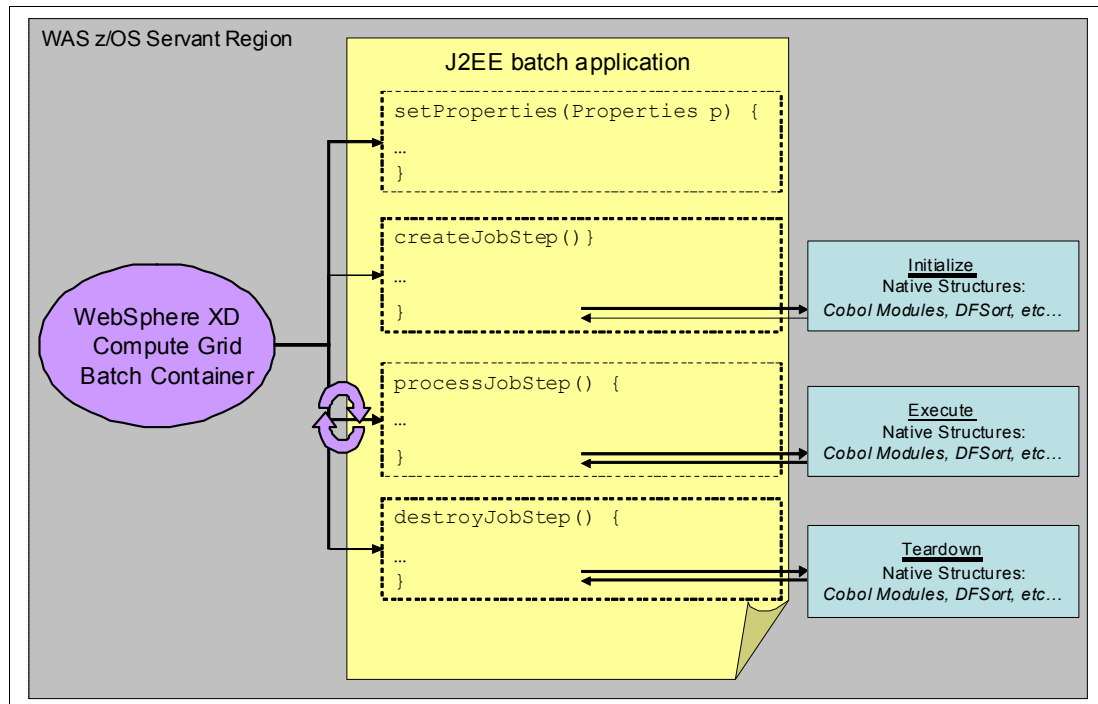


Figure 8-8 WebSphere XD Compute Grid Batch and traditional z/OS Interoperability

8.6 Batch programming using WebSphere XD Compute Grid

WebSphere XD Compute Grid provides the most complete enterprise Java batch programming solution available. WebSphere XD Compute Grid provides the following capabilities:

- ▶ A concise, powerful POJO-based programming model
- ▶ Simple packaging
- ▶ A simple deployment model
- ▶ Full-feature Job Control Language (JCL)
- ▶ A sophisticated job scheduler
- ▶ A robust execution environment
- ▶ Comprehensive workload management and administrative tools

The batch container operates with WebSphere Application Server for z/OS, which is a multi-machine configuration, but WebSphere XD Compute Grid also provides a unit test environment that you can run on a standalone WebSphere Application Server. WebSphere XD Compute Grid also offers an Eclipse-based development experience, and additionally supports Rational Application Developer or Rational Developer for System z as a full-function development environment.

At a high level, a batch job is a declarative construct that directs the execution of a series of one or more batch applications, and specifies their inputs and outputs. A batch job performs this set of tasks in sequence to accomplish a particular business function. Batch applications are programs designed to execute non-interactively in the background. Input and output is generally accessed as logical constructs by the batch application and are mapped to concrete data resources by the batch job definition.

Batch jobs commonly process large volumes of input/output data that are frequently record-oriented, usually representing critical business data. Business processing tasks performed by batch jobs can range widely. Batch jobs have been used in the System z environment for decades and continue as a backbone of many large and medium sized businesses to this day.

Figure 8-9 shows the basic anatomy of the elements in a batch job. The job definition describes the batch steps to execute and the sequence in which they run. Each step is defined with a particular batch application to invoke and its input and output data. Common sources and destinations for data include files, databases, transaction systems, message queues, and so on.

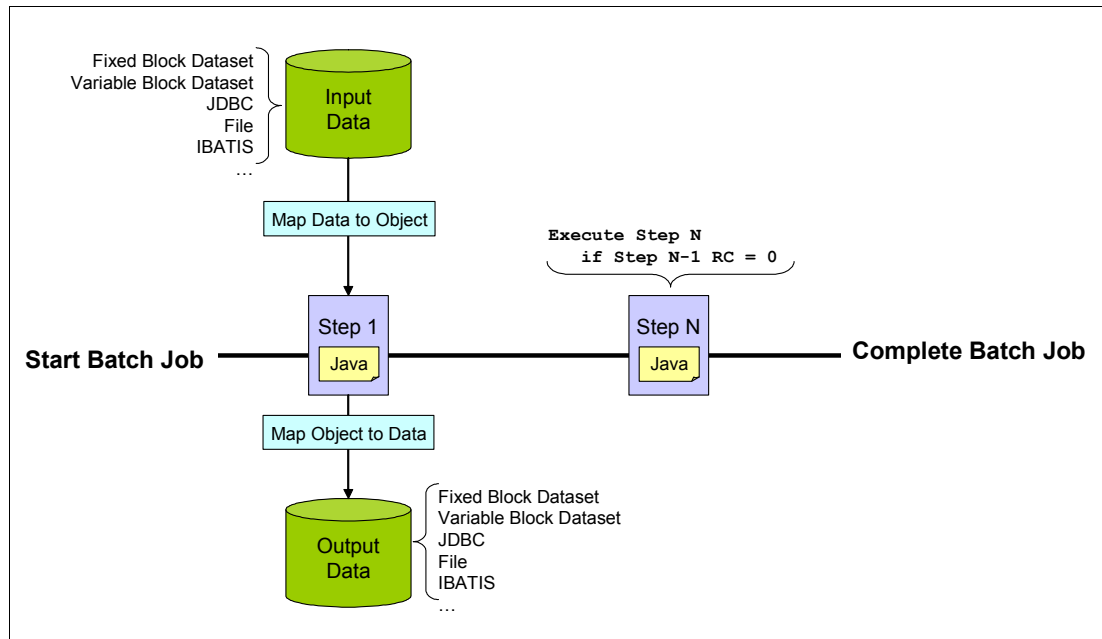


Figure 8-9 The basic flow of a batch application

WebSphere XD Compute Grid provides the following programming models:

- ▶ The *transactional* batch programming model requires the implementation of container-managed persistence entity bean (EJBs)
- ▶ In contrast, the *compute-intensive* programming model is implemented as a simple POJO and packaged into an enterprise archive (EAR) file for deployment into the WebSphere environment.

Note: WebSphere XD Compute Grid V6.1 extends the simple POJO style offered by the compute-intensive model to transactional batch. Transactional batch programs are also implemented as simple POJOs and packaged into EAR files for deployment. Here, we concentrate on building a POJO-based batch application because this method is the preferred method for building a batch application. For more information about the batch programming model, see the Information Center for WebSphere XD Compute Grid at: <http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r1m1/index.jsp>

A WebSphere XD Compute Grid batch application consists of a set of POJOs and runs under the control of the WebSphere XD Compute Grid batch container, which itself runs as an extension to a standard WebSphere Application Server. Figure 8-10 depicts the application components and their relationship to the batch container.

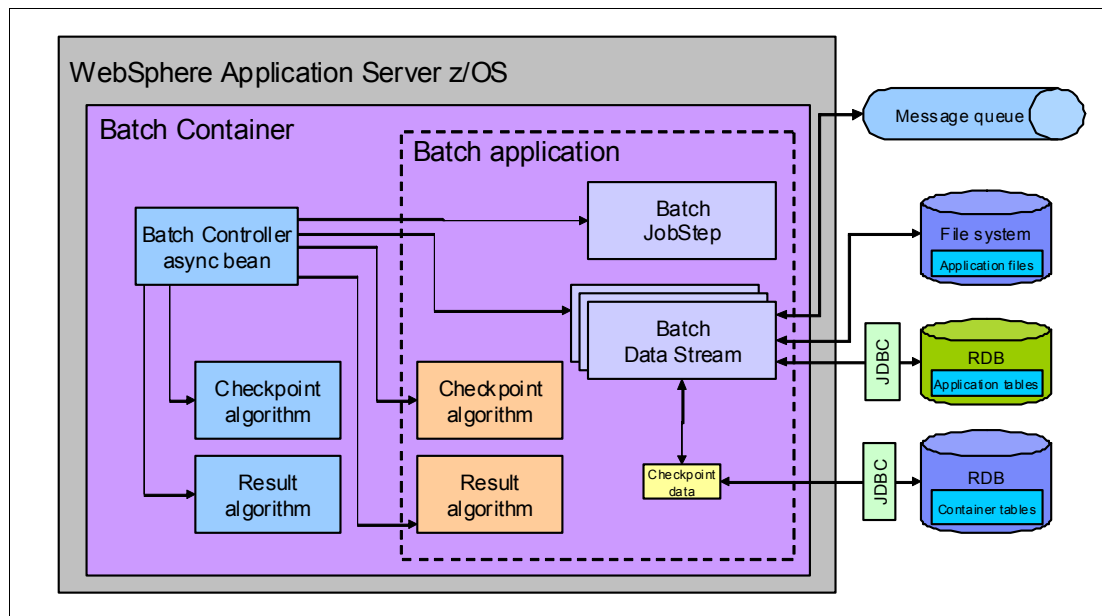


Figure 8-10 Batch programming model

The batch container runs a batch job under the control of an asynchronous bean, which you can think of as a container-managed thread. The batch container processes a job definition and carries out its life cycle, using an asynchronous bean as the unit of execution.

A batch application is made up of the following user-provided components:

- Batch job step This POJO provides the business logic to execute as a step in a batch job. The batch container invokes the batch job step during the course of processing a job definition.
- Batch data stream This POJO provides the batch job step with access to data. A batch application can be written to access one batch data stream or several. A batch data stream can be written to provide access to any sort of data, including data from RDBs, file systems, message queues, through J2C connectors, and so on.

The batch container is responsible for open, close, and checkpoint-related callbacks onto a batch data stream during the life cycle of a job step. The batch job step itself calls methods on the batch data stream to get and put data.

A batch application can optionally include these user-provided components:

- ▶ Checkpoint algorithm
 - The batch container provides a checkpoint/restart mechanism to support job restart from a known-point of consistency. A job might need to be interrupted and then subsequently restarted after a planned or unplanned outage. The batch container calls the checkpoint algorithm periodically to determine if it is time to take a checkpoint.
 - Compute Grid provides two pre-built checkpoint algorithms, one that supports a time-based checkpoint interval, and another that supports a checkpoint interval based on record-count.
- ▶ Results algorithm
 - Each batch job step supplies a return code when it is done. The results algorithm has visibility to the return codes from all steps in a batch job and returns a final, overall return code for the job as a whole.
 - WebSphere XD Compute Grid provides a pre-built results algorithm that returns the numerically highest step return code as the overall job return code.

The WebSphere XD Compute Grid batch programming model consists of the following principal interfaces, two of which are essential to building a batch application and two which are optional and intended for advanced scenarios:

- ▶ Essential interfaces
 - *BatchJobStepInterface* defines the interaction between the batch container and the batch application.
 - *BatchDataStream* abstracts a particular input source or output destination for a batch application and defines the interaction between WebSphere XD Compute Grid and a concrete *BatchDataStream* implementation.
- ▶ Optional interfaces
 - *CheckpointPolicyAlgorithm* defines the interaction between Compute Grid and a custom checkpoint policy implementation. A checkpoint policy is used to determine when Compute Grid will checkpoint a running batch job to enable restart after a planned or unplanned interruption.
 - *ResultsAlgorithm* defines the interaction between WebSphere XD Compute Grid and a custom results algorithm. The purpose of the results algorithm is to provide the overall return code for a job. The algorithm has visibility to the return codes from each of the job steps.

8.7 Developing a WebSphere batch application

With the basic concepts of a batch job and the WebSphere XD Compute Grid batch programming model, it is time to apply these concepts in a simple example that spotlights the essential batch interfaces, *BatchJobStepInterface* and *BatchDataStream*. The remainder of this section walks through the steps for implementing a sample batch job step and batch data stream using the Eclipse development environment, and testing them using a utility, called the *Batch Simulator*.

In this section, we show how to modernize an existing batch environment with a WebSphere XD Compute Grid application using the following steps:

1. Setting up the environment.
2. Creating a batch application using the BDS Framework.

3. Testing the batch application in Eclipse using the Batch Simulator, with a Unit Test Environment on the local machine, on z/OS with the Batch Simulator, and on WebSphere XD Compute Grid on z/OS.
4. Running Echo batch application on WebSphere XD Compute Grid z/OS.
5. Debugging your application in the Unit Test Server.
6. Reusing the Echo application for huge data processing using BDS.

8.7.1 Setting up the environment

WebSphere XD Compute Grid offers an Eclipse-based development experience, and additionally supports in the Rational Application Developer or Rational Developer for System z as a full-function development environment which helps you to develop Java EE-based batch applications. You can also install Eclipse V3.2 or higher, if you do not have it already installed.

Note: You can download all necessary material to create this example, such as the Eclipse workspace, from the Internet as described in Appendix C, “Additional material” on page 453.

Figure 8-11 shows the end-to-end application development tooling for lightweight, POJO-based development, the development frameworks and libraries, the functionality for unit testing, and application deployment.

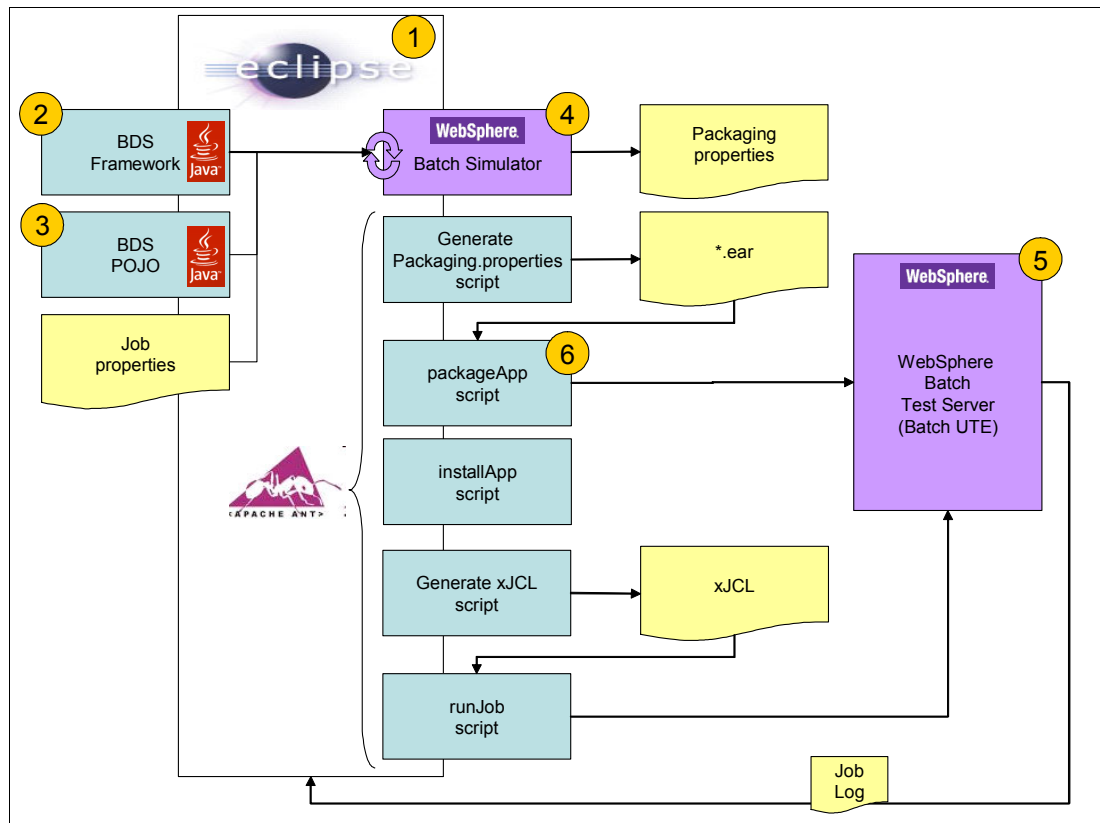


Figure 8-11 End-to-end development tooling for WebSphere batch applications

Referring to the numbers in Figure 8-11:

1. A Java IDE is used as development environment for WebSphere XD Compute Grid Batch. You can use your favorite Java EE application development workbench, but you just need to include a couple of libraries in the Java Build Path, package applications as normal Java EE applications (EAR files) and deploy them to a WebSphere XD Compute Grid batch runtime environment. You can also use Rational Application Developer or the WebSphere 6.1 Application Server Toolkit. The key benefit of using these tools is the integrated test environment which allows for testing of the application before deploying.
2. The Batch Data Stream (BDS) Framework is a development toolkit that implements the WebSphere XD Compute Grid interfaces for accessing common input and output sources such as files, databases, and so on. BDS acts as bridge between job business logic and the WebSphere XD Compute Grid programming model.
3. A POJO-based application development model. As of WebSphere XD Compute Grid, you only have to write POJO-based business logic. Tooling such as Eclipse or Rational Application Developer will generate the necessary WebSphere XD Compute Grid artifacts to run the application as shown in Figure 8-12.

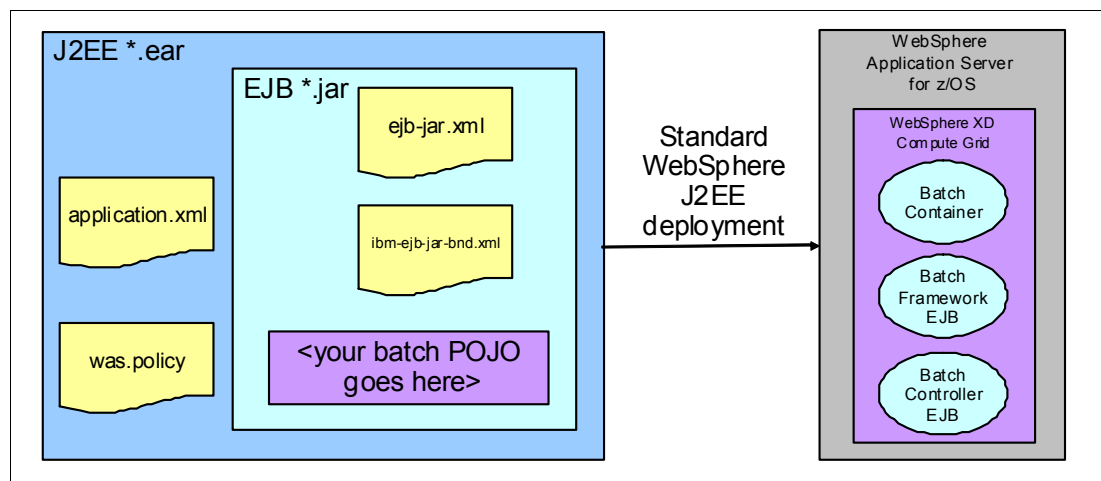


Figure 8-12 Simplified Batch Programming Model

4. The Batch Simulator is a light-weight, non-Java EE batch run time that exercises the WebSphere XD Compute Grid programming model. This runs in any standard Java development environment like Eclipse, and facilitates simpler application development because you are only dealing with POJOs and no middleware run time. The Batch Simulator is really for developing and testing your business logic. When your business logic is sound, you would execute function tests, system tests, and then deploy to production. You can download this from batch simulator download.
5. The Unit Test Environment (UTE) runs your batch application in a single WebSphere server that has the WebSphere XD Compute Grid run time installed. It is important to function-test your applications in the UTE to ensure that it behaves as expected when transactions are applied.
6. The Batch Packager is a utility that generates the necessary artifacts (for EAR creation) for deploying your POJO-based business logic into the WebSphere XD Compute Grid run time. The Batch Packager is a script that can be integrated into the deployment process of your application. It can also be run independently of the WebSphere run time, so you do not need any heavy-weight installs in your development environment.

To explain how to use the BDS Framework pattern and how to use the WebSphere XD Compute Grid Batch development Eclipse workspace that is delivered with this book, we use the simple Echo sample application for reading, processing, and writing the data.

Note: You can also download an Eclipse workspace from the WebSphere XD Compute Grid Wiki, which provides more than the Echo sample, from the following Web site:

<http://www.ibm.com/developerworks/forums/thread.jsps?threadID=228339&tstart=0>

The workspace is designed for batch application development and testing and is preconfigured with the BDS Framework, Batch Simulator, and Batch Packager utility (shown in Figure 8-11 on page 110).

Note: You can read more about the Compute Grid tools in the WebSphere XD Compute Grid Wiki at:

<http://www-128.ibm.com/developerworks/forums/thread.jsps?messageID=14038395�>

This project combines the WebSphere XD Compute Grid development tools into a ready-to-use environment. It includes sample applications and Ant scripts to install and run them in a WebSphere XD Compute Grid test server. The test server is optional and installed separately (see “Testing using the WebSphere XD Compute Grid UTE” on page 129 for more details).

The Echo application demonstrates a trivial batch application which we use for implementing all input and output patterns of the BDS Framework. This framework simplifies batch application development by allowing the developer to focus on writing core-business logic and not have to deal with the WebSphere XD Compute Grid specific programming model.

Quickstart and overview of the workspace

The workspace includes the Echo sample application and Ant scripts to help you take these applications through their basic life cycle. That life cycle can be summarized as:

1. Writing the code with help of Eclipse based tooling.
2. Initial testing using Batch Simulator.
3. Generating packaging properties.
4. Creating the install package.
5. Installing the package in UTE.
6. Generating xJCL.
7. Submitting the job.

Note: The joblogs are retrieved and stored back into Eclipse project.

Figure 8-11 on page 110 illustrates this life cycle.

To get started with the sample, download and decompress the `WXDCG_sample.zip` file and open the Eclipse workspace (`batchdevenv.6.1.0.3.1/workspace`). The workspace supplies source code and build/install/run Ant scripts for the Echo application.

Scripts are supplied for the following actions:

- ▶ Generate packaging properties
- ▶ Package application
- ▶ Install application in test server
- ▶ Generate job definition (xJCL)

► Run job in test server

The workspace includes the following directories:

<code>scr</code>	Sample application and workspace utility source code
<code>data</code>	Sample application input data and contains sample application output data after running the samples
<code>ear</code>	EAR files after packaging sample applications
<code>joblog</code>	Compute Grid job logs after running sample jobs
<code>lib</code>	POJO JAR files created during packaging process
<code>props.packaging</code>	Generated packaging properties. Packaging properties are generated by the Batch Simulator
<code>props.simulator</code>	Run properties for the Batch Simulator. One properties file is provided for each sample application
<code>script.ant</code>	Build/install/run Ant script for sample applications
<code>script.ant/config</code>	Environment configuration data to identify your WebSphere Application Server environment
<code>script.ant/imports</code>	Generic build/install/run Ant scripts
<code>script.wsadmin</code>	Generic WebSphere Application Server admin scripts
<code>tmp</code>	Scratch space used when deploying batch applications
<code>xJCL</code>	Generated xJCL (job definitions); xJCL files are generated by the Batch Simulator

The workspace is nearly ready to use when you first open it. In fact, no configuration is required if you intend only to execute the sample applications locally inside Eclipse using the Batch Simulator. Configuration is required only to use the WebSphere XD Compute Grid Test Server. To configure the Workspace to use the WebSphere XD Compute Grid Test Server, edit `scripts.ant/config/WASConfig.xml`. Read the comment block at the top of `WASConfig.xml` for instructions. Basically, you must specify a WebSphere Application Server profile directory and a WebSphere Application Server server host and port.

1. Open the `WASconfig.xml` file under **script.ant** → **config**.
2. Specify the following:
 - The WebSphere Application Server home installation directory, for example, `</WebSphere/AppServer/profiles/AppSrv01>`
 - The WebSphere Application Server host name, for example, `<localname>`
 - The WebSphere Application Server default port, for example, `<9080>`

See Figure 8-13.

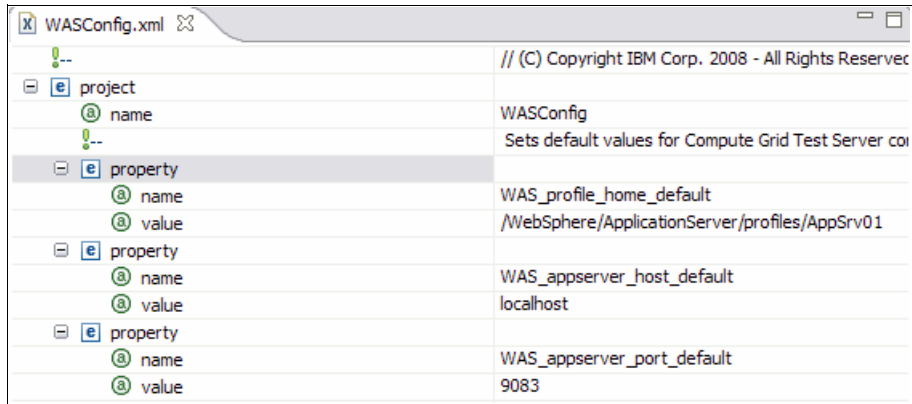


Figure 8-13 Configure the Workspace to use the Compute Grid Test Server, WASconfig.xml

8.7.2 Creating a batch application using the BDS Framework

Because the WebSphere XD Compute Grid batch framework provides only a BatchDataStream interface, it is useful to extend this framework with abstract classes that provide basic implementations of common patterns. The BDS framework provides an implementation for this model, called the GenericXDBatchStep, a simple step that uses one input and one output stream.

Rather than impose the burden of implementing the full WebSphere XD Compute Grid programming model on the developer, the BDS Framework takes a patterns approach and provides most of the implementation for the developer. The developer has to implement specific interfaces that correspond to the different patterns, for example, to map raw data to a domain object, to map a domain object to raw data, to apply some transformation to the input domain object to produce the output.

Figure 8-14 shows a summary of the BDS framework with the input and output pattern type.

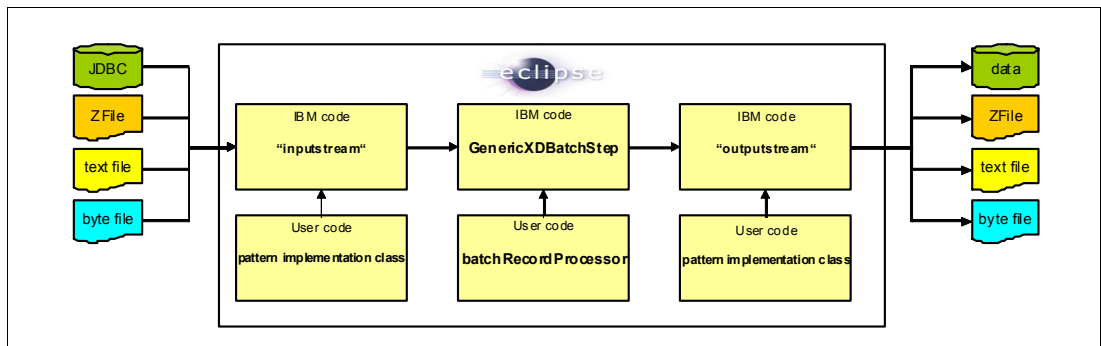


Figure 8-14 Summary of the BDS framework

In detail, the input and output pattern of the BDS framework are:

- ▶ JDBCReaderPattern and JDBCWriterPattern

Used to retrieve/write data from a database using a JDBC connection. The supported classes are:

- LocalJDBCReader / LocalJDBCWriter
- JDBCReader / JDBCWriter
- CursorHoldableJDBCReader

- ▶ `ByteReaderPattern` and `ByteWriterPattern`
Used to read/write byte data from a file. The supported class is `FileByteReader` / `FileByteWriter`.
- ▶ `FileReaderPattern` and `FileWriterPattern`
Used to read/write a text file. The supported class is `TextFileReader` / `TextFileWriter`.
- ▶ `RecordOrientedDatasetReaderPattern` and `RecordOrientedDatasetWriterPattern`
Used to read a z/OS data set. The supported classes are:
 - `ZFileStreamOrientedTextReader` / `ZFileStreamOrientedTextWriter`
 - `ZFileStreamOrientedByteReader` / `ZFileStreamOrientedByteWriter`
 - `ZFileRecordOrientedDataReader` / `ZFileStreamOrientedDataWriter`
- ▶ `JPARaderPattern` and `JPAWriterPattern`
Used to retrieve data from a database using OpenJPA and to write data to a database using a Java Persistence API (JPA) connection. The supported class is `JPARader` / `JPAWriter`.

Part of the workspace is a new enterprise application project which includes the necessary classes for mapping the input data to the domain object, processing the data and mapping the object data to the output data. The Echo sample is an easy application for processing input data into output data and includes the following necessary Java classes:

- ▶ A class to represent a record from the input stream `EchoDataHolder.java` in the `com.batch.domainobjects` folder as the domain object. This class is the object representation of a record in the batch input stream. This is constructed while reading a record from the input BDS, and passed on to the batch bean. This class is like a data or cargo bean that contains attributes and corresponding getters and setters, of the fields in the input stream.
- ▶ A batch input stream class `EchoReader.java` in the folder `com.batch.streams.inputstreams`. The key objective for this class is to map the raw data to a domain object. This class provides sample implementations for the following input patterns:
 - Reading text from a file (`FileReaderPattern`)
 - Reading rows from a database (`JDBCReaderPattern`)
 - Reading bytes from a file (`ByteReaderPattern`)
 - Reading bytes from a fixed-block MVS data set (`RecordOrientedDatasetReaderPattern`).

This class implements four different patterns (as described previously). Switching among input patterns is simpler from the xJCL. The developer is, therefore, responsible for mapping the raw data from whatever source to the domain object. The methods implemented in this class provide mappings from several input sources to an `EchoDataHolder` domain object. As a result, the source of the data can be specified in the xJCL as a change in the IBM-provided `BatchDataStream` implementation, and the developers implementation class can remain the same.

See Example 8-1.

Example 8-1 Batch input stream class

```
package com.ibm.websphere.batch.samples.echo.streams.inputstreams;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.IOException;
```

```

import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.util.HashMap;
import java.util.Properties;
import com.ibm.jzos.ZFile;
import com.ibm.websphere.batch.devframework.configuration.BDSFWLogger;
import com.ibm.websphere.batch.devframework.datastreams.patternadapter.ByteReaderPattern;
import com.ibm.websphere.batch.devframework.datastreams.patternadapter.FileReaderPattern;
import com.ibm.websphere.batch.devframework.datastreams.patternadapter.JDBCReaderPattern;
import com.ibm.websphere.batch.devframework.datastreams.patternadapter.RecordOrientedDatasetReaderPattern;
import com.ibm.websphere.batch.samples.echo.domainobjects.EchoDataHolder;

public class EchoReader implements FileReaderPattern, JDBCReaderPattern,
    ByteReaderPattern, RecordOrientedDatasetReaderPattern {
    ...
}

```

- ▶ A batch output stream class `EchoWriter.java` in the folder `com.batch.streams.outputstreams`. The key objective of this class is to map the domain object to raw data. This class provides sample implementations for the following output patterns:
 - Writing the domain object to a file (`FileWriterPattern`)
 - Writing the domain object to a database (`JDBCWriterPattern`)
 - Writing the bytes of a domain object to a file (`ByteWriterPattern`)
 - Writing the bytes of a domain object to a fixed-block MVS data set (`RecordOrientedDatasetWriterPattern`)

This class implements four different patterns (as described above). Switching among output patterns is therefore simpler from the xJCL. There might be different targets to which the data should be written. The domain object must be mapped to the various destinations. The developer is therefore responsible for mapping the domain object to the raw data format for the of the destination. The methods implemented in this class provide mappings from the `EchoDataHolder` domain object to several output sources. As a result, the destination for the data can be specified in the xJCL as a change in the IBM-provided `BatchDataStream` implementation, and the developers implementation class can remain the same.

See Example 8-2.

Example 8-2 Batch output stream class

```

package com.ibm.websphere.batch.samples.echo.streams.outputstreams;

import java.io.BufferedOutputStream;
import java.io.BufferedWriter;
import java.io.IOException;
import java.sql.PreparedStatement;
import java.util.Properties;
import com.ibm.jzos.ZFile;
import com.ibm.websphere.batch.devframework.configuration.BDSFWLogger;
import com.ibm.websphere.batch.devframework.datastreams.patternadapter.ByteWriterPattern;
import com.ibm.websphere.batch.devframework.datastreams.patternadapter.FileWriterPattern;
import com.ibm.websphere.batch.devframework.datastreams.patternadapter.JDBCWriterPattern;
import com.ibm.websphere.batch.devframework.datastreams.patternadapter.RecordOrientedDatasetWriterPattern;
import com.ibm.websphere.batch.samples.echo.domainobjects.EchoDataHolder;

public class EchoWriter implements FileWriterPattern, ByteWriterPattern,

```



```
JDBCWriterPattern, RecordOrientedDatasetWriterPattern {
    ...
}
```

- ▶ A batch step class Echo.java in the com.batch.steps package which corresponds to the basic flow echoReader --> Echo Step --> echoWriter. That means while not yet done processing all records in the input stream, read the next record from echoReader, pass the input record to the Echo step, which will apply some transformation to it, and write the transformed record to the output stream. A batch step, represented as a BatchJobStepInterface, whose task is to execute business logic against a stream of records.

See Example 8-3.

Example 8-3 Basic flow of a batch step class

```
while (!doneProcessingRecords) {
    inputRecord = getNextRecord();
    outputRecord = processRecord(inputRecord);
    writeOutputRecord(outputRecord);
}
```

The input batch data stream can be zero, one or n input or output streams. In the Echo example, there is one input stream reading from a file and one output stream writing to a file.

The job definition for the simple Echo batch application includes the following information:

- ▶ Job name and application name
- ▶ Controller JNDI name
- ▶ Utility jars
- ▶ Checkpoint implementation
- ▶ Declarations for the input/output stream and the data transformation

You have two options for your job definition to test your application:

- ▶ Write the xJCL
- ▶ Use a properties file and then generate the xJCL using a Ant script. The syntax is similar to the xJCL and shown in Example 8-4.

Example 8-4 Properties for Echo batch application

```
job-name=Echo
application-name=Echo

controller-jndi-name=ejb/com/ibm/ws/batch/EchoBatchController
#

utilityjars=../lib/batchframework.jar;../lib/ibmjzos-1.4.jar

checkpoint-algorithm=com.ibm.wsspi.batch.checkpointalgorithms.RecordbasedBase
checkpoint-algorithm-prop.recordcount=1000

#Input Stream declarations
bds.inputStream=com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteReader
bds-prop.inputStream.PATTERN_IMPL_CLASS=com.batch.streams.inputstreams.EchoReader
bds-prop.inputStream.FILENAME=${echo.data}/input.txt
bds-prop.inputStream.debug=false
bds-prop.inputStream.EnablePerformanceMeasurement=false
bds-prop.inputStream.EnableDetailedPerformanceMeasurement=false
```

```

#data transformation declarations

batch_bean-name=IVTStep1
batch-bean-jndi-name=ejb/GenericXDBatchStep
batch-step-class=com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep

#batch-bean-jndi-name=ejb/com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep

prop.BATCHRECORDPROCESSOR=com.batch.steps.Echo
prop.debug=false
prop.EnablePerformanceMeasurement=false
prop.EnableDetailedPerformanceMeasurement=false

#Output stream declarations
bds.outputStream=com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteWriter
bds-prop.outputStream.PATTERN_IMPL_CLASS=com.batch.streams.outputstreams.EchoWriter
bds-prop.outputStream.tablename=alg.tivpwx0
bds-prop.outputStream.FILENAME=${echo.data}/output.txt
bds-prop.outputStream.AppendJobIdToFileName=false
bds-prop.outputStream.EnablePerformanceMeasurement=false
bds-prop.outputStream.EnableDetailedPerformanceMeasurement=false
bds-prop.outputStream.debug=false

```

The WebSphere XD Compute Grid Batch Simulator includes an option for generating a properties file for the WebSphere XD Compute Grid Batch Packager utility. This utility creates a Java EE EAR file. To generate packaging properties using the Batch Simulator, simply invoke the simulator as part of the configuration in the Eclipse workspace, passing the simulator job properties, plus the `-writePackagingProps` flag, for example:

```

java com.ibm.websphere.batch.BatchSimulator EchoJobStep.props
-writPackagingProps

```

You can use the `generatePackagingProps.EchoBatchJobStep.xml` Ant script from the `script.ant` folder of the sample workspace to create the packaging properties for the EchoJobStep application, as shown in Figure 8-15.

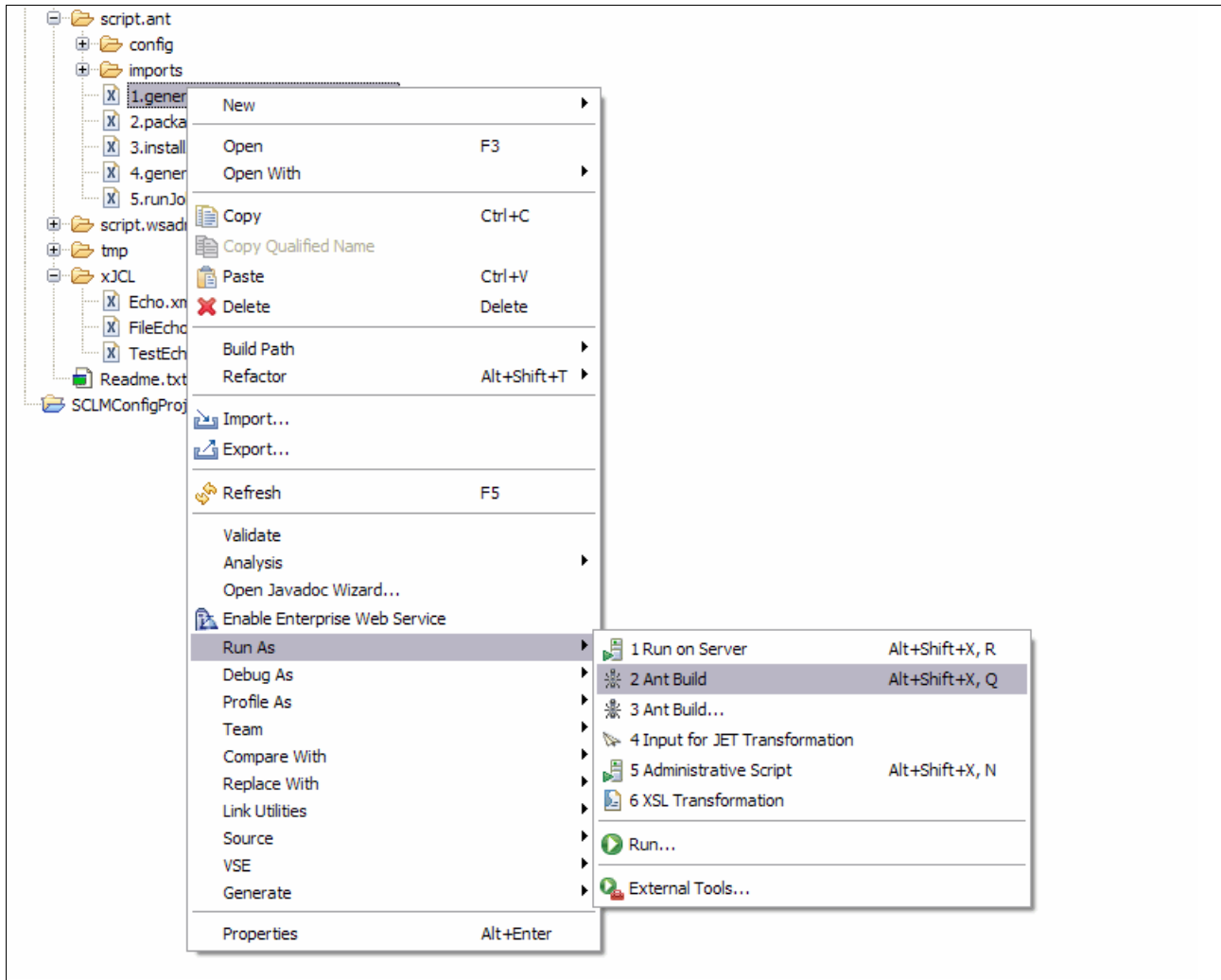


Figure 8-15 Generate packaging properties

The output looks as shown in Figure 8-16.

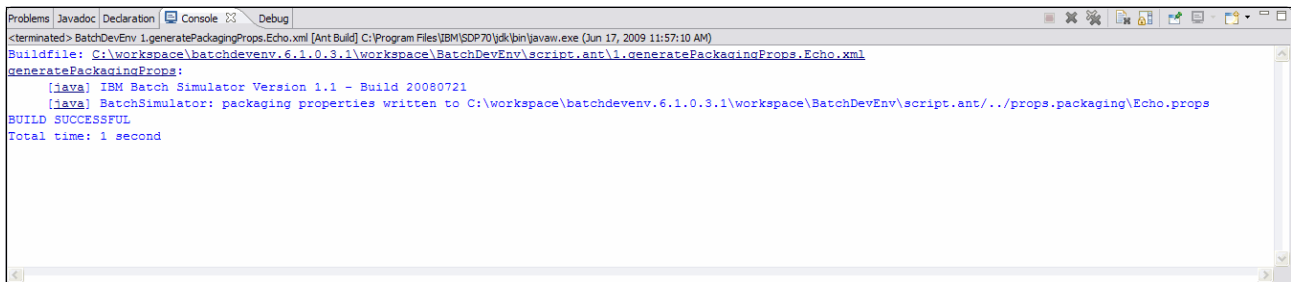


Figure 8-16 Output - Generate packaging properties

WebSphere XD Compute Grid includes a packaging utility called the *Batch Packager*. Because WebSphere XD Compute Grid batch applications run in WebSphere Application Servers, they are installed as Java EE EAR files. The Batch Packager handles many of the details surrounding the EJB deployment descriptor and other minor details of EAR file creation. The Batch Packager is driven by a properties file.

Use the packageApp.Echo.xml Ant script from the script.ant folder of the sample workspace to create the EAR file for the Echo application, as shown in Figure 8-17 on page 120.

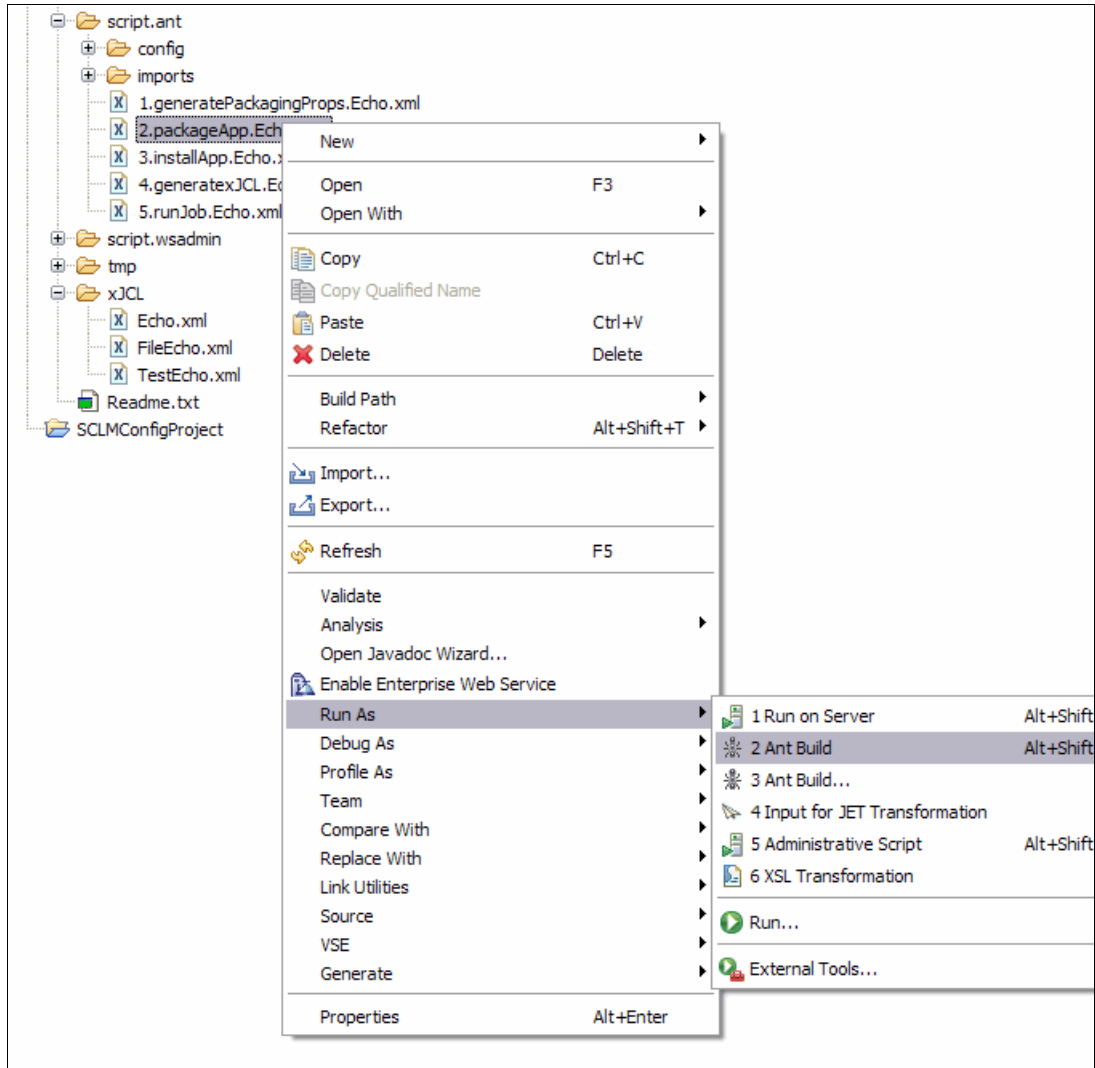


Figure 8-17 Package Compute Grid applications

The output is as shown in Example 8-5.

Example 8-5 Output - Package Compute Grid applications

```
Buildfile: C:\workspace\batchdevenv.6.1.0.3.1\workspace\BatchDevEnv\script.ant\2.packageApp.Echo.xml
jar:
[echo] Creating jar file
C:\workspace\batchdevenv.6.1.0.3.1\workspace\BatchDevEnv\script.ant\..\lib\Echo.jar
message:
[echo] Packaging Echo for Compute Grid Test Server
[echo] Test Server configuration: /WebSphere/ApplicationServer/profiles/AppSrv01
makeEAR:
[echo] Packaging batch artifacts from archive
C:\workspace\batchdevenv.6.1.0.3.1\workspace\BatchDevEnv\script.ant\..\lib\Echo.jar
[java] Application name : Echo
[java] Job level input JAR : ../lib/Echo.jar
[java] Output EAR file name : ../export/Echo
```

```
[java] Step Information : {IVTStep1, ejb/GenericXDBatchStep,
com.ibm.websphere.batch.devframework.steps.technologyadapters.GenericXDBatchStep, null, null}
[java] Utility Jar file(s) : ../lib/batchframework.jar;../lib/ibmzos-1.4.jar
[java] Packaging EAR file ../export/Echo ...
[java] Successfully packaged EAR file ../export/Echo.ear
getPlatform:
[echo] Running on windows
packageApp:
[echo] Running EJBDeploy ...
...
[exec] Invoking RMIC.
[exec] Invoking RMIC for all the ejb references.

Writing output file

[exec] Exporting archive 'Echo.deployed.ear'. Refreshing /Echo.ear.

Exporting archive 'Echo.deployed.ear'.
Exporting archive 'Echo.deployed.ear'. ibmzos-1.4.jar
...
Exporting archive 'Echo.deployed.ear'. META-INF/MANIFEST.MF
Exporting archive 'Echo.deployed.ear'. batchframework.jar
Exporting archive 'Echo.deployed.ear'.
Shutting down workbench.

[exec] EJBDeploy complete.0 Errors, 209 Warnings, 0 Informational Messages
...
run:
BUILD SUCCESSFUL
Total time: 39 seconds
```

If you are using a properties file instead of writing the xJCL, you can use an Ant script called generate.xJCL.Echo.xml, which is also provided in the workspace to generate the xJCL. See Figure 8-18.

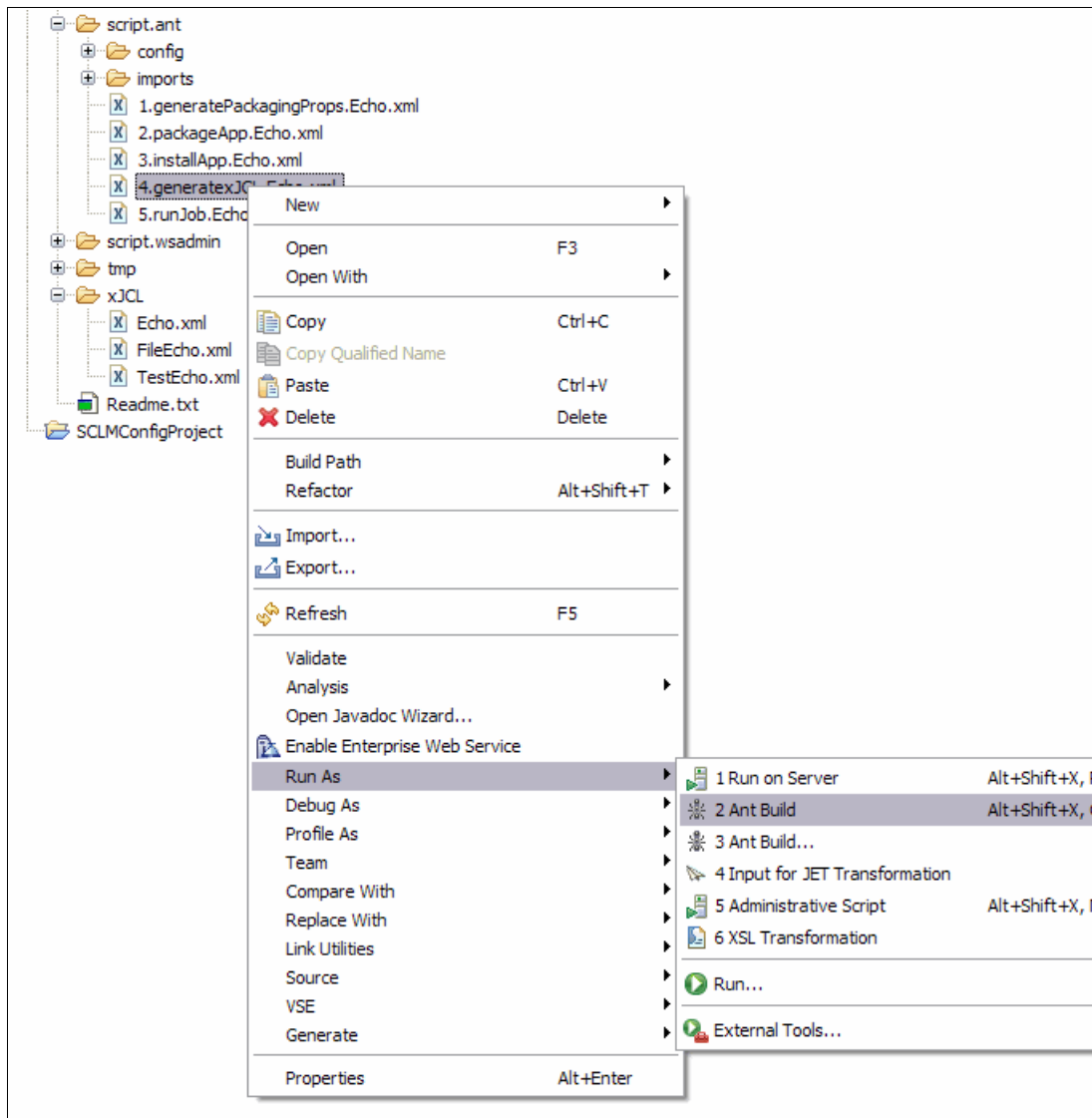


Figure 8-18 Generate xJCL

The output is shown in Figure 8-19.

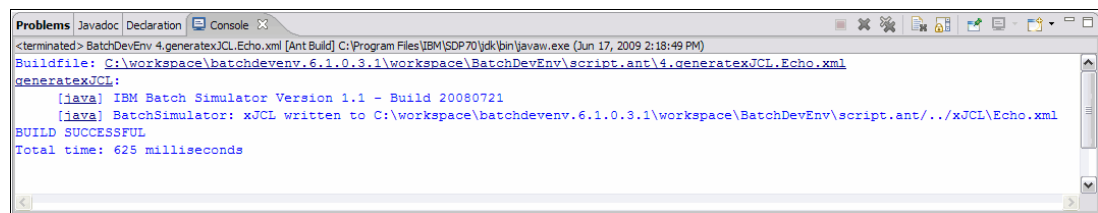


Figure 8-19 Output - Generate xJCL

If you are writing the xJCL by yourself, you have to specify for each step the input/output stream and the step you will use. The following code snippets explain each part in some more

detail. You can find the sample job in the xJCL folder (it is the generated xJCL from the properties file /props.simulator/Echo.props). It is built as follows:

1. You have to declare the `InputStream` that the `GenericXDBatchStep` will look up and resolve. See Example 8-6 for an excerpt and see `Echo.xml` in the xJCL directory of the workspace for the complete XML.

Example 8-6 Declaration of `InputStream` in xJCL

```
<job name="Echo" default-application-name="Echo" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
  <job-step name="Step1">
    ...
    <batch-data-streams>
      ...
      <bds>
        <logical-name>InputStream</logical-name>
        <props>
          <prop name="FILENAME" value="{echo.data}/input.txt"/>
          ...
          <prop name="PATTERN_IMPL_CLASS" value="com.ibm.websphere.batch.samples.echo.streams.inputstreams.EchoReader"/>
        </props>
        <impl-class>
          com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteReader</impl-class>
        </bds>
      </batch-data-streams>
    ...
  </job-step>
</job>
```

2. You have to declare the `OutputStream` that the `GenericXDBatchStep` will look up and resolve. See Example 8-7 for an excerpt and see `Echo.xml` in the xJCL directory of the workspace for the complete XML.

Example 8-7 Declaration of `OutputStream` in xJCL

```
<job name="Echo" default-application-name="Echo" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
...
  <job-step name="Step1">
    ...
    <batch-data-streams>
      <bds>
        <logical-name>OutputStream</logical-name>
        <props>
          <prop name="FILENAME" value="{echo.data}/output.txt"/>
          ...
          <prop name="PATTERN_IMPL_CLASS"
value="com.ibm.websphere.batch.samples.echo.streams.outputstreams.EchoWriter"/>
        </props>
        <impl-class>
          com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteWriter</impl-class>
        </bds>
      </batch-data-streams>
    ...
  </job-step>
</job>
```

3. You have to reference `GenericXDBatchStep` as the implementation for the job, as shown in Example 8-8.

Example 8-8 Reference the `GenericXDBatchStep`

```
<job name="Echo" default-application-name="Echo" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <jndi-name>ejb/com/ibm/ws/batch/EchoBatchController</jndi-name>
  <step-scheduling-criteria>
    <scheduling-mode>sequential</scheduling-mode>
  </step-scheduling-criteria>
  <checkpoint-algorithm name="chkpt">
    <classname>com.ibm.wsspi.batch.checkpointalgorithms.RecordbasedBase</classname>
    <props>
      <prop name="recordcount" value="1000"/>
    </props>
  </checkpoint-algorithm>
  <results-algorithms>
    <results-algorithm name="jobsum">
      <classname>com.ibm.wsspi.batch.resultsalgorithms.jobsum</classname>
    </results-algorithm>
  </results-algorithms>
  <job-step name="Step1">
    <jndi-name>ejb/GenericXDBatchStep</jndi-name>
    <checkpoint-algorithm-ref name="chkpt"/>
    <results-ref name="jobsum"/>
  ...
</job-step>
</job>
```

4. You have to specify your Batch Record Processor implementation as a step property, as shown in Example 8-9.

Example 8-9 Batch record processor implementation

```
<job name="Echo" default-application-name="Echo" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  ...
  <props>
    <prop name="EnablePerformanceMeasurement" value="false"/>
    <prop name="EnableDetailedPerformanceMeasurement" value="false"/>
    <prop name="debug" value="false"/>
    <prop name="BATCHRECORDPROCESSOR" value="com.ibm.websphere.batch.samples.echo.steps.Echo"/>
  </props>
</job-step>
</job>
```

Now you are finished with the application development, and you can start testing the batch application.

8.7.3 Testing the batch application

To test the batch application, you can use different options, such as batch simulation inside the Eclipse workspace or on z/OS by using a local WebSphere XD Compute Grid unit test environment. You can also use WebSphere on z/OS. In the following sections we discuss the different options for testing the batch application.

Testing using the Batch Simulator in the Eclipse workspace

As mentioned earlier, WebSphere XD Compute Grid provides a test environment that executes inside a standalone WebSphere Application Server, but to simplify things even further for this example (and avoid additional application packaging and deployment tasks), a testing aid called the Batch Simulator is included for your use. The Batch Simulator enables

you to initially test your batch POJOs right inside the Eclipse environment. Bear in mind, though, that the Batch Simulator runs in a J2SE environment, whereas the actual WebSphere XD Compute Grid execution environment is Java EE, because it runs inside WebSphere Application Server. Still, the Batch Simulator is useful for testing basic POJO applications and testing the essential business logic in the batch job step.

In the example the `batchsimulator.jar` file is included in the Eclipse workspace but this file is also part of the WebSphere Compute Grid installation and is located under, for example, `C:\IBM\WebSphere\AppServer\lib`.

For initial testing, with the batch application in your Eclipse-based workspace, you can use the Batch Simulator. To run the Echo sample with the Batch Simulator, you need to provide a job definition that describes the batch job step you want to execute and its batch data streams. A sample job definition that is ready to use for this sample batch job step can be found in the Eclipse workspace under `/props.simulator/Echo.props`.

To configure a specific run for an application using the Batch Simulator in Eclipse, you have to define that you want to run a java application. Add a name, for example, Echo, and add the project name, the main class, the arguments which are needed and Classpath. Figure 8-20, Figure 8-21, and Figure 8-22 show the configuration to run the Echo application using the Batch Simulator. This is already configured in the sample workspace.

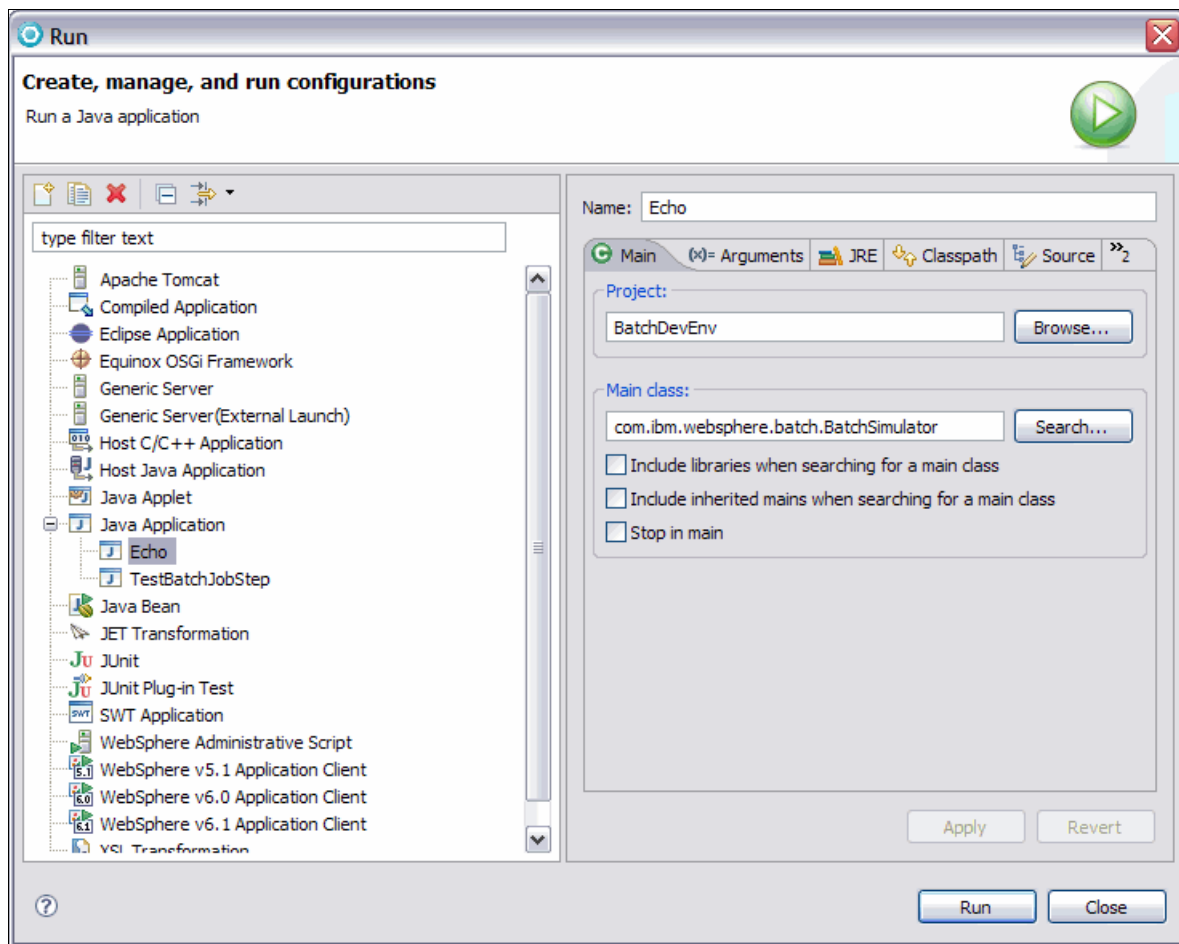


Figure 8-20 Run Echo with Batch Simulator - Main

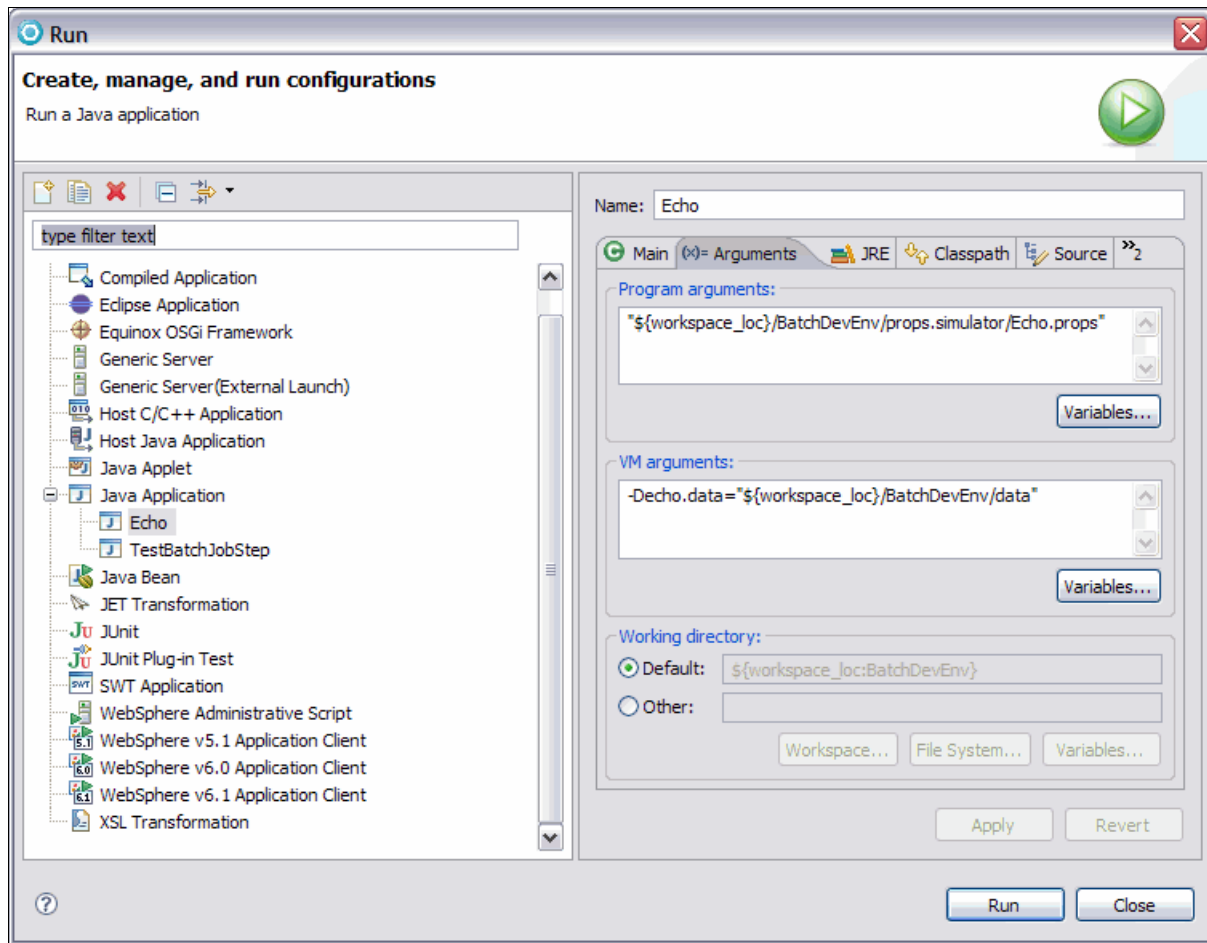


Figure 8-21 Run Echo with Batch Simulator - Arguments

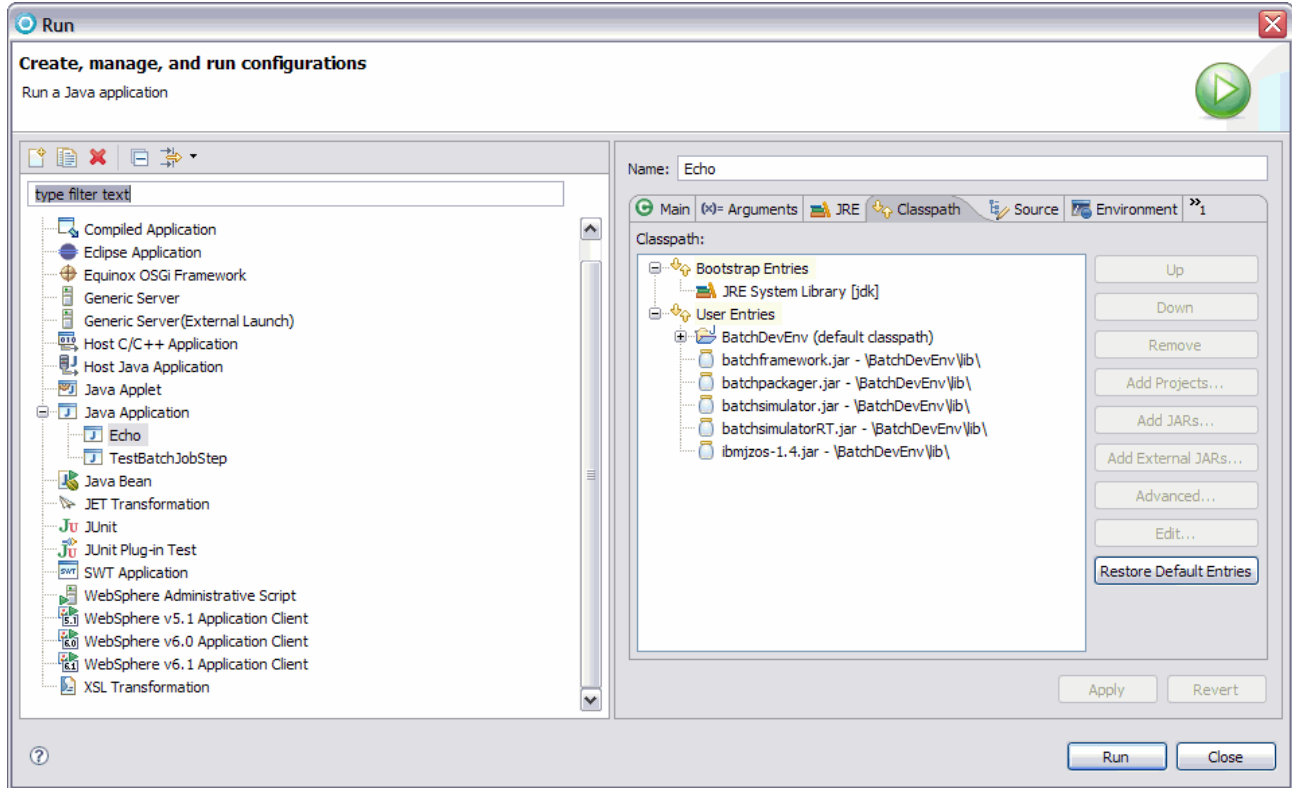


Figure 8-22 Run Echo with Batch Simulator - Classpath Definition

If you click Run Echo as shown in Figure 8-23, you receive the output as shown in Example 8-10 in the console from the Echo sample. In addition, the output file is created.

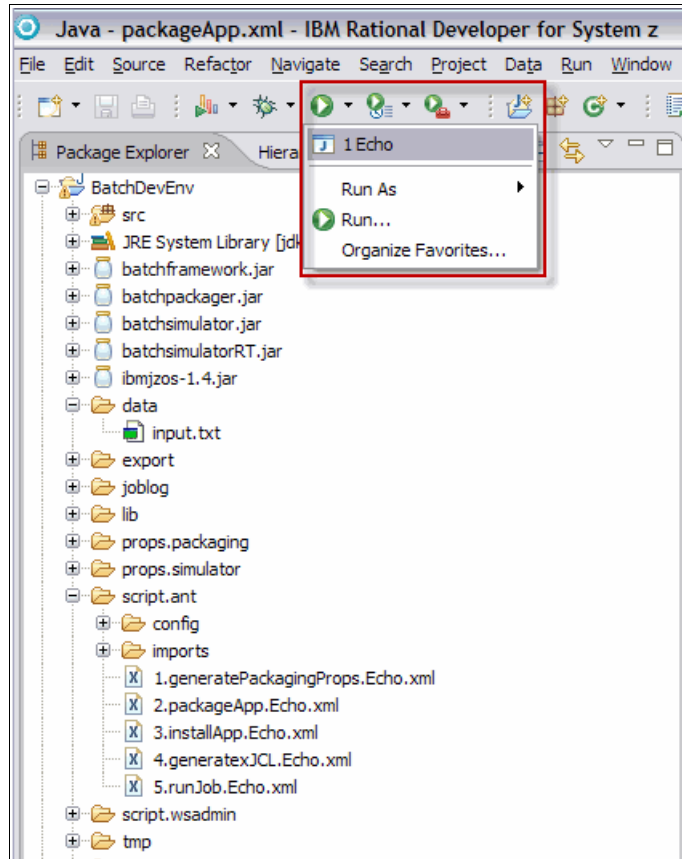


Figure 8-23 Using the Batch Simulator to run Echo application

Example 8-10 Output from running Echo application in Eclipse using the Batch Simulator

```
IBM Batch Simulator Version 1.1 - Build 20080721
BatchSimulator: start job Echo
BatchSimulator: outputStream checkpoint data: 0
BatchSimulator: inputStream checkpoint data: 0
BatchSimulator: outputStream checkpoint data: 80000
BatchSimulator: inputStream checkpoint data: 1000
BatchSimulator: outputStream checkpoint data: 160000
BatchSimulator: inputStream checkpoint data: 2000
BatchSimulator: outputStream checkpoint data: 240000
BatchSimulator: inputStream checkpoint data: 3000
BatchSimulator: outputStream checkpoint data: 320000
BatchSimulator: inputStream checkpoint data: 4000
BatchSimulator: outputStream checkpoint data: 400000
BatchSimulator: inputStream checkpoint data: 5000
BatchSimulator: outputStream checkpoint data: 480000
BatchSimulator: inputStream checkpoint data: 6000
BatchSimulator: outputStream checkpoint data: 560000
BatchSimulator: inputStream checkpoint data: 7000
BatchSimulator: outputStream checkpoint data: 640000
BatchSimulator: inputStream checkpoint data: 8000
BatchSimulator: outputStream checkpoint data: 720000
```

...
BatchSimulator: outputStream checkpoint data: 6160000
BatchSimulator: inputStream checkpoint data: 77000
INFO->jobid: Echo:GenericXDBatchStep.destroyStep()- Total Execution Time: 1094
BatchSimulator: end job Echo - RC=0

Testing using the WebSphere XD Compute Grid UTE

Before starting to work with WebSphere XD Compute Grid on z/OS, you can set up an environment for testing on your local machine. First, you have to install and configure the UTE, also known as *Compute Grid Test Server* or *Compute Grid UTE*.

Obtaining the Compute Grid UTE

The Compute Grid UTE is available from one of two sources:

- ▶ Compute Grid trial download which you can download from **developerWorks®** → **WebSphere** → **Downloads** → **Trial**. Select WebSphere Extended Deployment Compute Grid Version 6.1. The direct Web link is:

http://www.ibm.com/developerworks/downloads/ws/wscg/learn.html?S_TACT=105AGX10&S_CMP=ART

The recommended fix pack levels are:

- WebSphere Application Server 6.1.0.23, which is available at:

<http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg24022255>

- WebSphere Extended Deployment 6.1.0.5, which is available at:

<http://www-01.ibm.com/support/docview.wss?rs=3023&uid=swg24020719>

- ▶ UTE augment included in a licensed WebSphere XD Compute Grid product.

Setting up the Compute Grid UTE

The UTE can be setup in the following configurations:

1. Inside a trial version of WebSphere Application Server and using a trial download of UTE
2. Inside a licensed version of WebSphere Application Server and using a download of UTE
3. Inside Rational Application Developer or Rational Developer for System z using a download of UTE

The sample is based on configuration option #1 from the preceding list. The examples given are done using Windows XP. The steps for Linux are similar.

All the install and configuration steps for these environment are provided in the WebSphere Extended Deployment Compute Grid Wiki at:

<http://www.ibm.com/developerworks/forums/thread.jspa?threadID=227667&tstart=0>

Note: UTE is supported in standalone WebSphere Application Server application server only (or in the Rational Application Developer unit test server).

The WebSphere batch environment that we are using in the sample provides a single server UTE topology. The UTE is integrated with the product install, and a profile template, `wxdcgUTE_augment`, is provided for augmentation. Augmentation of this template configures the UTE on a standalone application server.

When the UTE is integrated during the product installation, all the necessary files are stored in the target profile. The selected application server is configured to host the Job Scheduler

and job execution environment. It also creates and configures local Derby databases to support the job components.

Figure 8-24 shows the UTE topology.

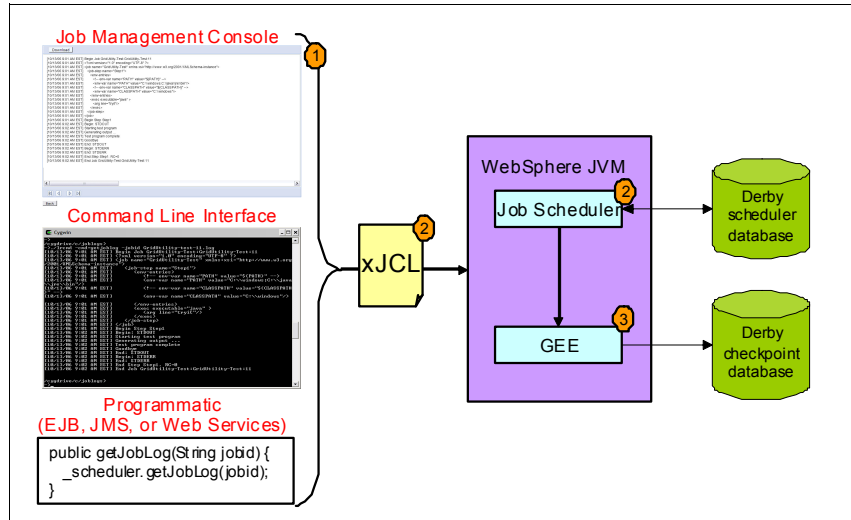


Figure 8-24 Sample WebSphere batch UTE

Using the Compute Grid UTE

The workspace also includes support for the Compute Grid Test Server. This support provides scripts for packaging, installation, and job submission of batch applications using the Test Server. The Test Server is not part of the workspace. You must install it separately as described in “Testing using the WebSphere XD Compute Grid UTE” on page 129.

Compute Grid batch applications run on WebSphere servers. You can install the `Echo.ear` file through regular WebSphere systems management interfaces, for example the Admin console or the `wsadmin` utility. Use the `installApp.Echo.xml` Ant script from the `script.ant` folder of the sample workspace to install the Echo application in the Compute Grid Test Server (UTE), as shown in Figure 8-25.

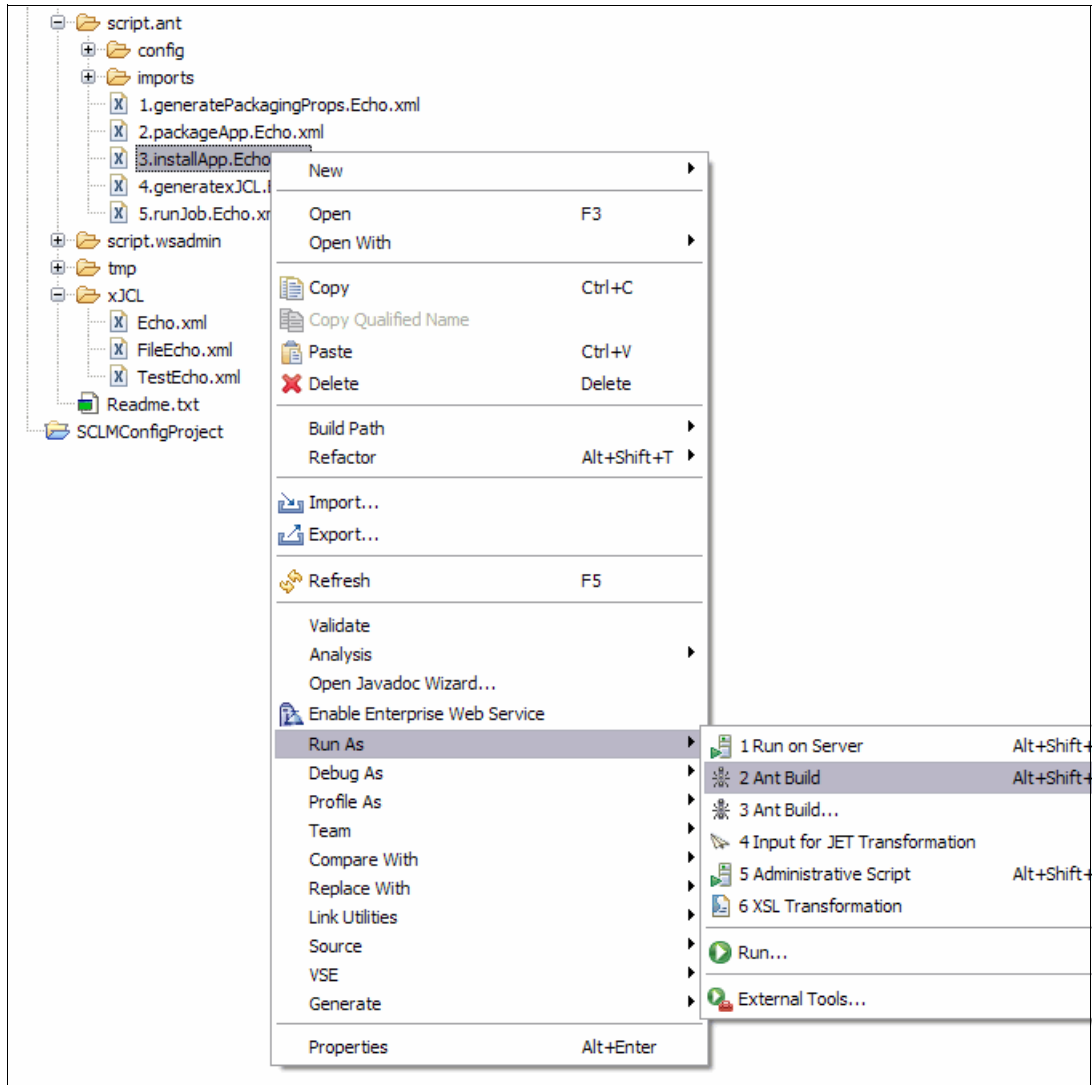


Figure 8-25 Install package in UTE

The output of the install process is shown in Example 8-11.

Example 8-11 Output - Install package in UTE

```
Buildfile: C:\workspace\batchdevenv.6.1.0.3.1\workspace\BatchDevEnv\script.ant\3.installApp.Echo.xml
installApp:
  [echo] Installing Echo in Compute Grid Test Server
  [echo] Test Server configuration: /WebSphere/ApplicationServer/profiles/AppSrv01
  [java] wsadmin
  [java] arg[0]=/WebSphere/ApplicationServer/profiles/AppSrv01
  [java]
arg[1]=C:\workspace\batchdevenv.6.1.0.3.1\workspace\BatchDevEnv\script.ant\..\script.wsadmin/installApp.py
  [java] arg[2]=Echo
  [java]
arg[3]=C:\workspace\batchdevenv.6.1.0.3.1\workspace\BatchDevEnv\script.ant\..\export\Echo.deployed.ear
  [java] Launched command /WebSphere/ApplicationServer/profiles/AppSrv01/bin/wsadmin.bat -f
C:/workspace/batchdevenv.6.1.0.3.1/workspace/BatchDevEnv/script.ant/..\script.wsadmin/installApp.py Echo
C:/workspace/batchdevenv.6.1.0.3.1/workspace/BatchDevEnv/script.ant/..\export/Echo.deployed.ear
  [java] WASX7209I: Connected to process "server1" on node suthomasNode01 using SOAP connector; The
type of process is: UnManagedProcess
```

```

[java] WASX7209I: Connected to process "server1" on node suthomasNode01 using SOAP connector; The
type of process is: UnManagedProcess
[java] WASX7303I: The following options are passed to the scripting environment and are available as
arguments that are stored in the argv variable: "[Echo,
C:/workspace/batchdevenv.6.1.0.3.1/workspace/BatchDevEnv/script.ant/./export/Echo.deployed.ear]"
[java] WASX7209I: Connected to process "server1" on node suthomasNode01 using SOAP connector; The
type of process is: UnManagedProcess
[java] WASX7303I: The following options are passed to the scripting environment and are available as
arguments that are stored in the argv variable: "[Echo,
C:/workspace/batchdevenv.6.1.0.3.1/workspace/BatchDevEnv/script.ant/./export/Echo.deployed.ear]"
[java] ADMA5017I: Uninstallation of Echo started.
[java] WASX7209I: Connected to process "server1" on node suthomasNode01 using SOAP connector; The
type of process is: UnManagedProcess
[java] WASX7303I: The following options are passed to the scripting environment and are available as
arguments that are stored in the argv variable: "[Echo,
C:/workspace/batchdevenv.6.1.0.3.1/workspace/BatchDevEnv/script.ant/./export/Echo.deployed.ear]"
[java] ADMA5017I: Uninstallation of Echo started.
[java] ADMA5104I: The server index entry for WebSphere:cell=suthomasNode01Cell,node=suthomasNode01 is
updated successfully.
[java] WASX7209I: Connected to process "server1" on node suthomasNode01 using SOAP connector; The
type of process is: UnManagedProcess
...
[java] installApp.py is installing Echo using
C:/workspace/batchdevenv.6.1.0.3.1/workspace/BatchDevEnv/script.ant/./export/Echo.deployed.ear
[java] ADMA5016I: Installation of Echo started.
[java] ADMA5058I: Application and module versions are validated with versions of deployment targets.
[java] ADMA5005I: The application Echo is configured in the WebSphere Application Server repository.
[java] ADMA5053I: The library references for the installed optional package are created.
[java] ADMA5005I: The application Echo is configured in the WebSphere Application Server repository.
[java] ADMA5001I: The application binaries are saved in
C:\WebSphere\ApplicationServer\profiles\AppSrv01\wstemp\Script121ef71ed7b\workspace\cells\suthomasNode01Ce1
1\applications\Echo.ear\Echo.ear
[java] ADMA5005I: The application Echo is configured in the WebSphere Application Server repository.
[java] SECJ0400I: Successfully updated the application Echo with the appContextIDForSecurity
information.
[java] ADMA5005I: The application Echo is configured in the WebSphere Application Server repository.
[java] ADMA5011I: The cleanup of the temp directory for application Echo is complete.
[java] ADMA5013I: Application Echo installed successfully.
[java] configuration saved
[java] application Echo started
[java] installApp.py complete
[java] Command complete - rc=0
BUILD SUCCESSFUL
Total time: 51 seconds

```

If the application is installed successfully in your local WebSphere environment, you have to “submit” the job using the Eclipse workspace or using the job management console as we describe later in this section.

From your workspace, you can use the Ant script `5.runJob.Echo.xml`, as shown in Figure 8-26, for submitting a job. Right-click the script file and choose **Run as** → **Ant build**.

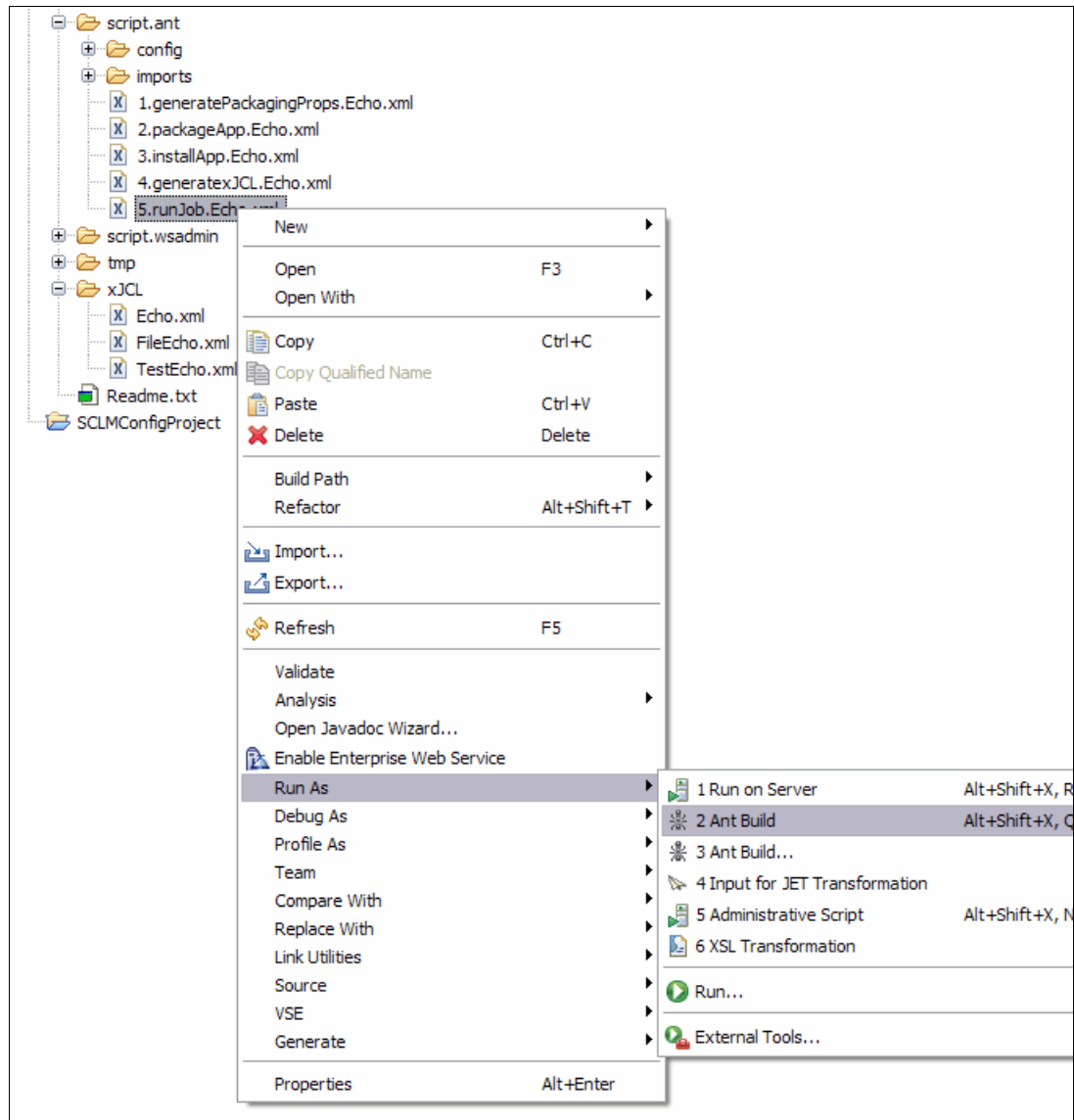


Figure 8-26 Submit Job

During the job runs, you receive the output in the console of your Eclipse workspace. The job log is also available in file under `/joblog/<jobname_jobid>.job.log.txt`. The output of this sample is shown in Example 8-12.

Example 8-12 Output - Submit Job

```
Buildfile: C:\workspace\batchdevenv.6.1.0.3.1\workspace\BatchDevEnv\script.ant\5.runJob.Echo.xml
checkFile:
runJob:
[echo] Running job Echo.xml in Compute Grid Test Server
[echo] Test Server configuration: /WebSphere/ApplicationServer/profiles/AppSrv01
[echo] Test Server host: localhost
[echo] Test Server port: 9083
[java] RunJob inputs:
[java] /WebSphere/ApplicationServer/profiles/AppSrv01
[java] C:\workspace\batchdevenv.6.1.0.3.1\workspace\BatchDevEnv\script.ant\..
[java] localhost
```

```

[java] 9083
[java] Echo.xml
[java] Substitution[0]: echo.data=C:/workspace/batchdevenv.6.1.0.3.1/workspace/BatchDevEnv/script.ant/./data
[java] Launched command /WebSphere/ApplicationServer/profiles/AppSrv01/bin/lrcmd.bat -cmd=submit
-xJCL=C:/workspace/batchdevenv.6.1.0.3.1/workspace/BatchDevEnv/script.ant/./xJCL/Echo.xml -host=localhost -port=9083
echo.data=C:/workspace/batchdevenv.6.1.0.3.1/workspace/BatchDevEnv/script.ant/./data
[java] Command complete - rc=0
[java] runJob jobid=Echo:00014
[java] Launched command /WebSphere/ApplicationServer/profiles/AppSrv01/bin/lrcmd.bat -cmd=status -jobid=Echo:00014
-host=localhost -port=9083
[java] Command complete - rc=0
[java] Echo:00014 ended normally
[java] Launched command /WebSphere/ApplicationServer/profiles/AppSrv01/bin/lrcmd.bat -cmd=saveJobLog
-jobid=Echo:00014 -fileName=C:/workspace/batchdevenv.6.1.0.3.1/workspace/BatchDevEnv/script.ant/./joblog/joblog.zip
-host=localhost -port=9083
[java] Command complete - rc=0
[java] Echo:00014 ended normally
[java] Launched command /WebSphere/ApplicationServer/profiles/AppSrv01/bin/lrcmd.bat -cmd=saveJobLog
-jobid=Echo:00014 -fileName=C:/workspace/batchdevenv.6.1.0.3.1/workspace/BatchDevEnv/script.ant/./joblog/joblog.zip
-host=localhost -port=9083
[java] Joblog saved to
C:/workspace/batchdevenv.6.1.0.3.1/workspace/BatchDevEnv/script.ant/./joblog\Echo_00014.job.log.txt
[java] CWLRB5671I: [06/17/09 14:21:48:593 GMT-06:00] Processing for job Echo:00014 started.
[java] Original XJCL
[java] 1 : <?xml version="1.0" encoding="UTF-8"?>
[java] 2 : <!--## WebSphere Batch xJCL## This file generated on 2009.06.17 at 14:18:51 GMT-06:00 by:# IBM Batch
Simulator Version 1.1 - Build 20080721#-->
[java] 3 : <job default-application-name="Echo" name="Echo" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
[java] 4 :     <jndi-name>ejb/com/ibm/ws/batch/EchoBatchController</jndi-name>
[java] 5 :     <step-scheduling-criteria>
[java] 6 :         <scheduling-mode>sequential</scheduling-mode>
[java] 7 :     </step-scheduling-criteria>
[java] 8 :     <checkpoint-algorithm name="chkpt">
[java] 9 :         <classname>com.ibm.wsspi.batch.checkpointalgorithms.RecordbasedBase</classname>
[java] 10 :         <props>
[java] 11 :             <prop name="recordcount" value="1000"></prop>
[java] 12 :         </props>
[java] 13 :     </checkpoint-algorithm>
[java] 14 :     <results-algorithms>
[java] 15 :         <results-algorithm name="jobsum">
[java] 16 :             <classname>com.ibm.wsspi.batch.resultsalgorithms.jobsum</classname>
[java] 17 :         </results-algorithm>
[java] 18 :     </results-algorithms>
[java] 19 :     <job-step name="Step1">
[java] 20 :         <jndi-name>ejb/GenericXDBatchStep</jndi-name>
[java] 21 :         <checkpoint-algorithm-ref name="chkpt"></checkpoint-algorithm-ref>
[java] 22 :         <results-ref name="jobsum"></results-ref>
[java] 23 :         <batch-data-streams>
[java] 24 :             <bds>
[java] 25 :                 <logical-name>outputStream</logical-name>
[java] 26 :                 <props>
[java] 27 :                     <prop name="FILENAME" value="{echo.data}/output.txt"></prop>
[java] 28 :                     <prop name="EnablePerformanceMeasurement" value="false"></prop>
[java] 29 :                     <prop name="EnableDetailedPerformanceMeasurement" value="false"></prop>
[java] 30 :                     <prop name="AppendJobIdToFileName" value="false"></prop>
[java] 31 :                     <prop name="debug" value="false"></prop>
[java] 32 :                     <prop name="tablename" value="alg.tivpwxdo"></prop>
[java] 33 :                     <prop name="PATTERN_IMPL_CLASS"
value="com.batch.streams.outputstreams.EchoWriter"></prop>
[java] 34 :                 </props>
[java] 35 :             </bds>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteWriter</impl-class>
[java] 36 :             </bds>
[java] 37 :         </bds>
[java] 38 :         <logical-name>inputStream</logical-name>
[java] 39 :     </props>

```

```

[.java] 40 :         <prop name="FILENAME" value="{echo.data}/input.txt"></prop>
[.java] 41 :         <prop name="EnablePerformanceMeasurement" value="false"></prop>
[.java] 42 :         <prop name="EnableDetailedPerformanceMeasurement" value="false"></prop>
[.java] 43 :         <prop name="debug" value="false"></prop>
[.java] 44 :         <prop name="PATTERN_IMPL_CLASS"
value="com.batch.streams.inputstreams.EchoReader"></prop>
[.java] 45 :         </props>
[.java] 46 :
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteReader</impl-class>
[.java] 47 :         </bds>
[.java] 48 :         </batch-data-streams>
[.java] 49 :         <props>
[.java] 50 :             <prop name="EnablePerformanceMeasurement" value="false"></prop>
[.java] 51 :             <prop name="EnableDetailedPerformanceMeasurement" value="false"></prop>
[.java] 52 :             <prop name="debug" value="false"></prop>
[.java] 53 :             <prop name="BATCHRECORDPROCESSOR" value="com.batch.steps.Echo"></prop>
[.java] 54 :         </props>
[.java] 55 :     </job-step>
[.java] 56 : </job>
[.java] Substituted XJCL
[.java] 1 : <?xml version="1.0" encoding="UTF-8"?>
[.java] 2 : <!--## WebSphere Batch XJCL## This file generated on 2009.06.17 at 14:18:51 GMT-06:00 by:# IBM Batch
Simulator Version 1.1 - Build 20080721#-->
[.java] 3 : <job default-application-name="Echo" name="Echo" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
[.java] 4 :     <jndi-name>ejb/com/ibm/ws/batch/EchoBatchController</jndi-name>
[.java] 5 :     <step-scheduling-criteria>
[.java] 6 :         <scheduling-mode>sequential</scheduling-mode>
[.java] 7 :     </step-scheduling-criteria>
[.java] 8 :     <checkpoint-algorithm name="chkpt">
[.java] 9 :         <classname>com.ibm.wsspi.batch.checkpointalgorithms.RecordbasedBase</classname>
[.java] 10 :         <props>
[.java] 11 :             <prop name="recordcount" value="1000"></prop>
[.java] 12 :         </props>
[.java] 13 :     </checkpoint-algorithm>
[.java] 14 :     <results-algorithms>
[.java] 15 :         <results-algorithm name="jobsum">
[.java] 16 :             <classname>com.ibm.wsspi.batch.resultsalgorithms.jobsum</classname>
[.java] 17 :         </results-algorithm>
[.java] 18 :     </results-algorithms>
[.java] 19 :     <job-step name="Step1">
[.java] 20 :         <jndi-name>ejb/GenericXDBatchStep</jndi-name>
[.java] 21 :         <checkpoint-algorithm-ref name="chkpt"></checkpoint-algorithm-ref>
[.java] 22 :         <results-ref name="jobsum"></results-ref>
[.java] 23 :         <batch-data-streams>
[.java] 24 :             <bds>
[.java] 25 :                 <logical-name>outputStream</logical-name>
[.java] 26 :                 <props>
[.java] 27 :                     <prop name="FILENAME"
value="C:/workspace/batchdev.6.1.0.3.1/workspace/BatchDevEnv/script.ant/./data/output.txt"></prop>
[.java] 28 :                     <prop name="EnablePerformanceMeasurement" value="false"></prop>
[.java] 29 :                     <prop name="EnableDetailedPerformanceMeasurement" value="false"></prop>
[.java] 30 :                     <prop name="AppendJobIdToFileName" value="false"></prop>
[.java] 31 :                     <prop name="debug" value="false"></prop>
[.java] 32 :                     <prop name="tablename" value="alg.tivpwx0"></prop>
[.java] 33 :                     <prop name="PATTERN_IMPL_CLASS"
value="com.batch.streams.outputstreams.EchoWriter"></prop>
[.java] 34 :                 </props>
[.java] 35 :             </bds>
[.java] 36 :         </batch-data-streams>
[.java] 37 :     </job-step>
[.java] 38 :     <logical-name>inputStream</logical-name>
[.java] 39 :     <props>
[.java] 40 :         <prop name="FILENAME"
value="C:/workspace/batchdev.6.1.0.3.1/workspace/BatchDevEnv/script.ant/./data/input.txt"></prop>
[.java] 41 :         <prop name="EnablePerformanceMeasurement" value="false"></prop>

```

```

[java] 42 :             <prop name="EnableDetailedPerformanceMeasurement" value="false"></prop>
[java] 43 :             <prop name="debug" value="false"></prop>
[java] 44 :             <prop name="PATTERN_IMPL_CLASS"
value="com.batch.streams.inputstreams.EchoReader"></prop>
[java] 45 :             </props>
[java] 46 :
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteReader</impl-class>
[java] 47 :             </bds>
[java] 48 :             </batch-data-streams>
[java] 49 :             <props>
[java] 50 :             <prop name="EnablePerformanceMeasurement" value="false"></prop>
[java] 51 :             <prop name="EnableDetailedPerformanceMeasurement" value="false"></prop>
[java] 52 :             <prop name="debug" value="false"></prop>
[java] 53 :             <prop name="BATCHRECORDPROCESSOR" value="com.batch.steps.Echo"></prop>
[java] 54 :             </props>
[java] 55 :         </job-step>
[java] 56 : </job>
[java] CWLRB5684I: [06/17/09 14:21:48:718 GMT-06:00] Job Echo:00014 is queued for execution
[java] CWLRB5586I: [06/17/09 14:21:48:718 GMT-06:00] CWLRS6006I: Job class Default, Importance 8, Service Class
null, Service Goal Type 0, Application Type j2ee, Submitter UNAUTHENTICATED.
[java] CWLRB5586I: [06/17/09 14:21:48:718 GMT-06:00] CWLRS6007I: Job Arrival Time 6/17/09 2:21 PM, Goal Max
Completion Time 0, Goal Max Queue Time 0, Breach Time 6/18/09 2:21 PM.
[java] CWLRB5586I: [06/17/09 14:21:48:718 GMT-06:00] CWLRS6021I: List of eligible endpoints to execute the job:
suthomasNode01/server1.
[java] CWLRB5586I: [06/17/09 14:21:48:718 GMT-06:00] CWLRS6011I: APC is not active. GAP will make the endpoint
selection.
[java] CWLRB5586I: [06/17/09 14:21:49:031 GMT-06:00] CWLRS6013I: GAP is dispatching job Echo:00014. Job queue time
0.313 seconds.
[java] CWLRB3090I: [06/17/09 14:21:49:343 GMT-06:00] Job Echo:00014 is dispatched to endpoint
suthomasNode01\server1: result: 0
[java] CWLRB5588I: [06/17/09 14:21:49:359 GMT-06:00] Setting up j2ee job Echo:00014 for execution in Grid Execution
Environment suthomasNode01Cell\suthomasNode01/server1: [jobClass Default] [jobName Echo] [module null] [user
UNAUTHENTICATED] [applicationName Echo] [applicationType j2ee] [transactionClass ${default_iiop_transaction_class}]
[java] CWLRB2210I: [06/17/09 14:21:49:453 GMT-06:00] Job setup manager bean is setting up job: Echo:00014
[java] CWLRB1690I: [06/17/09 14:21:49:468 GMT-06:00] No match found in job status table entry using key: [bjeename
*] [jobid Echo:00014]: Job Echo:00014 is not restarting.
[java] CWLRB1740I: [06/17/09 14:21:49:562 GMT-06:00] Job [Echo:00014] is in job setup.
[java] CWLRB1670I: [06/17/09 14:21:49:593 GMT-06:00] Creating abstract resources required by the job.
[java] CWLRB1760I: [06/17/09 14:21:49:625 GMT-06:00] Job [Echo:00014] is submitted for execution.
[java] CWLRB2230I: [06/17/09 14:21:49:687 GMT-06:00] Job setup manager bean completed job Echo:00014 setup: return
code: 0
[java] CWLRB1850I: [06/17/09 14:21:49:703 GMT-06:00] Initializing for step dispatch using scheduling mode:
sequential
[java] CWLRB1650I: [06/17/09 14:21:49:718 GMT-06:00] Created checkpoint repository table entry using key: [jobid
Echo:00014] [stepname Step1] [bdsname outputStream]
[java] CWLRB1650I: [06/17/09 14:21:49:734 GMT-06:00] Created checkpoint repository table entry using key: [jobid
Echo:00014] [stepname Step1] [bdsname inputStream]
[java] CWLRB1970I: [06/17/09 14:21:49:750 GMT-06:00] Created job step status table entry using key [jobid
Echo:00014] [stepname Step1]
[java] CWLRB1990I: [06/17/09 14:21:49:750 GMT-06:00] Found job results table entry matching on key: [jobid
Echo:00014]
[java] CWLRB2030I: [06/17/09 14:21:49:750 GMT-06:00] Initialization for sequential step dispatch is complete.
[java] CWLRB1870I: [06/17/09 14:21:49:750 GMT-06:00] Subscribing to job cancel or stop subject:
BizGridJobCancel_Echo:00014
[java] CWLRB1910I: [06/17/09 14:21:49:750 GMT-06:00] Dispatching job Echo:00014: job contains 1 step(s).
[java] CWLRB2420I: [06/17/09 14:21:49:828 GMT-06:00] Job [Echo:00014] Step [Step1] is in step setup.
[java] CWLRB5616I: [06/17/09 14:21:49:843 GMT-06:00] Setting step Step1 batch data stream outputStream properties:
EnableDetailedPerformanceMeasurement=false EnablePerformanceMeasurement=false
FILENAME=C:/workspace/batchdevnv.6.1.0.3.1/workspace/BatchDevEnv/script.ant/./data/output.txt
AppendJobIdToFileName=false debug=false PATTERN_IMPL_CLASS=com.batch.streams.outputstreams.EchoWriter
tablename=alg.tivpxd0
[java] CWLRB5618I: [06/17/09 14:21:49:843 GMT-06:00] Initializing step Step1 batch data stream outputStream
[java] CWLRB5620I: [06/17/09 14:21:49:859 GMT-06:00] Opening step Step1 batch data stream outputStream
[java] CWLRB5616I: [06/17/09 14:21:49:859 GMT-06:00] Setting step Step1 batch data stream inputStream properties:
EnableDetailedPerformanceMeasurement=false EnablePerformanceMeasurement=false

```

```

FILENAME=C:/workspace/batchdevenv.6.1.0.3.1/workspace/BatchDevEnv/script.ant/./data/input.txt debug=false
PATTERN_IMPL_CLASS=com.batch.streams.inputstreams.EchoReader
[java] CWLRB5618I: [06/17/09 14:21:49:859 GMT-06:00] Initializing step Step1 batch data stream inputStream
[java] CWLRB5620I: [06/17/09 14:21:49:875 GMT-06:00] Opening step Step1 batch data stream inputStream
[java] CWLRB5622I: [06/17/09 14:21:50:031 GMT-06:00] Loading job step bean for step Step1 using jndi name:
ejb/GenericXDBatchStep
[java] CWLRB5594I: [06/17/09 14:21:50:156 GMT-06:00] Step Step1 setup is complete: ended normally
[java] CWLRB2440I: [06/17/09 14:21:50:187 GMT-06:00] Job [Echo:00014] Step [Step1] is dispatched.
...
[java] CWLRB5628I: [06/17/09 14:21:54:093 GMT-06:00] Step Step1: chkpt checkpoint taken [iteration 76000]
[java] CWLRB5628I: [06/17/09 14:21:54:109 GMT-06:00] Step Step1: chkpt checkpoint taken [iteration 77000]
[java] CWLRB5630I: [06/17/09 14:21:54:156 GMT-06:00] Step Step1 completes normally: ended normally
[java] CWLRB2460I: [06/17/09 14:21:54:156 GMT-06:00] Job [Echo:00014] Step [Step1] is in step breakdown.
[java] CWLRB5606I: [06/17/09 14:21:54:171 GMT-06:00] Destroying job step: Step1
[java] System.out: [06/17/09 14:21:54:171 GMT-06:00] INFO->jobid: Echo:00014:GenericXDBatchStep.destroyStep()-
Total Execution Time: 4015
[java] CWLRB5608I: [06/17/09 14:21:54:171 GMT-06:00] Job step Step1 destroy completed with rc: 0
[java] CWLRB5610I: [06/17/09 14:21:54:218 GMT-06:00] Firing Step1 results algorithm
com.ibm.wsspi.batch.resultsalgorithms.jobsum: [RC 0] [jobRC 0]
[java] CWLRB5624I: [06/17/09 14:21:54:218 GMT-06:00] Stopping step Step1 chkpt checkpoint. User transaction
status: STATUS_ACTIVE
[java] CWLRB5602I: [06/17/09 14:21:54:234 GMT-06:00] Closing Step1 batch data stream: outputStream
[java] CWLRB5602I: [06/17/09 14:21:54:843 GMT-06:00] Closing Step1 batch data stream: inputStream
[java] CWLRB5604I: [06/17/09 14:21:54:859 GMT-06:00] Freeing Step1 batch data stream: outputStream
[java] CWLRB5604I: [06/17/09 14:21:54:859 GMT-06:00] Freeing Step1 batch data stream: inputStream
[java] CWLRB2600I: [06/17/09 14:21:54:859 GMT-06:00] Job [Echo:00014] Step [Step1] completed normally rc=0.
[java] CWLRB5594I: [06/17/09 14:21:54:890 GMT-06:00] Step Step1 execution is complete: ended normally
[java] CWLRB1890I: [06/17/09 14:21:54:906 GMT-06:00] Unsubscribing from job cancel or stop subject:
BizGridJobCancel_Echo:00014
[java] CWLRB3800I: [06/17/09 14:21:54:906 GMT-06:00] Job [Echo:00014] ended normally.
[java] CWLRB5596I: [06/17/09 14:21:54:953 GMT-06:00] Grid Execution Environment sequential step processing
complete: ended
[java] CWLRB2250I: [06/17/09 14:21:54:953 GMT-06:00] Job setup manager bean is breaking down job: Echo:00014
[java] CWLRB5598I: [06/17/09 14:21:54:953 GMT-06:00] Removing job abstract resources
[java] CWLRB5600I: [06/17/09 14:21:54:953 GMT-06:00] Removing job step status table entries
[java] CWLRB2270I: [06/17/09 14:21:54:984 GMT-06:00] Job setup manager bean completed job Echo:00014 breakdown
[java] CWLRB5764I: [06/17/09 14:21:54:984 GMT-06:00] Job Echo:00014 ended
[java] Command complete - rc=0
BUILD SUCCESSFUL
Total time: 34 seconds

```

The sample output data will be saved in /data/output.txt, as shown in Figure 8-27.

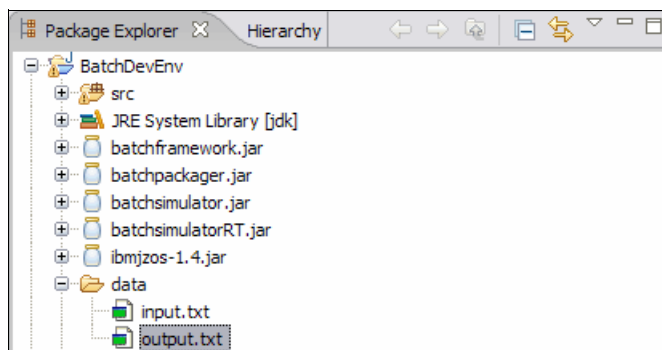


Figure 8-27 Sample output data

The other option is to use the Job Management Console (JMC) to schedule a job and submit jobs. You can also submit, manage, and view job logs using WebSphere XD Compute Grid's Job Management Console. Just point your browser at <http://localhost/jmc>. For example, for checking the job log, click **View jobs** under Job Management, open your specific job

identified by the job ID, and you can see the job output of the Echo sample, as shown in Figure 8-28.

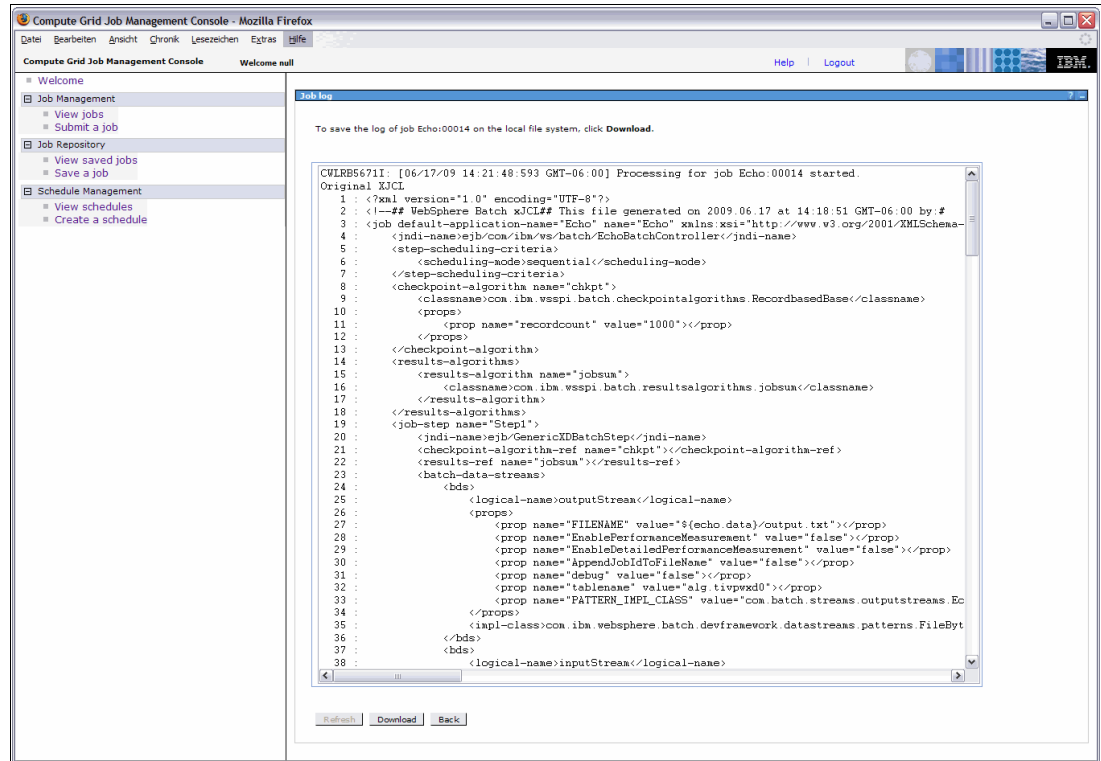


Figure 8-28 Echo Job output in the Job Management Console

Testing using the Batch Simulator on z/OS

If the Echo sample runs successfully on your local WebSphere UTE, you are ready to test your batch application on z/OS. The first option is also to use the Batch Simulator, which is very easy to use. You do not need access to the JMC to submit your test jobs and also your application does not have to be installed in WebSphere on z/OS which is very useful for testing applications.

1. The first step is to make some small changes in your properties file (remember that the Batch Simulator uses simple properties files instead of xJCLs), because your input/output file will be under UNIX System Services. A sample zEcho properties file is also in the workspace in the folder `props.simulator`. You only have to change the directory for your input and your output file in the zEcho properties file, as shown in Example 8-13, where you use your user directory.

Example 8-13 Sample zEcho properties file

```

bds-prop.inputStream.FILENAME=/u/sthomas/data/input.txt
...
bds-prop.outputStream.FILENAME=/u/sthomas/data/output.txt
  
```

2. The next step is to transfer the necessary files to the host. You can use FTP, an FTP client, an Ant script from your workspace, or tools such as Rational Developer for System z to transfer the following files to the UNIX System Services directory:
 - zEcho.props
 - Libraries
 - batchframework.jar
 - batchpackager.jar
 - batchsimulator.jar
 - batchsimulatorRT.jar
 - ibmjzos-1.4.jar
 - input.txt
 - Echo.jar (also available in the workspace in the lib directory)
3. Export your Echo sample to Echo.jar. Select domain objects, steps, input streams, and output streams, right-click, and select Export in the pop-up window, as shown in Figure 8-29.

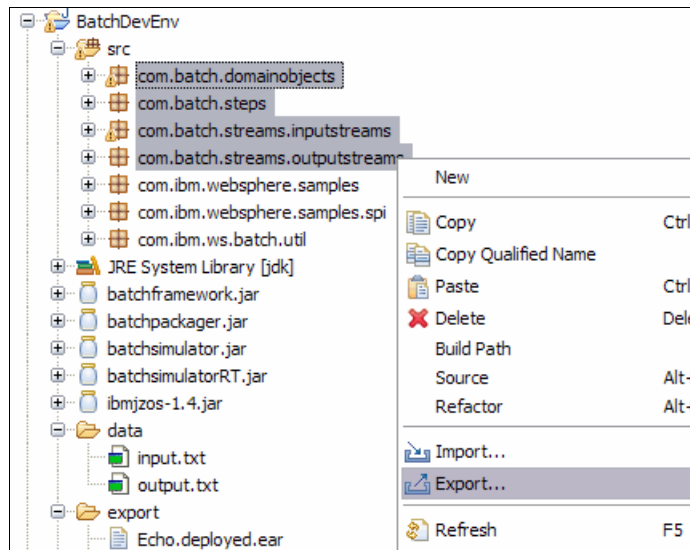


Figure 8-29 Export Echo.jar

4. In the following window, specify the output JAR file, and select other options, as shown in Figure 8-30.

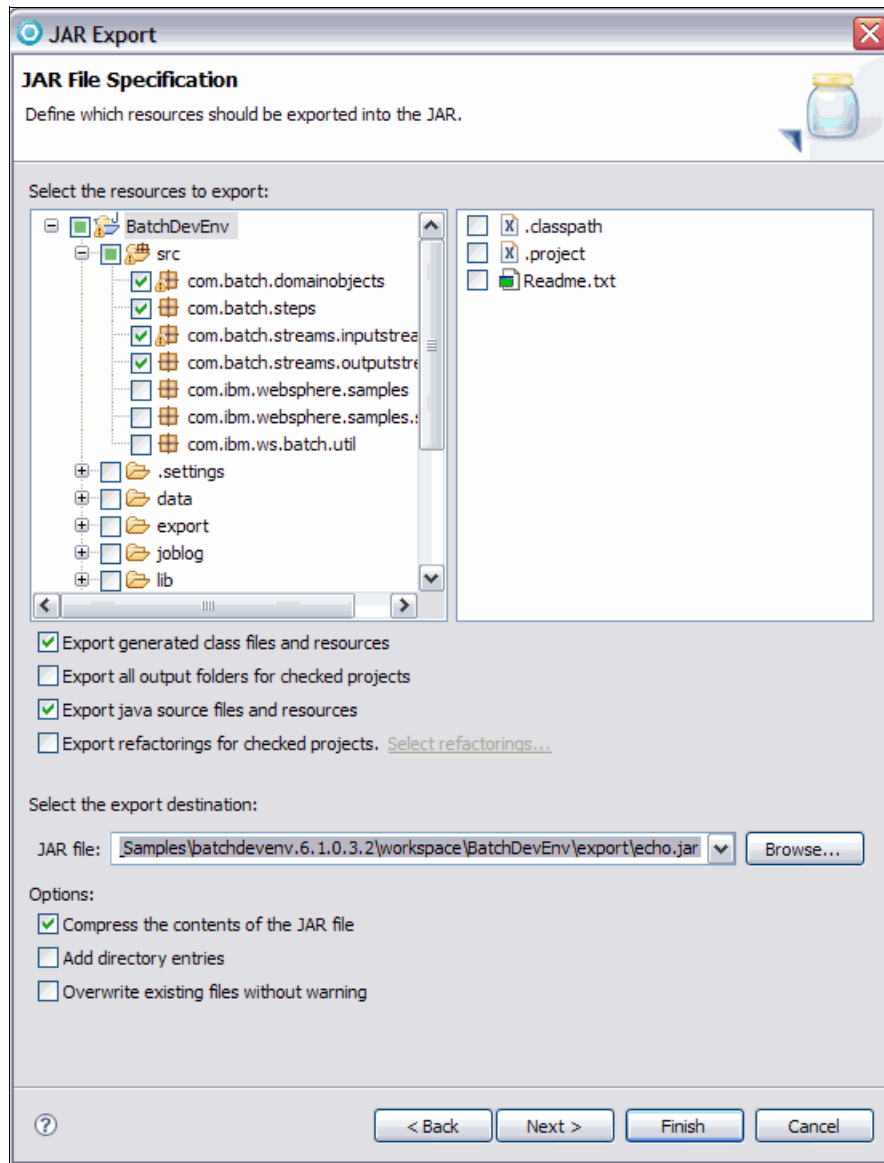


Figure 8-30 Export Echo.jar specification

- After the export and upload, the UNIX System Services file system on z/OS should look as shown in Figure 8-31.

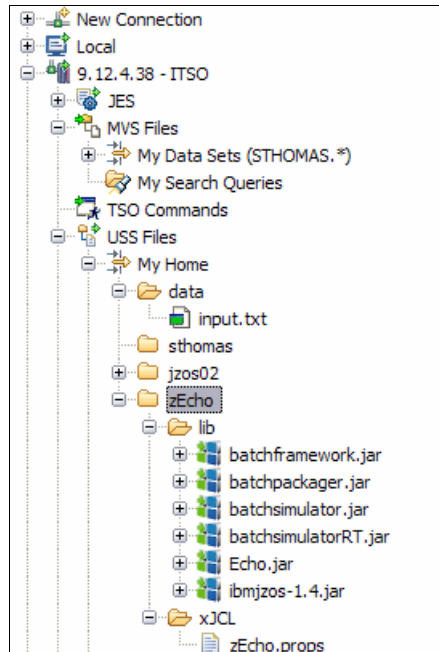


Figure 8-31 Using the Batch Simulator under UNIX System Services - Files, as shown through the Rational Developer for System z remote system explorer view

- Create a batch.sh file under UNIX System Services, and add the command shown in Example 8-14.

Example 8-14 Sample batch.sh file to run Echo under z/OS with Batch Simulator

```
java -classpath
lib/Echo.jar:lib/batchframework.jar:lib/batchsimulator.jar:lib/batchsimulatorRT
.jar:lib/batchruntime.jar:lib/ibmjzos1.4.jar
com.ibm.websphere.batch.BatchSimulator xJCL/zEcho.props
```

- Change the permission of the script to 755 and run the shell script `batch.sh` from your UNIX System Services console, as shown in Figure 8-32. The command window of Rational Developer for System z could also be used to invoke the `chmod` command and run the shell script.

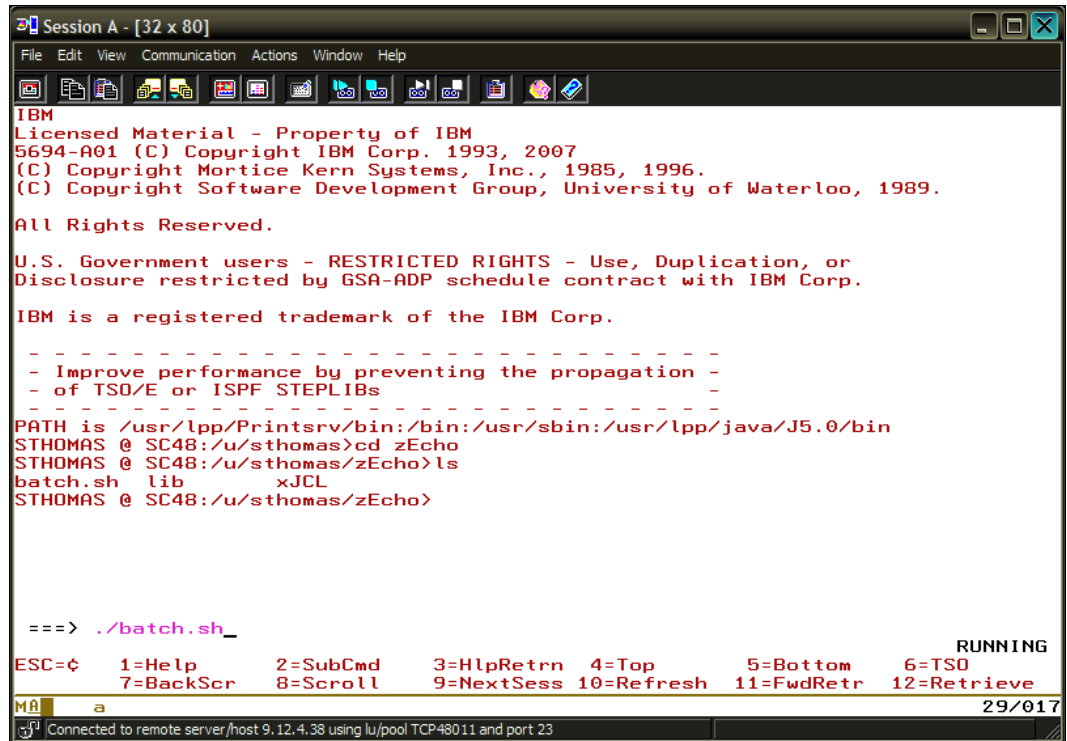


Figure 8-32 Run `batch.sh` to use the Batch Simulator under UNIX System Services

The output should look as shown in Figure 8-33 and Figure 8-34.

```

U.S. Government users - RESTRICTED RIGHTS - Use, Duplication, or
Disclosure restricted by GSA-ADP schedule contract with IBM Corp.

IBM is a registered trademark of the IBM Corp.

- - - - -
- Improve performance by preventing the propagation -
- of TSO/E or ISPF STEPLIBs -
- - - - -

PATH is /usr/lpp/Printsrv/bin:/bin:/usr/sbin:/usr/lpp/java/J5.0/bin
STHOMAS @ SC48:/u/sthomas>cd zEcho
STHOMAS @ SC48:/u/sthomas/zEcho>ls
batch.sh lib xJCL
STHOMAS @ SC48:/u/sthomas/zEcho>./batch.sh
STHOMAS @ SC48:/u/sthomas/zEcho>./batch.sh
IBM Batch Simulator Version 1.1 - Build 20080721
BatchSimulator: start job Echo
BatchSimulator: outputStream checkpoint data: 0
BatchSimulator: inputStream checkpoint data: 0
BatchSimulator: outputStream checkpoint data: 80000
BatchSimulator: inputStream checkpoint data: 1000
BatchSimulator: outputStream checkpoint data: 160000
BatchSimulator: inputStream checkpoint data: 2000
BatchSimulator: outputStream checkpoint data: 240000
BatchSimulator: inputStream checkpoint data: 3000
BatchSimulator: outputStream checkpoint data: 320000
BatchSimulator: inputStream checkpoint data: 4000
==> _
RUNNING
ESC=␣ 1=Help 2=SubCmd 3=HlpRetrn 4=Top 5=Bottom 6=TSO
7=BackScr 8=Scroll 9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve

```

Figure 8-33 Run Echo sample with Batch Simulator under UNIX System Services - Part 1

```

BatchSimulator: outputStream checkpoint data: 5280000
BatchSimulator: inputStream checkpoint data: 66000
BatchSimulator: outputStream checkpoint data: 5360000
BatchSimulator: inputStream checkpoint data: 67000
BatchSimulator: outputStream checkpoint data: 5440000
BatchSimulator: inputStream checkpoint data: 68000
BatchSimulator: outputStream checkpoint data: 5520000
BatchSimulator: inputStream checkpoint data: 69000
BatchSimulator: outputStream checkpoint data: 5600000
BatchSimulator: inputStream checkpoint data: 70000
BatchSimulator: outputStream checkpoint data: 5680000
BatchSimulator: inputStream checkpoint data: 71000
BatchSimulator: outputStream checkpoint data: 5760000
BatchSimulator: inputStream checkpoint data: 72000
BatchSimulator: outputStream checkpoint data: 5840000
BatchSimulator: inputStream checkpoint data: 73000
BatchSimulator: outputStream checkpoint data: 5920000
BatchSimulator: inputStream checkpoint data: 74000
BatchSimulator: outputStream checkpoint data: 6000000
BatchSimulator: inputStream checkpoint data: 75000
BatchSimulator: outputStream checkpoint data: 6080000
BatchSimulator: inputStream checkpoint data: 76000
BatchSimulator: outputStream checkpoint data: 6160000
BatchSimulator: inputStream checkpoint data: 77000

INFO->jobid: Echo:GenericXDBatchStep.destroyStep() - Total Execution Time: 146462
BatchSimulator: end job Echo - RC=0
STHOMAS @ SC48:/u/sthomas/zEcho>STHOMAS @ SC48:/u/sthomas/zEcho>
==> _
RUNNING
ESC=␣ 1=Help 2=SubCmd 3=HlpRetrn 4=Top 5=Bottom 6=TSO
7=BackScr 8=Scroll 9=NextSess 10=Refresh 11=FwdRetr 12=Retrieve

```

Figure 8-34 Run Echo sample with Batch Simulator under UNIX System Services - Part 2

8.7.4 Running Echo batch application on WebSphere XD Compute Grid z/OS

The last step is to install the Echo sample in the WebSphere XD Compute Grid on z/OS environment and submit the job. Figure 8-35 shows a simple WebSphere XD Compute Grid z/OS environment with two application servers, one for the Job Scheduler (JS) and one as Grid Execution Endpoint (GEE). In a production environment it is recommended to use a complete clustered environment, but for testing this environment should be enough.

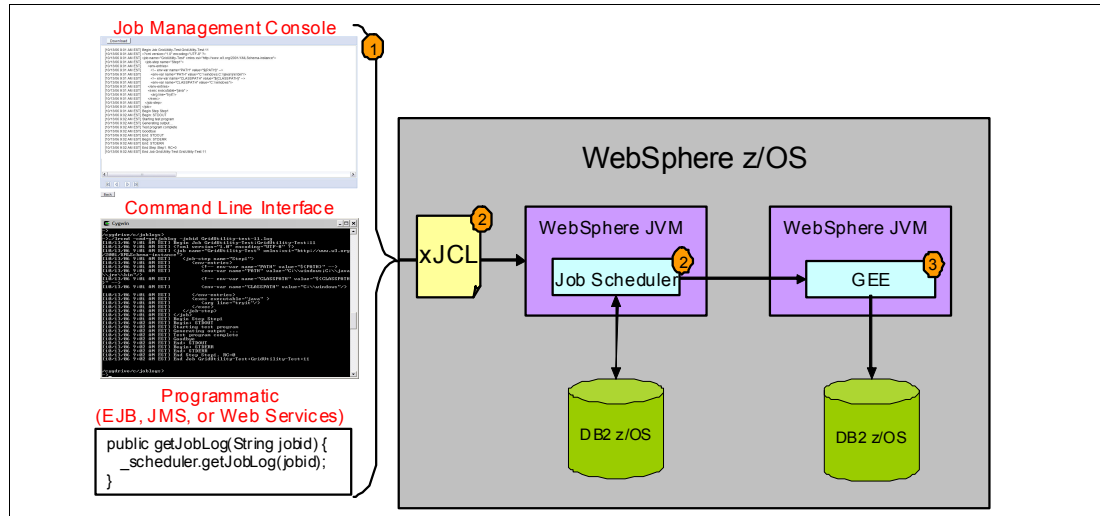


Figure 8-35 Sample WebSphere XD Compute Grid z/OS environment

The first step is to configure the Compute Grid environment.

Note: This is not a complete configuration walk through, only some important facts are mentioned here. More details for how to configure a WebSphere XD Compute Grid environment are provided in the Information Center or the WebSphere XD Compute Grid Wiki.

As mentioned previously, we are using two application server, allrsa01 which host the Job Scheduler and alqcea01 as Grid Execution Endpoint, as shown in Figure 8-36.

Application servers

Application servers

Use this page to view a list of the application servers in your environment and the status of each of these servers. You can also use this page to change the status of a specific application server.

Preferences

New Delete Templates... Start Stop ImmediateStop Terminate

Select	Name	Node	Version	Cluster Name	Status
<input type="checkbox"/>	alqcea01	alnodea	ND 6.1.0.23 WS FEP 6.1.0.23 WXDCG 6.1.0.5		+
<input type="checkbox"/>	allrsa01	alnodea	ND 6.1.0.23 WS FEP 6.1.0.23 WXDCG 6.1.0.5		+

Total 2

Figure 8-36 WebSphere Application Server for z/OS Job Scheduler and Grid Execution Endpoint

We use DB2 z/OS instead of the default Derby database tables (already configured after the installation). To use DB2 for z/OS, define a DB2 Universal JDBC Driver Provider and two data sources for the job scheduling and for the checkpoint database. See Figure 8-37.

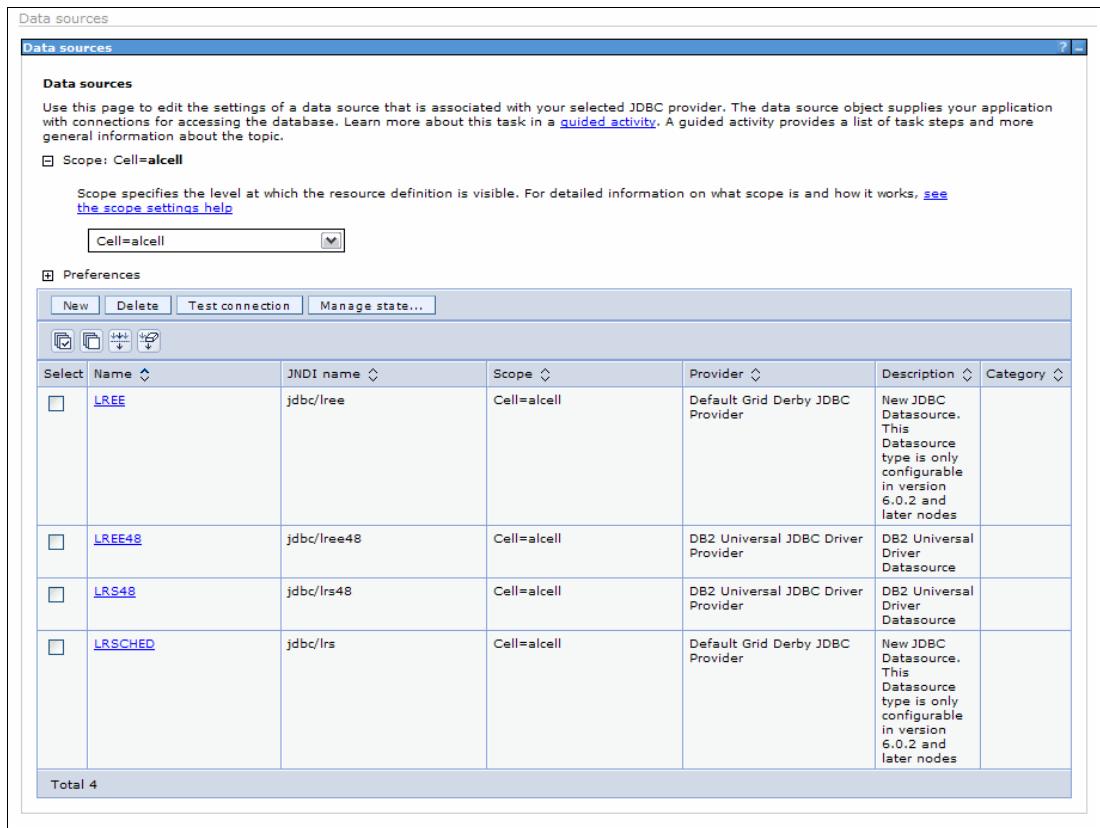


Figure 8-37 Data source definition for job scheduling and checkpoint database

The last step is to configure the Job Scheduler under **System administration** → **Job scheduler**. Set the following properties, as shown in Figure 8-38:

- ▶ Application server on which the Job Scheduler is hosted
- ▶ Database schema name
- ▶ Data source JNDI name
- ▶ Endpoint job log location

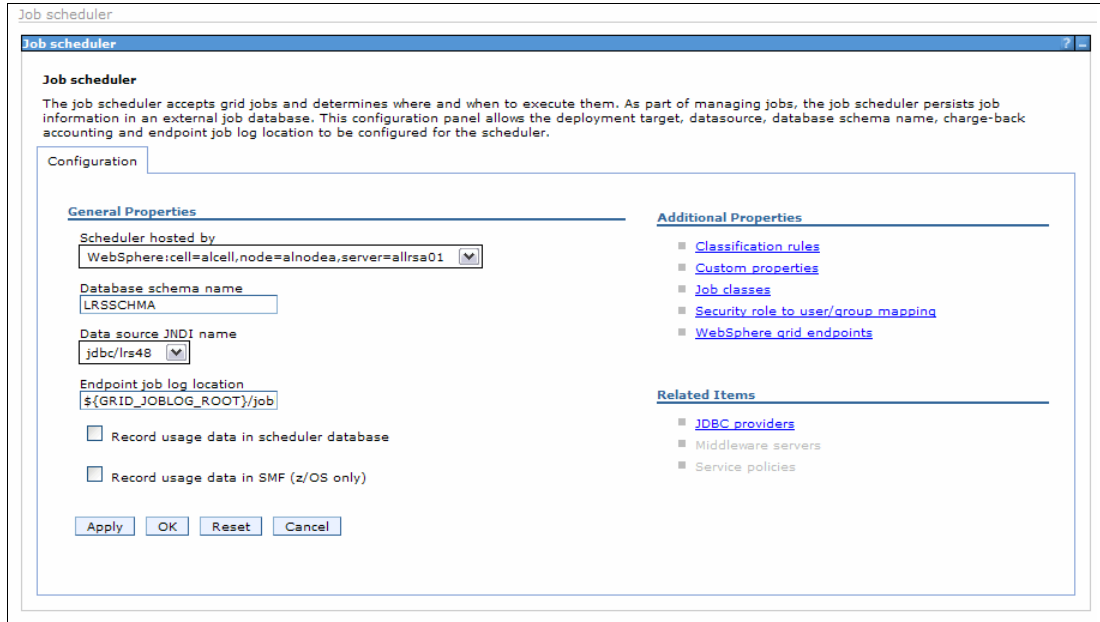


Figure 8-38 Job Scheduler configuration

The Grid Execution Endpoints are defined under **Job scheduler** → **WebSphere grid endpoint**, as shown in Figure 8-39.



Figure 8-39 Grid endpoint definition

The next step, after the WebSphere XD Compute Grid z/OS is running, is to install the Echo sample application. You can use a `wsadmin` script or the Admin console. When using the Admin console, you can install your Echo sample under **Applications** → **Install New Application**. Browse to the `Echo.deployed.ear` file in the Eclipse workspace, and click **Next**, as shown in Figure 8-40.

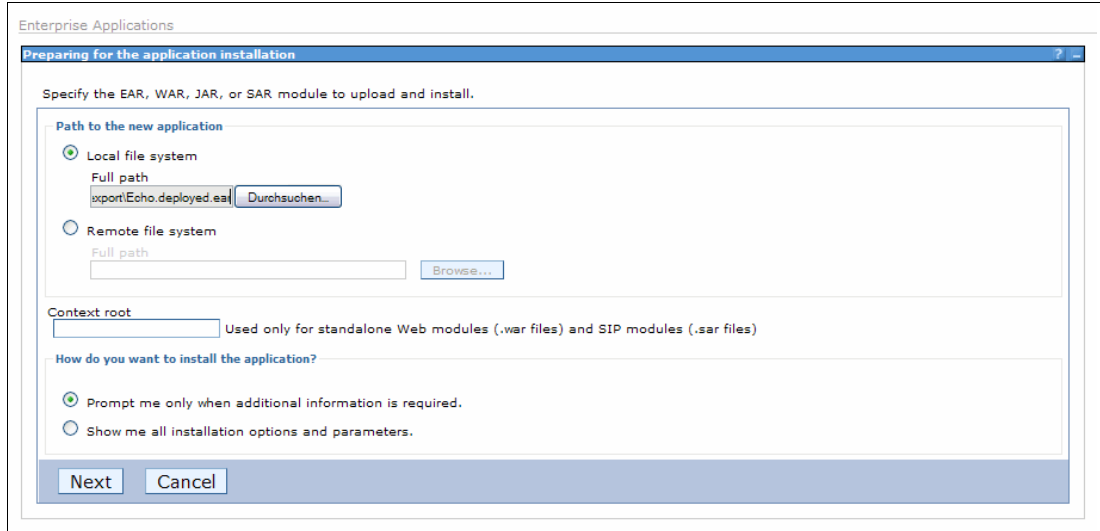


Figure 8-40 Install Echo sample on WebSphere on z/OS

Click **Next** to accept the defaults on the next window, because we do not need any installation options. These might be necessary for other applications. To map the modules to the GEE server, choose the server for the grid execution, select the module `EchoEJBs`, click **Apply** and then click **Next**, as shown in Figure 8-41.

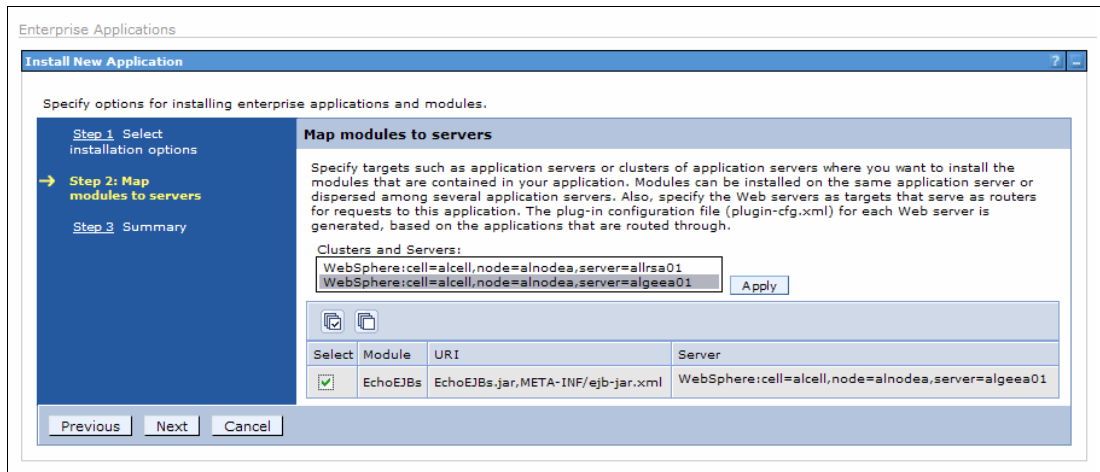


Figure 8-41 Map Modules to Grid Execution Endpoint server

Click **Finish**. Your Echo application is now installed on the grid execution server. Click **Save** (directly to the master configuration), as shown in Figure 8-42.

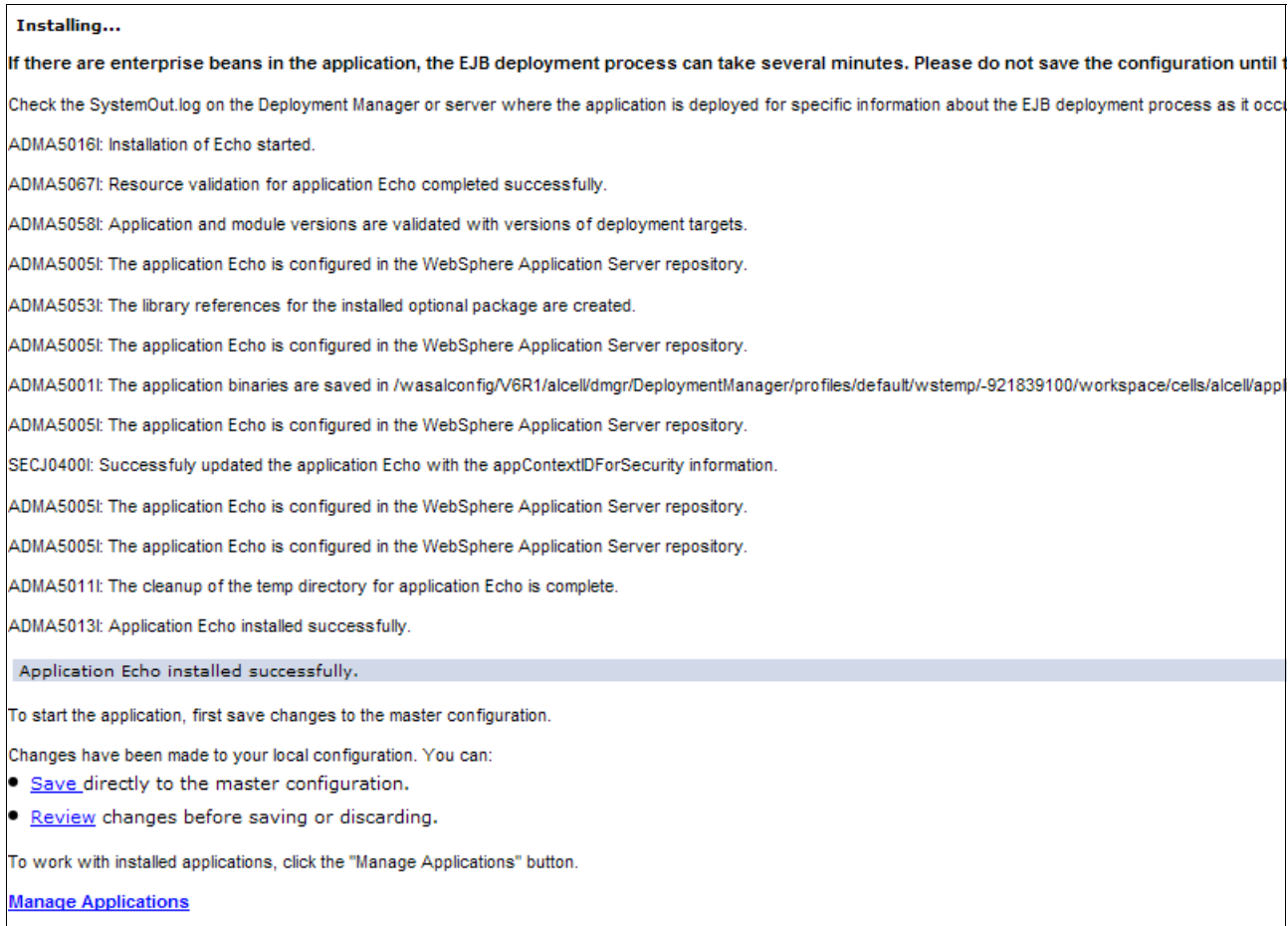


Figure 8-42 Save installed Echo application in the master configuration

The last step, is to start the Echo application. Select **Echo**, and click **Start**, as shown in Figure 8-43.

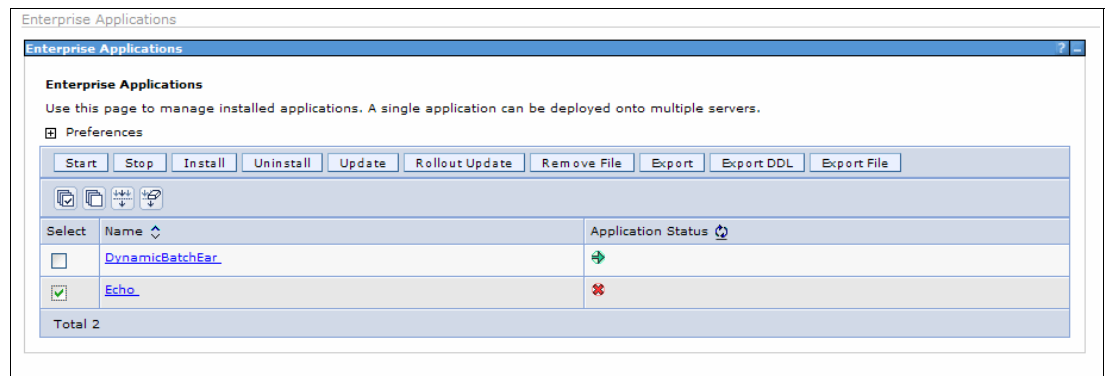


Figure 8-43 Start Echo application on WebSphere on z/OS

Your Echo sample application should now start successfully, resulting in a green arrow, as shown in Figure 8-44.

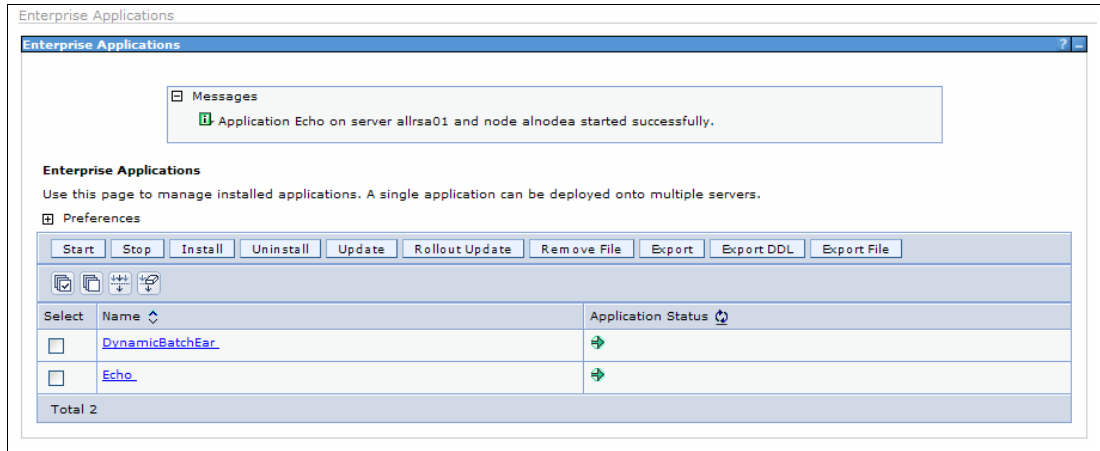


Figure 8-44 Run Echo application on WebSphere Application Server for z/OS

You can find different options to trigger a batch application in 14.2, “Using WebSphere XD Compute Grid trigger mechanisms” on page 234.

8.7.5 Debugging your application in the Unit Test Server

Use the Java remote debugger for source debugging of WebSphere XD Compute Grid batch applications. To prepare the environment:

1. First, enable debugging in your WebSphere XD Compute Grid Test Server by selecting **Application Server** → **server1** → **Process Definition** → **Java Virtual Machine**. See Figure 8-45 for the values to use.

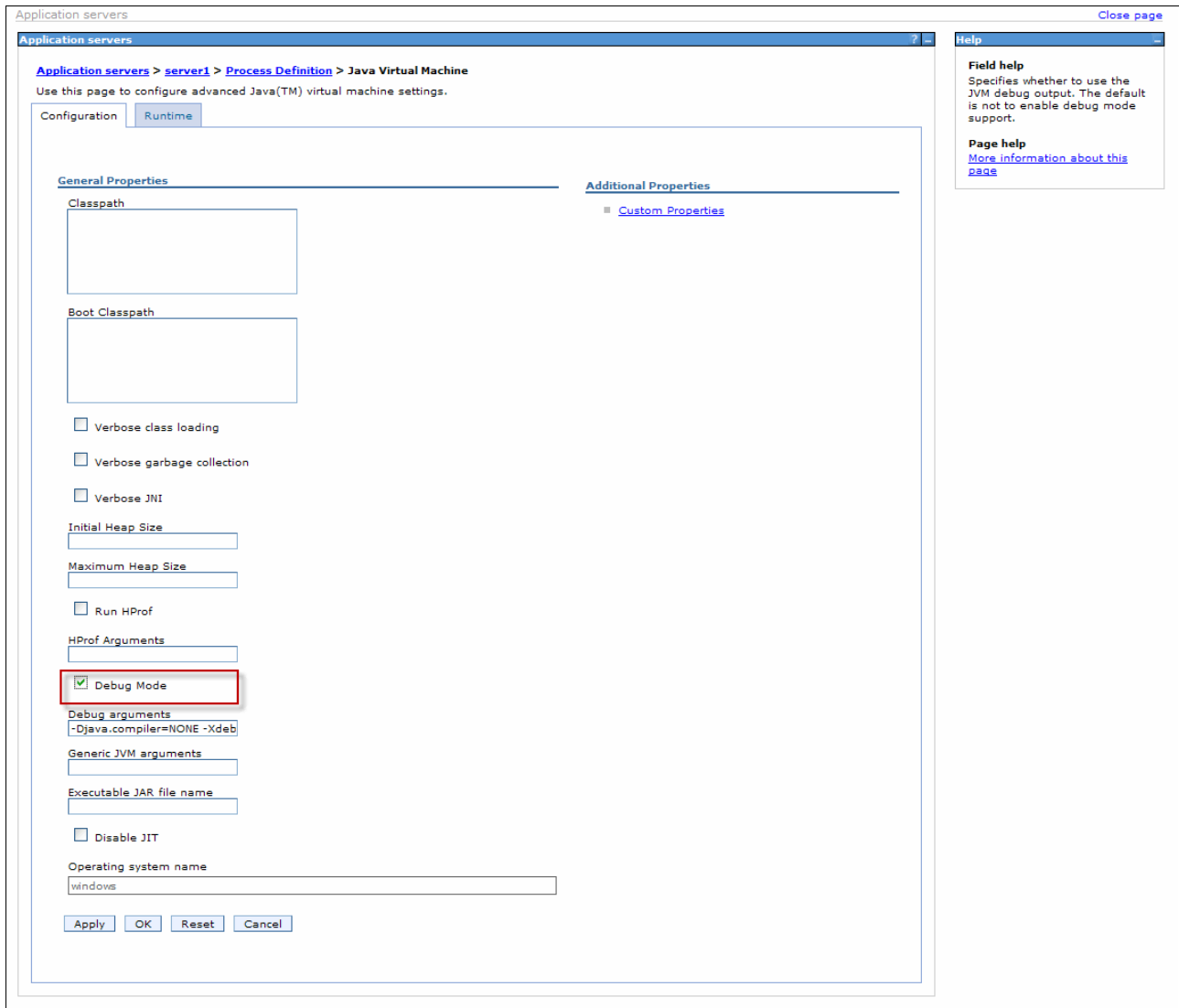


Figure 8-45 WebSphere Admin console - setting Debug Mode

Take note of the debugger port number. The default of 7777 should be fine. In addition, remember that you must restart the Test Server to activate the debugger port.

- Next, create a Debug Configuration in Eclipse. The port number in the debug configuration must match the one you used in your Test Server configuration, as shown in Figure 8-46. Click **Debug** to activate the debug session.

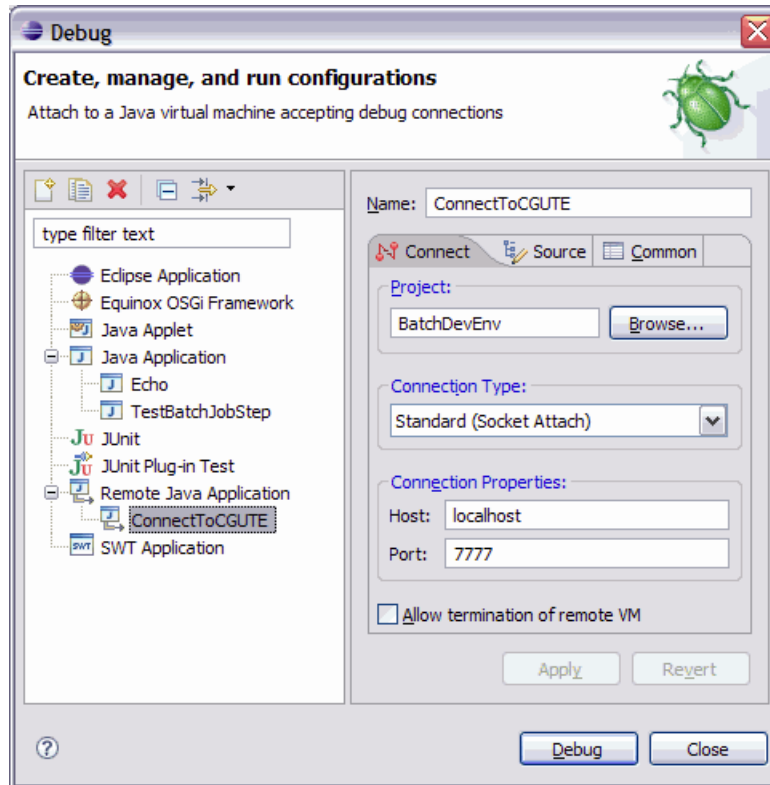


Figure 8-46 Eclipse Debug Configuration

3. Finally, set a break point in your batch application and run a job that executes it, as shown in Figure 8-47, by clicking in the left margin of the source code pane next to the statement where you want the debug sessions to pause.

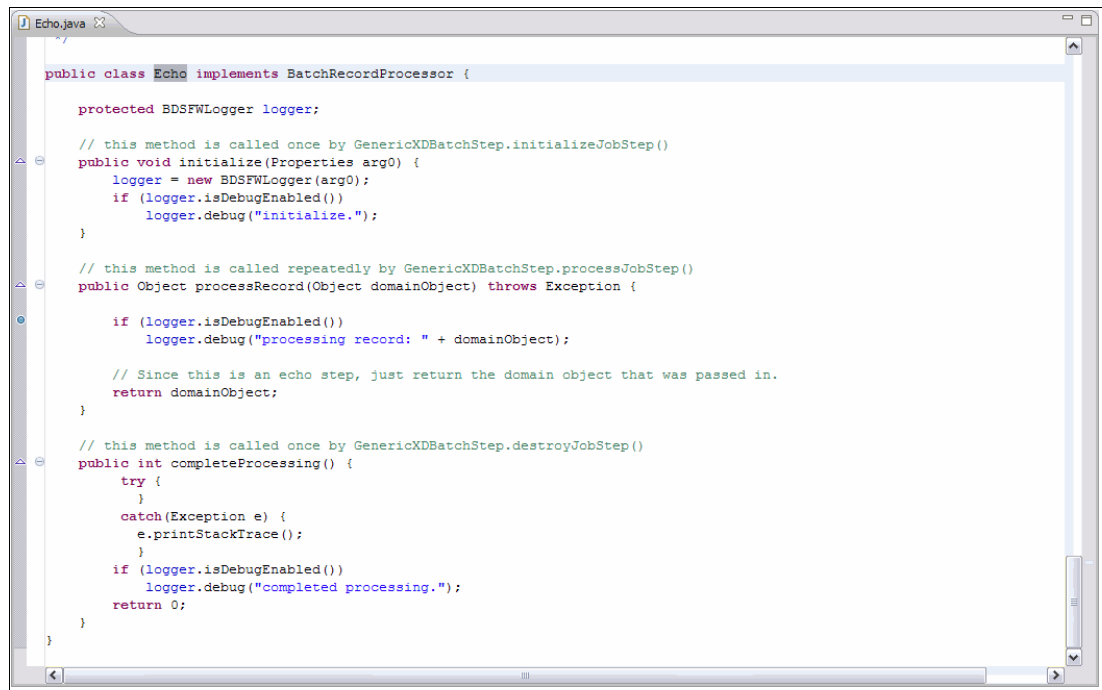


Figure 8-47 Eclipse - Set breakpoint

4. Now, run the job using the runJobEcho.xml Ant script, as shown in Figure 8-48, and this time, the debug window opens.

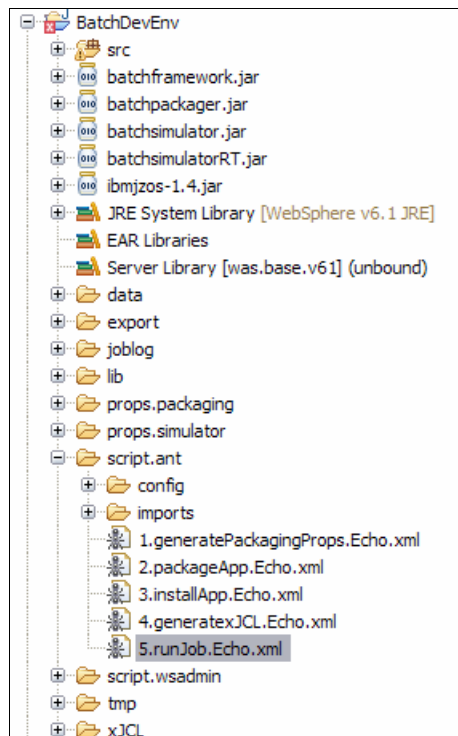


Figure 8-48 Run Echo with Ant script

8.7.6 Reusing the Echo application for huge data processing using BDS

With the simple Echo application you can start to create more data processing batch flows only by using xJCL and the patterns of the BDS framework. To run the Echo example as shown in Figure 8-49, the xJCL includes the following steps each having an input and an output stream mapping:

- ▶ Step 1: File to ZFile
- ▶ Step 2: ZFile to database
- ▶ Step 3: Database to database
- ▶ Step 4: Database to ZFile
- ▶ Step 5: ZFile to File

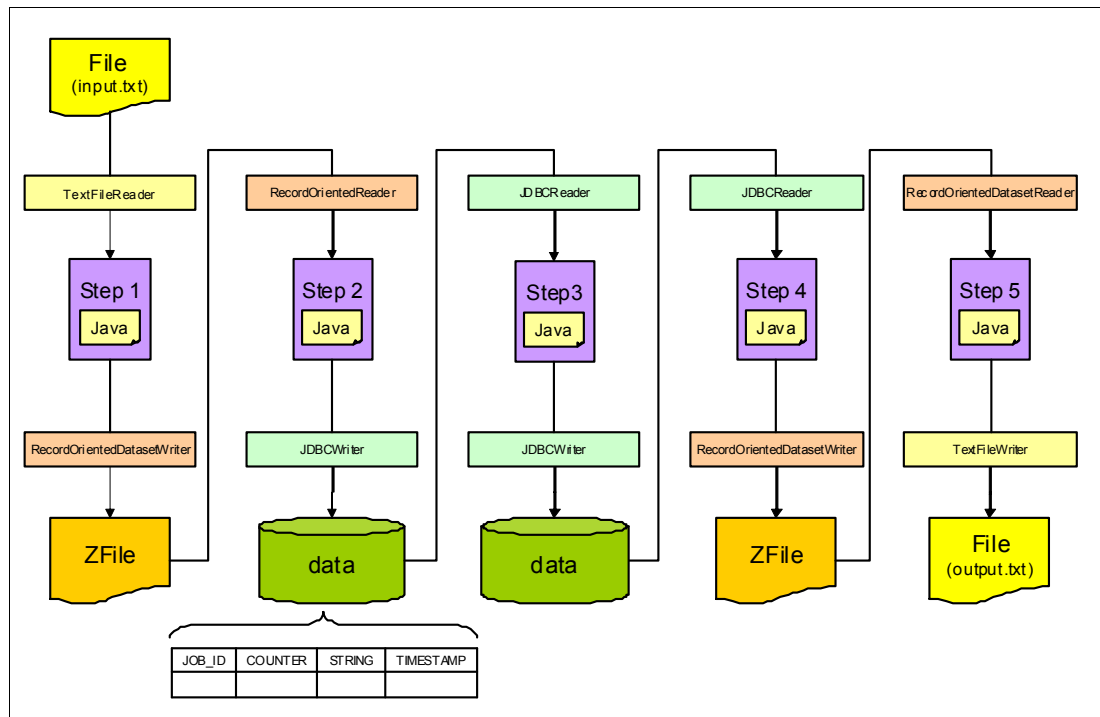


Figure 8-49 Sample batch application for huge data processing

Note: The Echo sample uses a batch step and input/output patterns from the BDS framework.

To create such data flow processing using the Echo application, which implements all BDS patterns, you have to change the properties for the input/output stream in the xJCL:

- ▶ For data in files
 - `FileByteReader / FileByteWriter` (Example 8-15)
 - `TextFileReader / TextFileWriter` (Example 8-16)

Example 8-15 xJCL properties for reading from a text file

```
<bds>
<logical-name>inputStream</logical-name>
<props>
<prop name="PATTERN_IMPL_CLASS" value="com.batch.streams.inputstreams.EchoReader"></prop>
<prop name="FILENAME" value="/u/sthomas/data/input.txt"></prop>
<prop name="debug" value="false"></prop>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.TextFileReader</impl-class>
</bds>
```

Example 8-16 xJCL properties for writing to a text file

```
<bds>
<logical-name>outputStream</logical-name>
<props>
<prop name="PATTERN_IMPL_CLASS" value="com.batch.streams.inputstreams.EchoWriter"></prop>
<prop name="FILENAME" value="/u/sthomas/data/output.txt"></prop>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.TextFileWriter</impl-
class>
</bds>
```

- ▶ For data in z/OS data sets
 - `ZFileStreamOrientedTextReader / ZFileStreamOrientedTextWriter` (Example 8-17)
 - `ZFileStreamOrientedByteReader / ZFileStreamOrientedByteWriter` (Example 8-18)
 - `ZFileRecordOrientedDataReader / ZFileRecordOrientedDataWriter`

Example 8-17 xJCL properties for reading byte data from a z/OS data set

```
<bds>
<logical-name>inputStream</logical-name>
<props>
  <prop name="PATTERN_IMPL_CLASS" value="com.batch.streams.inputstreams.EchoReader"/>
  <prop name="DSNAME" value="STHOMAS.BATCH.RECORD.INPUT"/>
  <prop name="ds_parameters" value="rt"/>
  <prop name="file.encoding" value="CP1047"/>
  <prop name="debug" value="true"/>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.ZFileStreamOrientedByteReader</impl-
class>
</bds>
```

Example 8-18 xJCL properties for writing byte data into a z/OS data set

```
<bsd>
<logical-name>outputStream</logical-name>
<props>
<prop name="PATTERN_IMPL_CLASS" value="com.batch.streams.outputstreams.EchoWriter"/>
<prop name="DSNAME" value="STHOMAS.BATCH.RECORD.OUTPUT"/>
<prop name="ds_parameters" value="wt"/>
<prop name="file.encoding" value="CP1047"/>
<prop name="debug" value="${debug}"/>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.ZFileStreamOrientedByteWriter</impl-class>
</bsd>
```

- ▶ For data in databases
 - LocalJDBCReader/LocalJDBCWriter (Example 8-19)
 - JDBCReader/JDBCWriter (Example 8-20)
 - CursorHoldableJDBCReader

Example 8-19 xJCL properties for reading data from a database using a JDBC connection

```
<bsd>
<logical-name>inputStream</logical-name>
<props>
<prop name="PATTERN_IMPL_CLASS" value="com.batch.streams.inputstreams.EchoReader"/>
<prop name="jdbc_url" value="jdbc:derby:C:\mysample\CREDITREPORT"/>
<prop name="jdbc_driver" value="org.apache.derby.jdbc.EmbeddedDriver"/>
<prop name="user_id" value="myuserid"/>
<prop name="pswd" value="myps wd"/>
<prop name="debug" value="true"/>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.LocalJDBCReader</impl-class>
</bsd>
```

Example 8-20 xJCL properties for writing data to a database using a JDBC connection

```
<bsd>
<logical-name>outputStream</logical-name>
<props>
<prop name="PATTERN_IMPL_CLASS" value="com.batch.streams.outputstreams.EchoWriter"/>
<prop name="jdbc_url" value="jdbc:derby:C:\mysample\CREDITREPORT"/>
<prop name="jdbc_driver" value="org.apache.derby.jdbc.EmbeddedDriver"/>
<prop name="user_id" value="myuserid"/>
<prop name="pswd" value="myps wd"/>
<prop name="debug" value="true"/>
</props>
<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.LocalJDBCWriter</impl-class>
</bsd>
```

8.7.7 Conclusion

WebSphere Extended Deployment Compute Grid provides a simple abstraction of a batch job step and its inputs and outputs. The programming model is concise and straightforward to use. The built-in checkpoint/rollback mechanism makes it easy to build robust, restartable Java batch applications. The Batch Simulator utility offers an alternative test environment that

runs inside your Eclipse (or Rational Application Developer or Rational Developer for System z) development environment. Its xJCL generator can help jump start you to the next phase of testing in the Compute Grid Unit Test Server.

8.8 Summary

Batch applications within a WebSphere XD Compute Grid environment are deployed like any regular Java EE application. After it is deployed, WebSphere Compute Grid detects that it is a Java EE-based batch application. When the application is deployed, an administrator can define service policies for the application, in preparation for submitting a job. The service policies are different for such batch applications, the only metrics supported for batch applications are maximum desired queue time and discretionary.



Implement new functionality using PHP on z/OS

In this chapter we provide an overview of using PHP on z/OS. This chapter includes the following topics:

- ▶ Introduction to PHP for z/OS
- ▶ PHP Interoperability with other languages
- ▶ Development tools
- ▶ Sample application for using PHP in stand-alone batch

9.1 Introduction to PHP for z/OS

In addition to using stand-alone as well as container-managed Java for new functionality in z/OS batch, PHP is also an interesting alternative to consider.

PHP is a scripting language that is very often used for dynamic Web sites. It offers a rich set of functionality. Because it is widely used by a large community of programmers around the world, many skills are available on the market today. For z/OS, there is also a version available as part of the IBM Ported Tools for z/OS, which is available at:

<http://www.ibm.com/servers/eserver/zseries/zos/unix/ported/php/index.html>

For this book we used version 5.1.2 of PHP for z/OS.

Figure 9-1 shows how PHP for z/OS works. The PHP interpreter as base for PHP scripts is using UNIX System Services under z/OS. In addition to the base PHP libraries that are delivered together with the PHP interpreter, PHP for z/OS also offers a PDO extension to support access to DB2 on z/OS. This extension allows to access DB2 on z/OS data using the Open Database Connectivity (ODBC) interface.

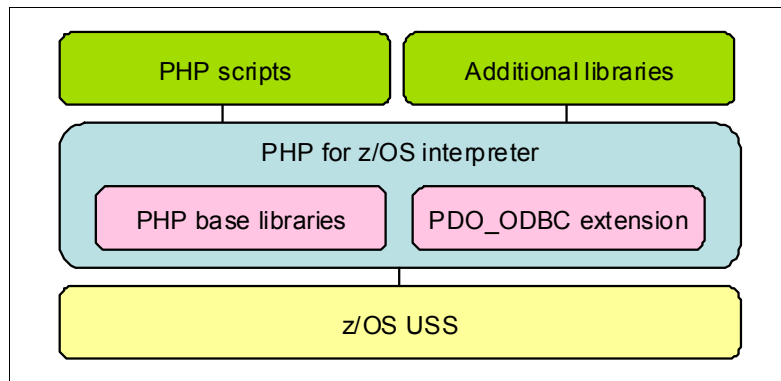


Figure 9-1 PHP for z/OS overview

Note: Custom PHP extensions are not supported on z/OS.

Based on the PHP interpreter and the PHP base libraries, we can run our PHP scripts. A script can be called using the **php** command in a shell or from the command line. For example, to call a HelloWorld program in PHP, enter the following command on the UNIX System Services shell:

```
php HelloWorld.php
```

Similar to Java, there are also lots of additional PHP libraries available that offer extra functionality on top of the base PHP libraries. These libraries are available on Open Source base as well as from software vendors.

To exploit PHP in batch, the interpreter needs to be launched using a z/OS batch launcher that opens a UNIX System Services shell and calls the php interpreter as shown in Figure 9-2. The batch launcher itself is called by a JCL and returns the shell output directly to the JES sysout. Furthermore, the job can be managed by a workload scheduler like Tivoli Workload Scheduler.

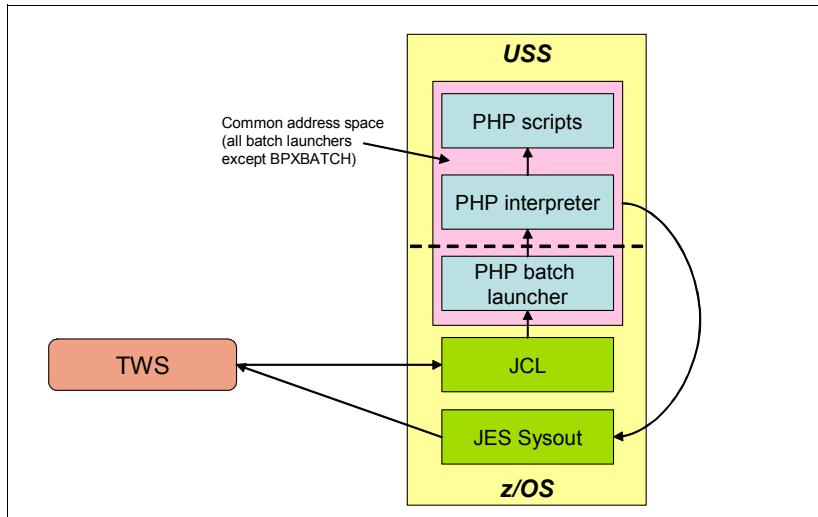


Figure 9-2 PHP batch launcher overview

Available batch launcher options for PHP are:

- ▶ BPXBATCH as a part of z/OS
- ▶ BPXBATSL as a part of z/OS
- ▶ AOPBATCH as part of the Infoprint Server
- ▶ Co:Z Batch from Dovetailed Technologies, LLC

Because each batch launcher has different characteristics, we provide a comparison in Table 9-1.

Table 9-1 z/OS UNIX System Services batch launcher comparison

Feature	BPXBATCH	BPXBATSL	AOPBATCH	COZBATCH
Same address space	No	Yes	Yes	Yes
STDOUT/STD to data set/sysout	Yes (1.7+)	Yes (1.7+)	Yes	Yes
STDIN from data set/sysin	No	No	Yes	Yes
Run a default login shell	Yes	No (unless root)	No	Yes
Run a UNIX command without a shell	Yes	Yes	Yes	Yes
Set ENV vars from JCL vars	Limited ^a	Limited ^a	Limited ^a	Yes
PARM= args support quoting	No	No	Yes	Yes
Override(set) step return code	No	No	No	Yes

a. One or more ENVAR() Language Environment runtime options can be specified on the PARM to set environment variables using JCL variables. The size of the entire PARM field is limited to 100 characters.

9.2 PHP Interoperability with other languages

In contrast to Java, there is no supported method for executing native calls from PHP to other languages and the other way around. However, the use of functionality provided by the z/OS batch environment to establish a mixed environment of PHP and other languages in stand-alone batch still applies. That means we can integrate PHP into existing batch environments by using job nets of different programming languages that exchange data using files or databases (see 5.5, “Java Interoperability with COBOL and PL/I” on page 55).

9.3 Development tools

To develop PHP batch applications, we can use ISPF or Eclipse based tools. Because features such as syntax highlighting or code completion for PHP are difficult in ISPF, we created an Eclipse project for using PHP batch on z/OS.

Figure 9-3 shows how Eclipse can be used for PHP development on z/OS. We used the PHP Development Tools (PDT) for Eclipse to develop PHP code on a local workstation, which allows features such syntax highlighting and automatic code completion for PHP code.

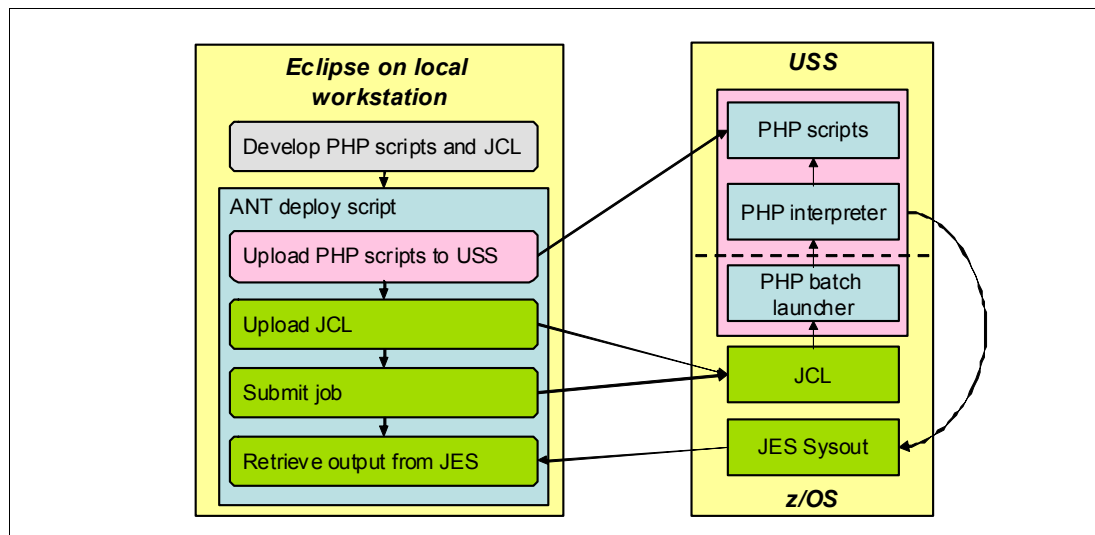


Figure 9-3 PHP z/OS batch development with Eclipse

To deploy and execute the locally developed scripts on z/OS afterwards, we used an Ant script that executes the following steps:

1. Upload all locally developed *.php files using FTP in ASCII mode to a z/OS UNIX System Services directory.

Optionally, *.php files from other external PHP libraries can also be uploaded into a separate z/OS UNIX System Services directory.

2. Upload a JCL to an MVS data set using FTP in ASCII. This JCL is calling a batch launcher like BPXBATCH to execute PHP under UNIX System Services.
3. Submit the job with a Java program, which is like using the Jakarta Commons Net FTP classes to submit the job using FTP.
4. Retrieve the Output using FTP using the same program described in step 3.

All steps are completely automated, that means one click deploys, submits and retrieves the output of the PHP batch. We provide a complete explanation, including the sample code, in the next section.

9.4 Sample application for using PHP in stand-alone batch

In this section, we provide a sample application that shows how to use PHP in batch under z/OS to enrich an existing batch environment with more functionality.

Figure 9-4 shows what the sample is doing. Basically, it is the same sample as provided in 7.5, “Sample stand-alone Java batch application” on page 82. The only difference is that the PDF file creation job and the e-Mail send job are implemented in PHP instead of Java.

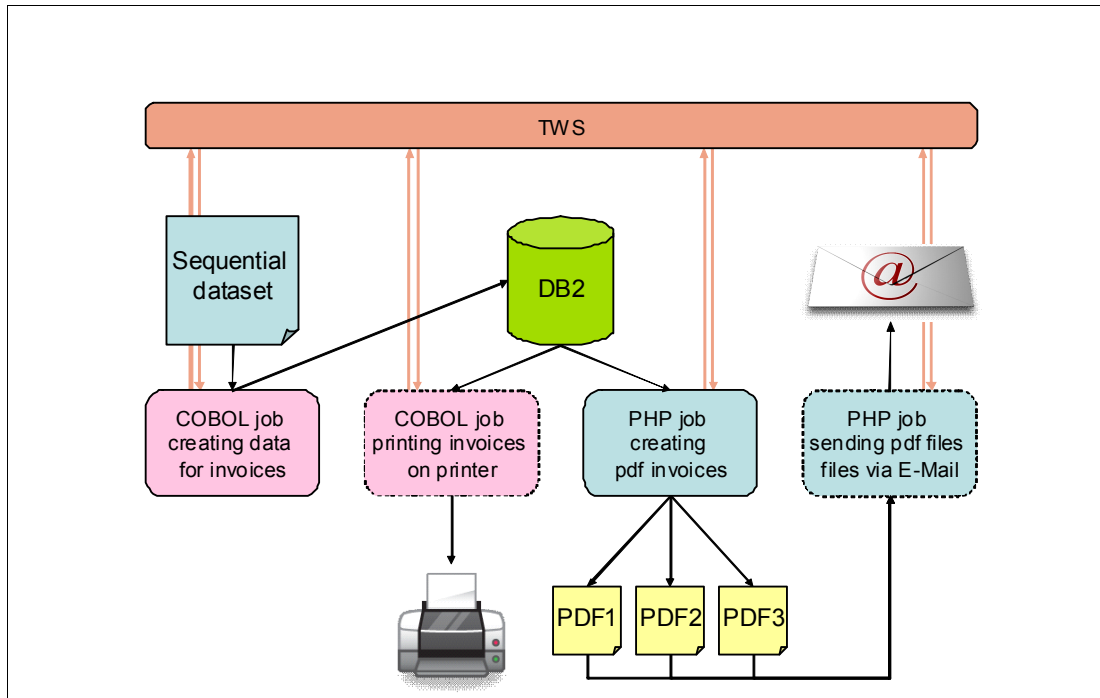


Figure 9-4 Stand-alone PHP batch sample overview

The ERM model for the DB2 database and the data creation job in COBOL remain the same. For convenience, we also included it in the sample Eclipse project part of the Additional Material as described in “Importing and customizing the sample project” on page 164 later on.

On the following pages, we will explain how to implement this sample with the development environment explained in 9.3, “Development tools” on page 160.

9.4.1 Eclipse setup

To set up the Eclipse environment, follow these steps:

1. Set up a proper Eclipse environment for PHP development by downloading the PHP Development Tools (PDT), which are available at the following Web site:

<http://www.eclipse.org/pdt/downloads/>

We use the “PDT 2.1 All In Ones / EPP PHP Package” for our sample. Decompress the package, and start `eclipse.exe`. Then, select **Help** → **Install New Software** to open the Eclipse Java Development Tools. These tools are required for proper Ant support.

2. In the “Work with:” field, enter the following site, as shown in Figure 9-5:

<http://download.eclipse.org/releases/ganymede/>

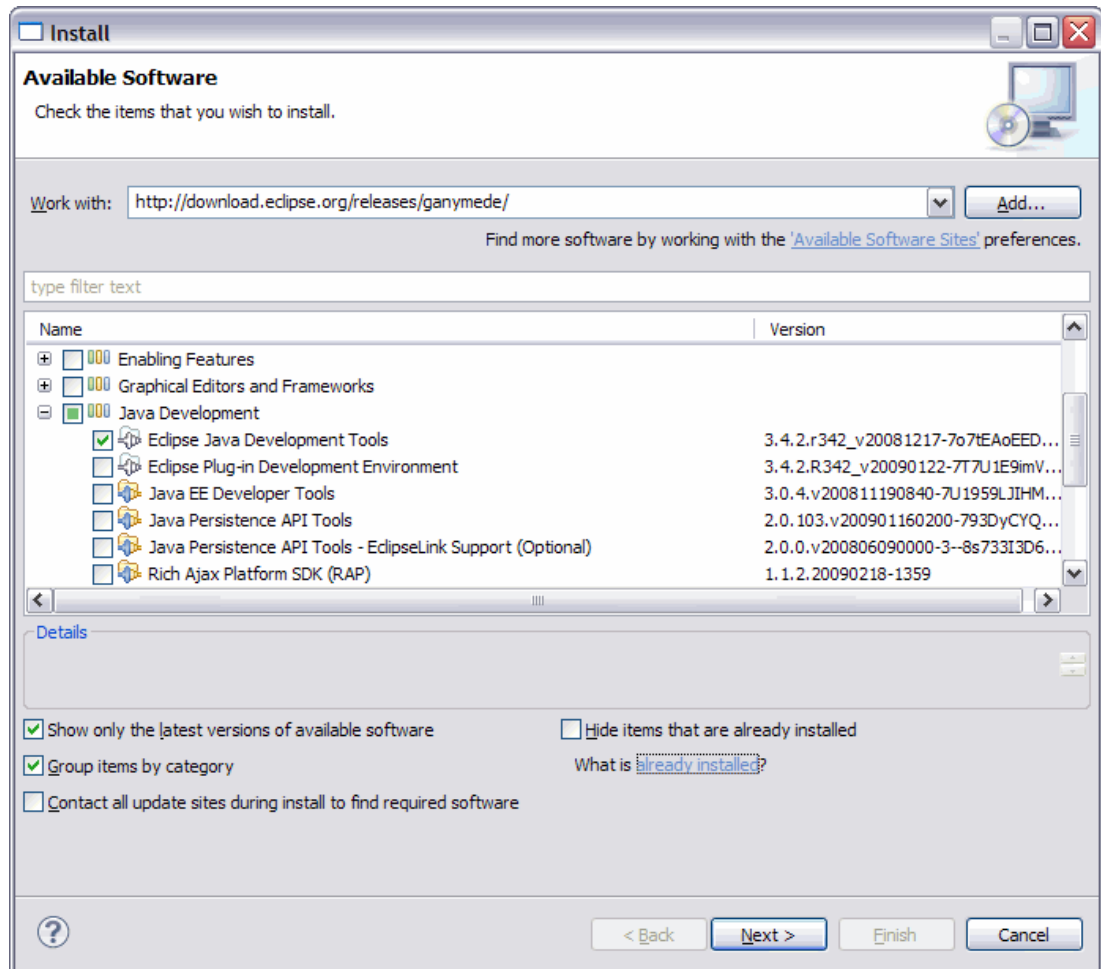


Figure 9-5 Update of Eclipse PDT with Java Development Tools

3. Select the Eclipse Java Development Tools as shown in Figure 9-5 on page 162, click **Next**, and follow the instructions to install the package.

To facilitate the application deployment from the Eclipse workbench, the Ant FTP support in Eclipse needs to be included in the classpath.

Download `commons-net-2.0.zip` and `jakarta-oro-2.0.8.zip` from:

- http://commons.apache.org/downloads/download_net.cgi
- http://jakarta.apache.org/site/downloads/downloads_oro.cgi

Then, extract commons-net-2.0.jar and jakarta-oro-2.0.8.jar from the compressed files, and put them into a directory called c:\ant_ftp_jars.

4. Open Eclipse, and select **Window** → **Preferences** → **Ant** → **Runtime**. On the classpath tab, select **Ant Home Entries**, and click **Add External JARs** as shown in Figure 9-6.

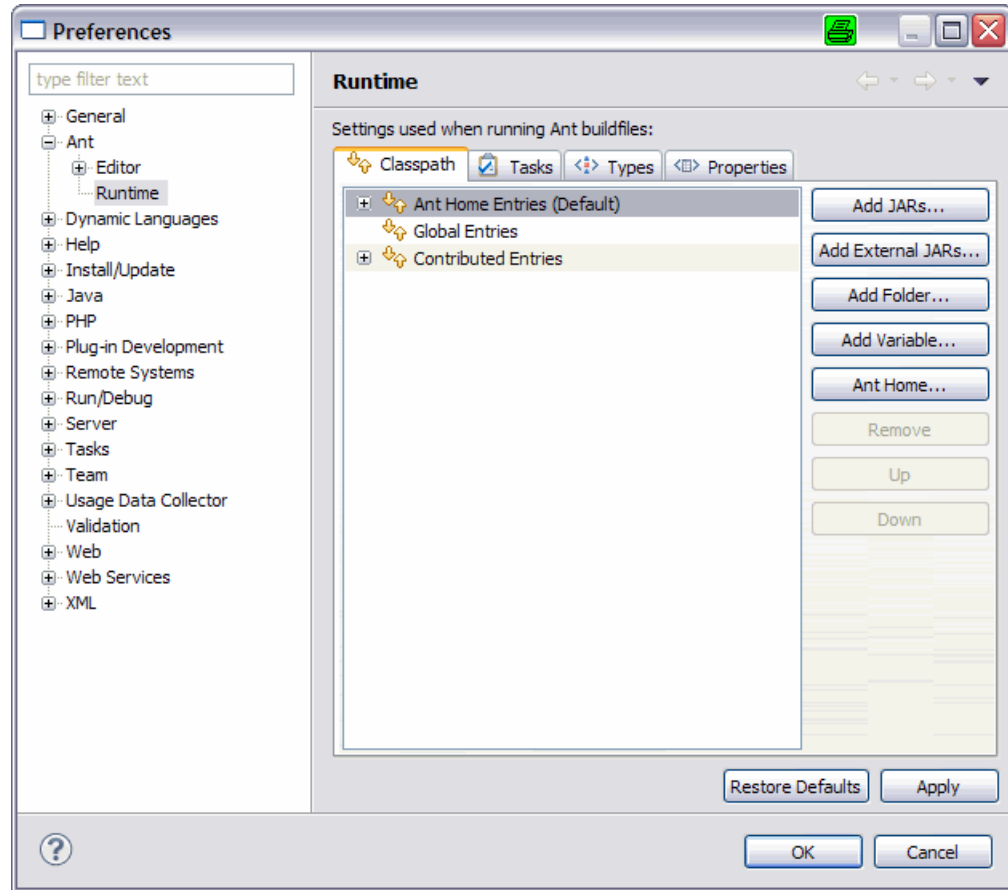


Figure 9-6 Eclipse Ant Classpath

5. Select the two JAR files in the c:\ant_ftp_jars directory, and click **Open**. The Ant FTP support JAR files are added to the Ant classpath. Click **OK**.

Now, Eclipse is ready for PHP development with Ant support.

9.4.2 Importing and customizing the sample project

To start with the project, use the `PHP_pdf_generator.zip` as described in Appendix C, “Additional material” on page 453. Then, follow these steps:

1. Switch to Eclipse, and select **File** → **Import**.
2. Select **General** → **Existing Project into Workspace** as shown Figure 9-7, and click **Next**.

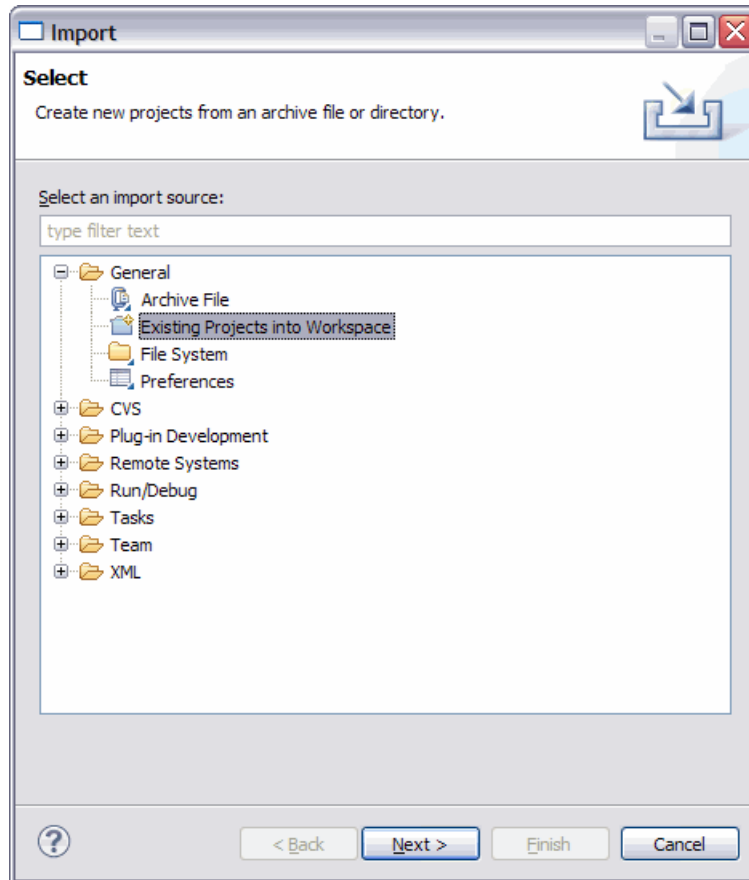


Figure 9-7 Import of sample project into Eclipse

3. In the window that opens, select the “Select archive file” option, and then select the compressed file using the **Browse** button as shown in Figure 9-8. Click **Finish**. Now the project is imported.

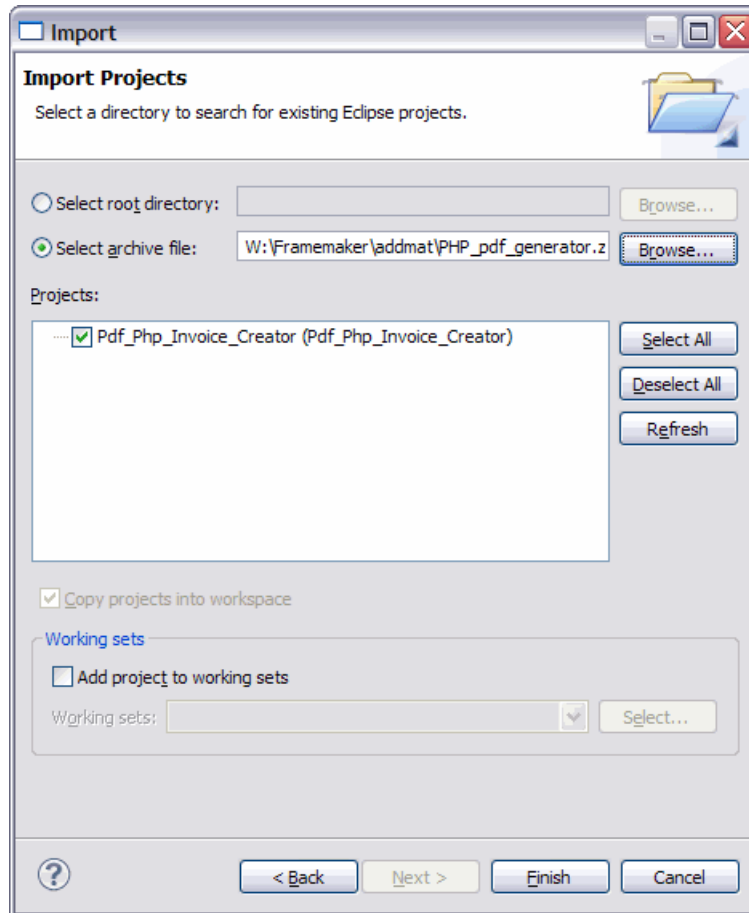


Figure 9-8 Import of sample project into Eclipse 2

Figure 9-9 shows how the project is organized.

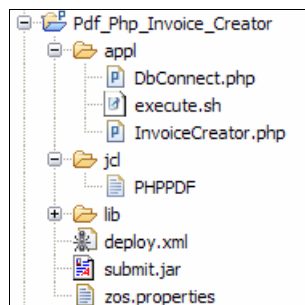


Figure 9-9 Eclipse project overview

The project consists of the following folders and files:

appl	Contains the following *.php source files:
	DbConnect.php A .php script to connect DB2 z/OS.
	execute.sh A shell script that sets environment variables, changes permission bits and executes the PHP program.
	InvoiceCreator.php The main program that creates the PDF files.
jcl	Contains the JCL to start the job.
PHPPDF	The JCL that starts the PHP job with a batch launcher.
lib	Stores *.php library files, for example for PDF creation.
deploy.xml	The Ant script that performs the steps described in 9.3, "Development tools" on page 160.

Important: You do not need to modify the `deploy.xml` script. All parameters, such as destination directories, are managed by an external properties file.

submit.jar	The Java program that is called by the Ant script. It submits the job using FTP and retrieves the job output back to Eclipse.
zos.properties	Contains all variables that are required by the Ant script.

Customizing the `zos.properties` file

First, you need to customize the `zos.properties` file by adjusting the following parameters:

server	The DNS name or IP address of the z/OS system to where you want to deploy the PHP application.
userid	A TSO user name that allows you to log in using FTP on the z/OS destination system.
password	The corresponding password for the user ID.
jcl.dsn	A partitioned data set where you can store the JCL as member.

Note: This partitioned data set must already exist or be newly allocated.

appl.home	The UNIX System Services directory where you want to store the PHP application.
jclPath	For each Eclipse project, the JCL file names need to be different to avoid overwriting members with the same name on the host. For that reason, you can change the path in the <code>zos.properties</code> file.
waittime	If a PHP batch job runs very long, the Ant task that tries to retrieve the job output from JES using FTP might run into a timeout. In that case, you can use this parameter to specify a wait time in seconds.

Example 9-1 shows an example `zos.properties` file.

Example 9-1 Sample `zos.properties` file

```
# Customize this file or move a customized copy of this file up to
# either the Eclipse workspace directory or your "home directory"
server = wtsc48oe.itso.ibm.com
userid = STRAUER
```

```

password = PASSWORD

# The JCL PDS dataset, which must be in single quotes:
jcl.dsn = 'STRAUER.SAMPLE.JCL'

# the home directory for deploying the application jar
appl.home = /u/strauer/php

# local JCL pathes
jclPath = jcl/PHPPDF

# time to wait for output in seconds
waittime = 0 for output in seconds
waittime = 0

```

JCL

Next, you need to customize the JCL. Open the PHPPDF file in the jcl folder. In this JCL, customize the following lines:

- ▶ Job name

The job name has be your user ID plus exactly one character. Otherwise, we cannot submit the job using FTP.
- ▶ PARM statement in STEP1 and STEP2

We have to adjust the path to point <appl_home>/appl/execute whereby <appl_home> is the value of appl.home set in zos.properties.

Example 9-2 shows a sample JCL.

Example 9-2 Sample JCL to submit PHP job

```

//STRAUERA JOB
//*JOBPARM SYSAFF=SC48,L=9999
//*****
//* Run PHP under a UNIX System Service shell
//*****
//HOLD OUTPUT JESDS=ALL,DEFAULT=Y,OUTDISP=(HOLD,HOLD)
//STEP1 EXEC PGM=BPXBATCH,
// PARM='SH chmod 755 /u/strauer/php/appl/execute.sh'
//STEP2 EXEC PGM=BPXBATCH,
// PARM='SH /u/strauer/php/appl/execute.sh'
//STDIN DD DUMMY
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//

```

Note: We use BPXBATCH in our sample. Of course, you can also use one of the other batch launchers listed in Table 9-1 on page 159.

9.4.3 Implementing and deploying the PHP application

To create PDF files in PHP, we use the FPDF Library. First, download the V1.6 compressed file from the following Web site:

<http://www.fpdf.org>

Then, decompress the following files into the `lib` folder of the project:

- ▶ `fpdf.php`
- ▶ All files in the `font` directory and its subdirectories

To see the files in Eclipse, right-click the project, and select **Refresh** as shown in Figure 9-10. Now, all the required files of the FPDF Library displays in the `lib` folder in Eclipse.

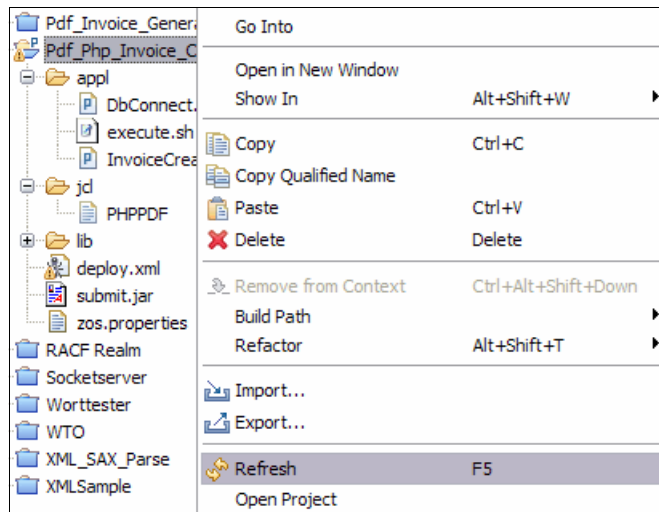


Figure 9-10 Refreshing Eclipse project

You can find the source code that we developed to read the DB2 z/OS data and to create PDF files, which consists of `DbConnect.php` and `InvoiceCreator.php`, in the `src` folder and in “PHP PDF creator” on page 442.

Assuming that ODBC for DB2 z/OS is set up properly, you need to change `$dsn`, `$username`, and `$passwd` in `DbConnect.php`, as shown in Example 9-3.

Example 9-3 Parameters to connect to DB2 z/OS

```
$dsn='odbc:DB9G';  
$username='STRAUER';  
$passwd='PASSWORD';
```

Because the JCL calls the PHP application using the `execute.sh` shell script, you need to customize it for your environment. The shell script consists of the following statements:

- ▶ A `PATH` variable export pointing to the `bin` directory of PHP.
- ▶ A `STEPLIB` variable export pointing `SDSNLOAD` and `SDSNLOD2` for proper ODBC usage.
- ▶ A `DSNAOINI` export pointing to the ODBC ini file because ODBC would not work without it.

- ▶ A chmod of the DbConnect.php file because the file should only be accessible by the owner because it contains the user name and password for DB2.
- ▶ The final call of the PHP script, including a directory path as a parameter to the PHP script. The created PDF files are stored in this directory.

You need to customize these statements to your environment. Example 9-4 shows the customization for our example.

Example 9-4 Sample execute.sh shell script

```
export PATH=$PATH:/usr/lpp/php/bin
export STEPLIB=$STEPLIB:DB9G9.SDSNLOAD:DB9G9.SDSNLOD2
export DSNAOINI="/u/strauer/php/odbc.ini"
chmod 700 /u/strauer/php/app1/DbConnect.php
php /u/strauer/php/app1/InvoiceCreator.php /u/strauer/php/pdf
```

Now, you can deploy the application by right-clicking deploy.xml, and selecting **Run As** → **Ant Build** as shown in Figure 9-11.

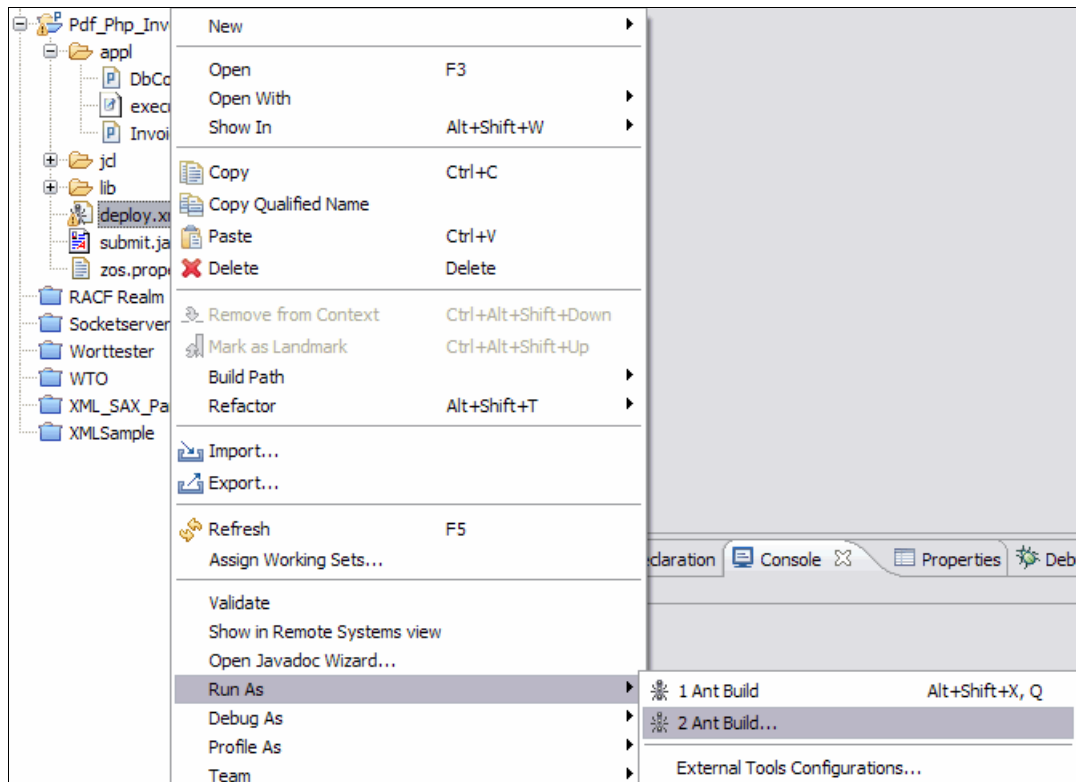


Figure 9-11 Eclipse PHP Deployment with Ant

Then, select which deployment targets to use (see Figure 9-12). The default is all. We select **Run** to execute the Ant script.

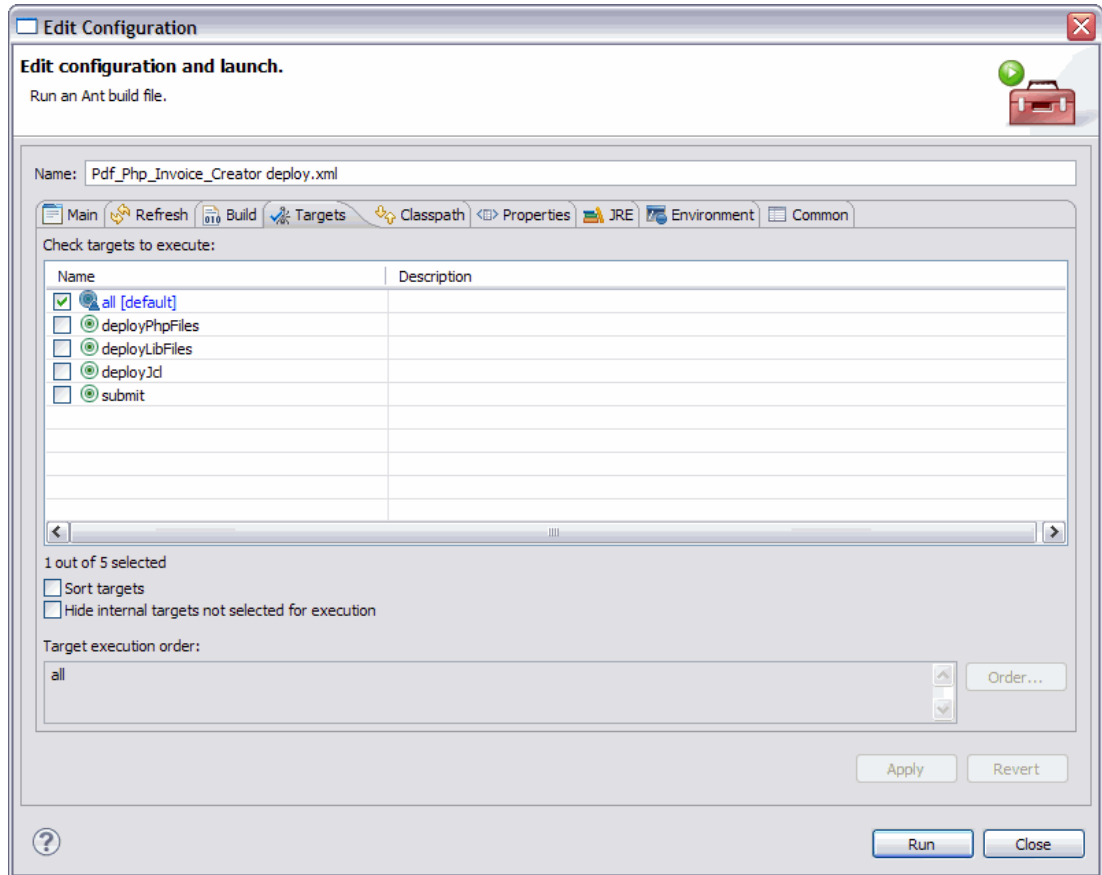


Figure 9-12 Eclipse PHP Deployment with Ant 2

Example 9-5 shows the output on the console.

Example 9-5 PHP batch job output

```
[...]  
[java] PATH is /usr/lpp/Printsrv/bin:/bin:/usr/sbin:/usr/lpp/java/J5.0/bin  
[java] X-Powered-By: PHP/5.1.2  
[java] Content-type: text/html  
[java] Connecting to DB2  
[java] ... Connected  
[java] Starting to create pdf file /u/strauer/php/pdf/Invoice_No_1.pdf...  
[java] Starting to create pdf file /u/strauer/php/pdf/Invoice_No_2.pdf...  
[java] Starting to create pdf file /u/strauer/php/pdf/Invoice_No_3.pdf...  
[java] Starting to create pdf file /u/strauer/php/pdf/Invoice_No_7.pdf...  
[java] Starting to create pdf file /u/strauer/php/pdf/Invoice_No_8.pdf...  
[java] !! END OF JES SPOOL FILE !!  
[...]
```

Finally, you can check the results. Figure 9-13 shows how one batch created PDF files.

John Fellow Nicestreet 345 23456 Boston, MA USA		Bobs Batch Store JES Street 1111 12601 Poughkeepsie, NY USA		
Pos	Quantity	Description	Single price	Total price
1	2	Espresso Machine	632.30 USD	1,264.60 USD
2	1	Sandwich Maker	82.50 USD	82.50 USD
3	3	950 Watts Microwave Oven	138.00 USD	414.00 USD
4	2	12-Cup Coffee Maker	18.00 USD	36.00 USD
Total				1,797.10 USD

Please read our terms and conditions.

Figure 9-13 PDF invoice created in PHP

Note: The PHP script produces PDF files that are encoded in EBCDIC. To get around this, we identified the following different options:

- ▶ Download the PDF files in ASCII mode using FTP (this will do an automatic code page conversion to ASCII) for further processing by other applications.
- ▶ Implement an `iconv` job in UNIX System Services.
- ▶ Modify the FPDF PHP script directly to use the PHP function `iconv` when performing file system output. That means we already do a conversion from IBM-1047 to ISO8859-1 before writing to the file system.

The next step is to implement another job in PHP that takes all of these PDF files and sends them to their recipients.

To sum it up, this sample application shows how PHP together with a batch launcher can be used to easily get additional functionality in a batch environment.



Summary of new functional requirements in z/OS batch

Based on the assumption that a business regularly requires new functionality for a batch environment, in this part of the book, we discussed various approaches on z/OS to make use of a broadened set of technology. Each solution that we discussed has its advantages and disadvantages. Your choice depends on which solution best serves the requirements.

At a minimum, we suggest that you consider the following criteria:

► **Functionality provided by the programming language**

Depending on your current and future batch functional requirements, you can choose between the following main options regarding the programming language:

– **COBOL and PL/I**

These languages offer excellent capabilities in a typical data driven environment. Furthermore, all of them also offer basic XML capabilities. But when it comes to functionality such as PDF creation, using e-Mail and remote access, things can become difficult to implement.

– **Java**

Java offers an extremely broad functionality in all kinds of runtime environments. There are lots of built-in functions in Java available, but also a huge number of software vendors and Open Source projects provide Java libraries.

– **PHP**

As PHP is widely used by a huge community of developers, it also provides state-of the art technology for all kinds of current functional requirements.

► **Functionality provided by the run time**

Compared to stand-alone batch, a run time such as WebSphere XD Compute Grid can offer additional functionality through the Java EE model, for example managed security.

► Available skills

While talking about different programming languages and runtime environments, skills can become a critical factor, such as:

- What kind of programming skill is available in your company?
- Is enough skill for the runtime environment (CICS, IMS, DB2, and WebSphere XD Compute Grid) available?
- How will the situation be in the future? Will people retire, for example? If yes, a strategy for a new programming language or runtime environment might be evaluated.

► IT Strategy

How does each scenario fit into your IT strategy (for example, do you have a Java EE strategy)?

► Platform independency

If platform independency is important for you, PHP or Java in every z/OS run time might be the preferred language compared to COBOL, PL/I, or Assembler.

► Runtime cost

Depending on which approach you choose, you might get a substantially different cost to run a certain application with different technologies. Some factors that influence the (run time) cost are:

- Java runs largely on zAAPs, and if you have only zIIP engines and no zAAP engines, you can use the zAAP on zIIP solution from IBM.
- If you use XML in COBOL, PL/I, or Assembler, the XML parsing part can also run on a zIIP
- PHP only runs on General Purpose Processors (GPs)

► Development cost

Development cost can differ depending on the programming language. Important factors determining development cost are:

- Skills availability (skills that are widely available tend to be cheaper than skills that are scarce)
- Tools productivity
- Functionality of the language
- Robustness of the technology (troubleshooting takes a lot of time)

► Development tools

All solutions offer very good tool capabilities to develop batch applications for z/OS, so we do not expect the tooling to be a big factor in making a decision between the options discussed in this part of the book.

► Access to z/OS data

COBOL, PL/I, C/C++, and Java in all run times offer APIs for accessing z/OS data. However, access to native z/OS data (MVS data sets) is supported in an easier and more natural way in COBOL, PL/I, and Assembler than in Java, C/C++ or PHP. Especially in a high-volume batch environment with lots of I/O on MVS data sets this can become a key decision factor. In summary:

- COBOL, PL/I, and Assembler provide the most natural access to native z/OS data
- C/C++ provides access for all types of data on z/OS too

- Java programs in stand-alone batch, CICS, IMS or DB2 stored procedures can access z/OS data sets and VSAM files using specific classes, and IMS and DB2 data on z/OS using a native high performance JDBC driver
 - In addition to the Java data access methods above, Java in WebSphere XD Compute Grid can furthermore exploit Java EE features to externalize data sources
 - PHP only allows access to z/OS UNIX System Services files and DB2 on z/OS. There is no support for accessing MVS data sets.
- ▶ Workload scheduler integration

Very often, jobs are managed by a workload scheduler. All discussed solutions offer the possibility to submit jobs using a JCL, that means they can easily be integrated into a workload scheduler like Tivoli Workload Scheduler. The integration might only differ in small areas, for example if you use a DB2 stored procedure for batch processing, you would have to wait for the output in JES until the stored procedure terminates.
 - ▶ Transactionality

If Two Phase Commit is required for your batch application, you carefully have to think about your runtime environment. For example, all stand-alone alternatives offer no transactionality. In contrast, containers like IMS, DB2 and WebSphere XD Compute Grid have the ability to manage transactions. Depending on your exact requirements, these would have to be examined in more detail concerning transactionality.
 - ▶ Checkpointing

As batch jobs can run very long, checkpoints can become a very important criteria to avoid very complex rollbacks or locking aspects. For stand-alone batch applications, this has to be implemented manually. A container such as IMS or WebSphere XD Compute Grid in contrast already provide this functionality.
 - ▶ JVM startup cost

Stand-alone Java batch or Java in IMS cause a JVM startup for every job. If the jobs are long running tasks, this is negligible. But if you have many short running Java batch jobs, you might consider WebSphere XD Compute Grid or a Java DB2 stored procedure, which efficiently reuse JVMs.
 - ▶ Performance

Different programming languages and different programming models often have a big difference in performance. This also has to be taken into account for new functionality in batch. In certain situations performance could be the determining factor to choose for a certain technology.

All those criteria have to be kept in mind while choosing a technology for new functionality in batch. Besides these distinction criteria, you should remember that all solutions can leverage the high availability and scalability concept of the z/OS operating system and its stack products.

To sum it up, z/OS offers a lot of excellent capabilities to implement state of the art batch functionality.



Batch environment enhancements in z/OS V1R13

In this chapter we discuss various batch environment enhancements in z/OS V1R13. We review the IBM z/OS Batch Runtime which provides the ability to update the DB2 database from both COBOL and Java in a single transaction, and review Java COBOL interoperability.

We also describe the following updates to improve batch job processing:

- ▶ Instream data in PROCs (cataloged and instream)
- ▶ Support for JOBRC (return code)
- ▶ Spin and SPIN data set
- ▶ Requeue a job by command on a step boundary

11.1 Introduction to z/OS Batch Runtime

In current z/OS environments there is a need to re-engineer existing native z/OS COBOL applications to incorporate Java in order to take advantage of its many language features and the large developer skill base.

A mixed z/OS COBOL and Java batch application must share a DB2 connection and use relevant language APIs to communicate with DB2 in the same unit of work (UOW). During the running of such a re-engineered program, Embedded Structured Query Language (SQL) DB2 access in Enterprise COBOL and Java Database Connectivity (JDBC), Structured Query Language for Java (SQLJ) DB2 access in Java, or both, must coexist transparently.

The z/OS Batch Runtime environment is a new option for running batch work in z/OS V1R13. It provides a managed environment for the integration of Java and COBOL, and is consistent with IBM WebSphere-based batch. It enables shared access to a DB2 connection by both COBOL and Java programs, where DB2 COBOL and Java applications share a single database connection. Updates to DB2 are committed in a single transaction. It allows the interoperability of COBOL and Java applications within the same unit of work (UOW) and provides database integrity.

The application runs within the Batch Runtime environment provided with z/OS V1R13 and preserves COBOL assets while migrating to Java applications where required. It also provides commit and rollback services by leveraging the z/OS Recovery Subsystem Services (RSS) and the Java Database Connectivity (JDBC) driver. Executing your DB2 hybrid applications in this container simplifies your code and helps ensure the integrity of your data.

The Java program execution is inserted as a step into the batch job and handles the following tasks:

- ▶ To launch Batch Runtime, a special environment that is able to run the desired Java program.
- ▶ To handle return codes that differ from those of traditional batch. This difference in handling return codes requires modifications that are provided in JES2 of z/OS V1R13, as described in 11.2, “JES2 batch modernization” on page 181“.

As with any other program, this new Batch Runtime environment is started by an initiator. It is then dubbed into a UNIX System Services process in which a JVM is started to run the desired Java program. This entire address space is named a JZOS address space, which is conceptually the equivalent of the overall process for non-interpreted languages such as C or COBOL. Refer to 7.2, “Running Java with JZOS” on page 79.

11.1.1 Java COBOL with DB2 interoperability

Java COBOL with DB2 interoperability support in z/OS V1R13 provides the ability to replace or add functions in current 3GL DB2 (for example, COBOL DB2) application inventory with new Java DB2 code. It requires local attach z/OS DB2 connection sharing for common DB2 access and UOW transactional integrity among the application components. This offers a generalized solution that does not require a specific runtime or middleware. That is, a pure batch environment and its implementation require few or no changes to existing code; it only needs special callbacks for commit and rollback.

The topology of the z/OS Batch Runtime is displayed in Figure 11-1 on page 179. It depicts the job flow from the submission of the JCL to the Java batch application (BCDBATCH) to the z/OS batch container.

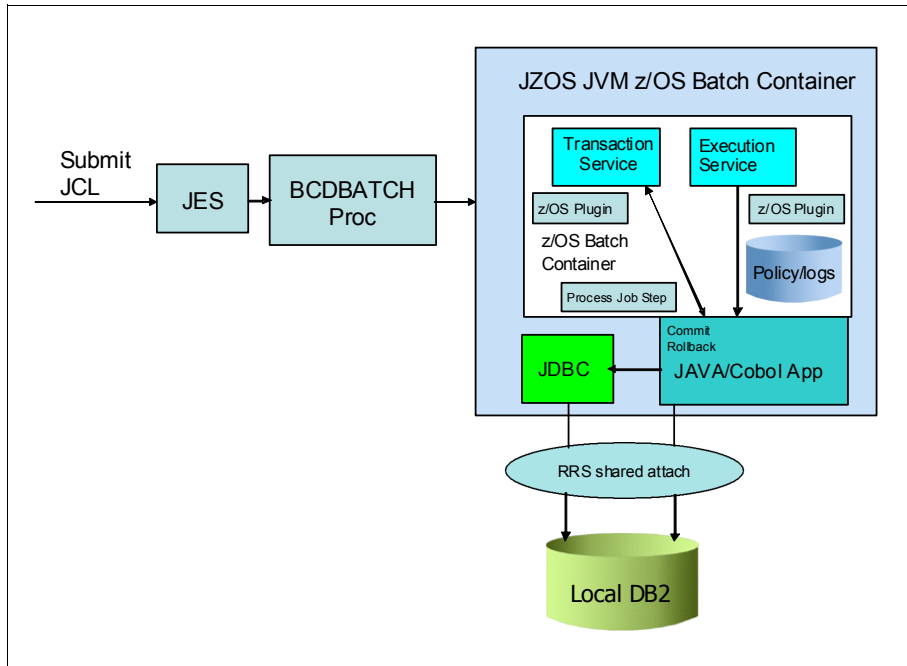


Figure 11-1 Topology of the z/OS Batch Runtime

Usage and invocation

Batch Runtime is invoked through batch JCL. A sample JCL procedure, BCDPROC, is shown in Example 11-1.

Example 11-1 BCDPROC procedure

```
//BCDPROC PROC VERSION='61', JVMLDM version: 61 (Java 6.0.1 31bit)
// LOGLVL='+I', Debug level: +I(info) +T(trc)
// LEPARM='' Language Environment parms
//*
//*****
//* *
//* Proprietary Statement: *
//* *
//* Licensed Materials - Property of IBM
//*
//* 5694-A01 *
//* Copyright IBM Corp. 2011. *
//* *
//* Status = HBB7780 *
//* *
//* Component = z/OS Batch Runtime (SC1BC) *
//* *
//* EXTERNAL CLASSIFICATION = OTHER *
//* END OF EXTERNAL CLASSIFICATION: *
//* *
//* Sample procedure JCL to invoke z/OS Batch Runtime *
//* *
//* Notes: *
//* *
//* 1. Override the VERSION symbolic parameter in your JCL *
//* to match the level of the Java SDK you are running. *
```

```

/** *
/** VERSION=61 Java SDK 6.0.1 (31 bit) *
/** *
/** 2. Override the LOGLVL symbolic parameter to control *
/** the messages issued by the JZOS Java launcher. *
/** *
/** Use the +T option when reporting problems to IBM or *
/** to diagnose problems in the STDENV script. *
/** *
/** 3. Override the LEPARM symbolic parameter to add any *
/** application specific language environment options *
/** needed. *
/** *
/** Change History = *
/** *
/** $LO=BATCH,HBB7780,100324,KDKJ: *
/** *
/** *
/*******
/**JAVA EXEC PGM=JVMLDM&VERSION,REGION=OM,
/** PARM='&LEPARM/&LOGLVL'
/**
/**SYSPRINT DD SYSOUT=* System stdout
/**SYSOUT DD SYSOUT=* System stderr
/**STDOUT DD SYSOUT=* Java System.out
/**STDERR DD SYSOUT=* Java System.err
/**BCDOUT DD SYSOUT=* Batch container messages
/**BCDTRACE DD SYSOUT=* Batch container trace
/**
/**CEEDUMP DD SYSOUT=*
/**

```

z/OS batch runtime is invoked through JCL that invokes the job BCDBATCH, which invokes the JZOS launcher to initialize the Java environment. Because BCDBATCH invokes JZOS, one level of the JZOS launcher exists for each Java SDK level and bit mode. You define the level with a symbolic, and your installation can update the symbolic as new levels of the Java SDK are added or made the default.

11.1.2 JZOS Batch Launcher enhancements

The enhancements to the JZOS Batch Launcher and Toolkit in z/OS V1R13 are:

- ▶ Support for z/OS Java SDK 1.4.2 (31-bit), z/OS Java SDK 5.0, SDK 6.0.0, (31-bit and 64-bit) and SDK 6.0.1 (31-bit and 64-bit).
- ▶ The ability to access z/OS Workload Manager (WLM) services
- ▶ The ability to submit z/OS batch jobs from Java

11.2 JES2 batch modernization

Enhancements were required to fully support Java applications in batch environments. Various updates were made in z/OS V1R13 to improve BATCH job processing, which include support for:

- ▶ Instream data in PROCs
- ▶ Control of job return codes
- ▶ Spin and SPIN data set
- ▶ Requeue a job by command on a step boundary

11.2.1 Instream data in PROCs and INCLUDEs

Support was added in z/OS V1R13 for instream data sets in JCL PROCs and INCLUDEs that provides batch modernization for Java.

- ▶ It simplified the writing of JCL PROCs to allow JCL coders to combine the JCL and control data sets in one PROC member.
- ▶ Support was added to allow instream data sets to be created when processing DD DATA or DD * JCL within proclibs or INCLUDEs.
 - The DD * and DD DATA JCL cards and all their operands can now be placed in a JCL PROC followed by the instream data. During conversion processing, this instream data is stripped out and placed into a JES2 instream data set. When the job runs, it can access this data set like it would an instream data set in a normal JCL stream.
 - The difference with standard instream support is that it does not automatically generate a //SYSIN DD * card. If a non-JCL card is encountered in a PROC or INLCUDE outside a DD * or DD DATA it continues to be flagged as an error.
- ▶ Supports all users of PROCs started tasks and batch jobs. However, it only works for jobs running under a JES2 subsystem, it does not work if the job is run under the master subsystem.
- ▶ Once z/OS V1R13 is installed, it is required that the job convert on a z/OS V1R13 member. It can later execute on any level member.
- ▶ Support was added for both cataloged and instream procedures; see Example 11-2.

Example 11-2 Example of a new procedure

```
//ABC JOB
//INCLUDE MEMBER=HELLO
//STEPS EXEC PGM=IEBGENER
//SYSIN DD DUMMY
//SYSPRINT DD SYSOUT=A
//SYSUT2 DD SYSOUT=A
//SYSUT1 DD DATA
HELLO WORLD
/*
```

Instream data sets in PROCs

New SYSIN data sets are included in the extended status DSLIST function. They are not included in the SPOOL data set browse of JCLIN and are not part of the original JCL submitted. They are not transmitted to other nodes or offloaded and are not part of the original JCL submitted.

Instream data sets are not part of the submitted JCL, so they are not included when looking at the original JCL and are not sent over NJE to other nodes. But they do appear in the data set list. Instream data sets work for batch jobs and started tasks. The support in z/OS V1R13 provides the ability for conceptual control data sets, where the application parameters reside, to not be separated from proclibs or INCLUDEs.

See Example 11-3 for an instream data in PROC example.

Example 11-3 Instream data in PROC

```
//HELLO  PROC
//STEP1  EXEC ASMHCLG
//C.SYSIN DD *
TEST    CSECT ,
        STM  14,12,12(13)
        BALR 12,0
        USING *,12
        ST   13,SAVAREA+4
        LA   13,SAVAREA
        SPACE 1
        WTO  'Hello world!'
        SPACE 1
        L    13,SAVAREA+4
        LM   14,12,12(13)
        SR   15,15
        BR   14
        SPACE 1
SAVAREA DC  18F'0'
        END
//L.TEST DD DUMMY
//L.SYSXX DD *
//      PEND
```

Example 11-4 shows a simple example of a PROC that calls another nested PROC and has an instream input. This can be run in a batch job or by just doing a S HELLO.

Example 11-4 PROC calling a nested PROC

```
s hello
$HASP100 HELLO    ON STCINRDR
$HASP373 HELLO    STARTED
+Hello world!
$HASP395 HELLO    ENDED
```

Example 11-5 is an example of instream data in PROCs and INCLUDEs for JES2 only. It requires both z/OS V1R13 and JES2 1.13 on the converting system and the initiating system. This is not supported for MSTR subsystems or by JES3.

Example 11-5 Instream data in PROCs and INCLUDEs

```
//PROC1  PROC
//ASTEP  EXEC PGM=xyz
//DD1    DD *
This is instream data
//      PEND
```

11.2.2 Support for JOBRC (job return code)

The success or failure of a JCL stream is not always determined by the highest completion code of any step. The highest completion code of a job is not necessarily the most meaningful. Sometimes it is the completion code of the last step or the completion code of a specific step. A new JCL keyword was added to control the reported completion code. The new JOBRC= operand on the JOB card now gives the JCL writer the ability to control what completion code is presented for the job. This provides a more useful way to take into account the most meaningful completion code. This new JCL keyword allows you to select the last step, a specific step, or the highest step completion code.

The new JOBRC keyword on the JOB card to control a job is:

```
JOBRC= MAXRC | LASTRC | (STEP,name.name)
```

The possible values for the JOBRC keyword are:

- ▶ MAXRC – Default, the job return code is the maximum of any step.
- ▶ LASTRC – The job return code is the return code of the last step.
- ▶ (STEP, stepname.procstepname) – The job return code of the identified step; if the step does not execute it defaults to MAXRC.

The \$T JOBCLASS(x) command has also been enhanced with a new JOBRC operand to affect processing for all jobs in the job class:

```
JOBCLASS JOBRC= MAXRC | LASTRC
```

Note: The JOBRC keyword on the job card takes precedence.

Job return code

The job return code is presented in ENF 70, \$DJ CC= command, extended status (SDSF) and the \$HASP165 message:

- ▶ \$DJQ,CC= value
- ▶ ENF 70 field ENF70_MAXCC

Extended status STTRMXRC

The extended status max return code field STTRMXRC was updated to return the new JOBRC information. It now includes two new conditions regarding how a job can end:

- ▶ A new bit indicates that the JOBRC specification affected the completion code.
- ▶ The old maximum return code is available in the verbose job STVBMXRC field.
- ▶ The IAZSS2 (SAPI) fields SSS2MXRC and SSS2LSAB are not affected.

Origin is JCTMAXRC and JCTLSTAB, which are not changed.

You can get new combinations of conditions:

- ▶ JOB ABENDED but has a condition code from:
 - A step that was named in JOBRC=(STEP,xxxx) completed
 - A final step that specified COND= ONLY or EVEN and JOBRC=LASTRC
- ▶ Job EOM similar to ABEND if JOBRC=(STEP,xxxx) and step completed

- ▶ Ended by CC and completion code from JOBRC=(STEP,x)
Not CC from the step that caused the job to end.

Processing for extended status is not affected by this support. It returns a different view of the job completion code. It does not indicate how a job ended, but the last ABEND code and a high return code.

An effect of these changes is that you can get new combinations of return codes that might not have been expected. This can happen when the job ended abnormally but the step specified on JOBRC ran normally. To ensure that the correct data is reported, use the STTRXAB and STTRXCDE bits to interpret the information in the 3-byte code field STTRMXCC.

Note: A JES2 cancel command, a JCL error, or a system failure overrides any JOBRC or maximum return code for a job.

HASP165 message text

The HASP165 message was updated to accommodate the new JOBRC processing:

- ▶ To always display the job completion code as a 4-digit value (with leading zeros).
- ▶ To indicate whether the displayed completion code is the maximum completion code for the job or whether it was affected by the JOBRC processing (last or specific step completion code).

This helps identify if a specific step was requested in JOBRC and that step did not run, resulting in the maximum return code being returned.

The updated HASP165 message text:

Jobname ENDED AT node reason

Examples of reason:

- ▶ MAXCC=code - JOBRC was not specified.
Code is now always 4 digits, MAXCC=0000
- ▶ JOBRC=code - JOBRC was specified and affected the return code.
- ▶ MAXRC=code - JOBRC was specified but MAXRC was returned.
- ▶ ABENDED Sxxx,Uyyy.
- ▶ ABENDED abend_code, JOBRC=code.
JOBRC=(STEP,stepname), step executed, but later step ABENDED.

Another case that can occur is that the JOBRC requests the return code from a specific step but the job also ABENDs. In this case, the ABEND code and the JOBRC are both presented in the HASP165. Two additional job error case return codes were defined:

- ▶ Converter error – The converter returned a bad return code and failed the job.
- ▶ System failure – The job was executing at the time of a system failure and was not requeued for execution.

Usage and invocation

The use and invocation with this new option in z/OS V1R13:

- ▶ If JOBRC=LASTRC is specified, this specification indicates to use the return code of the last executed step as the completion code for the job.

- ▶ If `JOBRC=(STEP,C.HLASM)` is specified, the return code for the C step in the HLASM procstepname is used as the completion code for the job.

Benefit and value

By using `JOBRC`, you can better determine the success of a job without having to examine the job output.

11.2.3 Spin and SPIN data set

JESLOG SPIN processing was added in z/OS 1.2 to allow long-running jobs to spin their job log and system messages data sets, and provided the ability to specify the automatic spin options in JCL. z/OS V1R13 extended the JESLOG support to any SPIN data set the job may allocate, even those with large amounts of SPOOL space.

The `SPIN=` operand on the DD statement was enhanced with an optional value to indicate when a SPIN data set should be spun off.

`SPIN=(UNALLOC,option)`

where

- 'hh:mm' - Spin at specific time
- +hh:mm' - Spin every hh:mm interval
- nnn, nnnK, nnnM - Spin every nnn lines
- NOCMND - Cannot be spun by command
- CMNDONLY - Can be spun via operator command (default if no interval)

This function can be disabled by specifying `NOCMND`. By default spin data sets can be spun by operator command. The `DDNAME=` operand was added to the `$TJ.SPIN` command to spin a specific data set.

Note: No application code or JCL change is required.

Benefit and value

Using `SPIN` eliminates the requirement to take down a long-running process to release spool space.

11.2.4 Evict a job on a step boundary

Long-running jobs can be an issue when an installation is trying to shut down a system. z/OS V1R13 batch modernization provides an orderly method of getting jobs out of execution. This enhancement can help jobs that have multiple long-running steps. A new operand has been implemented on the `$E J` command that forces a job out of execution when the current step ends. The job resumes execution from the next step.

A new `STEP` operand on the `$EJ` command is provided:

`$EJxxxx,STEP[,HOLD]`

This option deals with cross-member requests in the following way:

- ▶ It forces a job out of execution when the current step ends.
- ▶ The optional `HOLD` operand makes the job held.
- ▶ The job is requeued for execution and held, if requested.

- ▶ It requires the JES journal to be active:
 - JOBCLASS(x) JOURNAL=YES
- ▶ The option utilizes existing restart logic of z/OS to restart the job from the next step.
Previously used to restart jobs after an IPL
- ▶ It uses the new step end SSI to communicate with the initiator to requeue the job:
 - New JES2 exit 58 called in step end SSI.
 - This can be used to inhibit or to trigger the function.

Benefit and value

This provides more orderly shutdowns of systems to enhance batch processing.

In summary, z/OS V1R13 offers various batch environment enhancements to improve batch job processing.



Part 3

Implement agile batch

In this part of the book, we discuss a collection of topics that are related to the requirement of being able to run batch processes any time. As previously mentioned earlier in this book, many times batch processes are locked into a batch window, typically at night. There can be different reasons for this type of processing, such as application dependencies, but also technical reasons, such as spreading the workload efficiently across the 24 hours inside a day or not being able to access the same data in a database concurrently with both batch and OLTP processes.

In this part of the book, we do not discuss the topic of application dependencies, because this topic is very specific in every situation. Instead, we focus on the following key areas:

- ▶ The ability to work smarter with data, so that there is more opportunity to run batch processes operating on data during the OLTP window.

In Chapter 12, “Create agile batch by optimizing the Information Management architecture” on page 189, we discuss a number of Information Management solutions that can help to make batch bulk data operations more efficient and also offload direct I/O from master databases onto data extracts or Data Warehouses.

- ▶ Optimizing data access to DB2 databases, so that locking of data and tables is shortened allowing for more concurrent I/O operations on the same data, which increases the potential to run batch and OLTP concurrently while using the same DB2 database.

We discuss best practices in this area in Chapter 13, “Create agile batch by optimizing DB2 access” on page 207.

- ▶ In some environments, there is no easy way or no way at all to trigger a batch job any time during the day, on demand. For this purpose, you need flexible triggering of batch jobs, for example, by just clicking a button in a browser. In certain cases, you might have a requirement for application-driven triggering of a batch job.

In Chapter 14, “Create agile batch by implementing trigger mechanisms” on page 223, we show different ways of implementing batch job triggering. Some of these scenarios keep on exploiting Tivoli Workload Scheduler as the manager of all batch jobs.



Create agile batch by optimizing the Information Management architecture

This chapter discusses architectures to make batch jobs related to data access more agile.

We discuss Data Warehousing on System z and the process of extract, transform, and load (ETL). This process is often implemented on the mainframe using Job Control Language (JCL) batch technology. There is nothing wrong with using batch jobs, but business drivers might force a redesign of ETL process to achieve a higher degree of agile batch.

Today, business optimization is dependent on having trusted information, that is integrated, enterprise ready, and dynamic. Information should be a service throughout the enterprise, similar to electric or water utilities, that are always available and easily accessed when required. This availability and ease of access requires a Data Warehouse that has accurate data that is ready for use by business analysts.

In this chapter, we focus on gaining confidence by creating clean, standardized, and organized data. We will describe ETL techniques that have minimal impact on the online transaction processing (OTLP) environment, that is used to facilitate and manage transaction-oriented applications.

We also discuss IBM InfoSphere DataStage in relation to agile batch. IBM InfoSphere DataStage is particularly useful when you are dealing with many transformations and differences between the source and target data model. For those of you who want to learn about the installation of the Information Server, it is best to read *Enterprise Data Warehousing with DB2 9 for z/OS*, SG24-7637, which describes the setup of InfoSphere DataStage. For more information about the design of InfoSphere DataStage flows, read *IBM InfoSphere DataStage Data Flow and Job Design*, SG24-7576.

This chapter includes the following topics:

- ▶ Data Warehousing on System z
- ▶ ETL

12.1 Data Warehousing on System z

Data Warehouses have become a significant part of today's operational IT environment, where they support decision making systems of the business.

A *Data Warehouse* is a collection of the organization's data with a corporate-wide scope for use in decision support and informational applications. Therefore, a Data Warehouse is designed to serve all possible decision support processes for an organization. Two types of data subsystems make up the *Data Warehouse environment*:

- ▶ The Enterprise Data Warehouse, which contains all data with a global scope
- ▶ The *Data Marts*, which contain data for a specific business

Figure 12-1 shows an example of a Data Warehouse and Business Intelligence (BI) or Decision Support system of the enterprise. This Data Warehouse and BI system provides executives and line managers with information to help support their decisions and to sustain and improve their business.

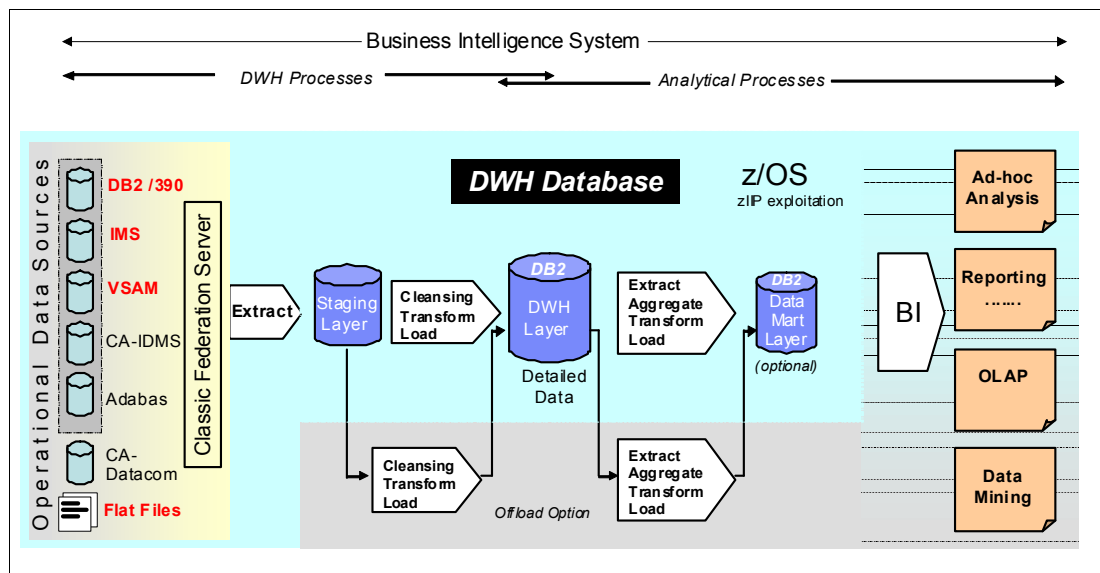


Figure 12-1 Data Warehousing on z/OS: Architectural view

Data Warehouses contain the consolidated information of multiple sources such as financial, marketing and inventory transaction data. Today's Data Warehouses have to support mixed workloads, such as:

- ▶ Continuous data loading
- ▶ Standard reports
- ▶ Ad-hoc query users
- ▶ Business Intelligence analytics
- ▶ Operational Business Intelligence

These workloads can come in large quantities that include the processing of millions of data records, representing millions of business transactions, all which need to be ordered, organized, transformed and analyzed.

Because of the size of Data Warehouses and the fact that we have to support multiple workloads, the System z mainframe is in many cases the best choice to operate as a Data Warehouse DBMS server. Mainly because System z is known for its reliability, availability, security and its protected access to data fulfilling compliance regulation. In addition, the

System z platform has outstanding workload management capabilities, as well as it provides hardware compression to minimize disk space of DB2 for z/OS and improved I/O performance.

To meet today's business requirements IBM has invested heavily in best of breed technologies to provide a unified framework for delivering trusted information. Many of the products IBM has to offer revolve around the process of ETL the data so that it ensures data consistency, data integrity and that it is structured by an enterprise model that ties many different uses of the information together. All this can be achieved on the System z platform as well.

It is very common to see batch jobs as being part of the delivery of information. Especially during the ETL processing and also as being part of the cleansing process. By *cleansing*, we mean the process of detecting and correcting inaccurate data records.

In the next section, we discuss some enhancements that can be made to the ETL process in order to meet today's demand of agile batch.

12.2 ETL

The ETL process is commonly used in enterprises when building a data repository, in most cases a Data Warehouse. In many cases we see this process is being implemented by using batch job procedures. Figure 12-2 illustrates the concept of ETL.

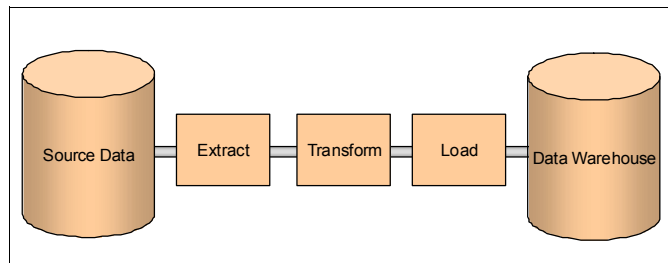


Figure 12-2 ETL concept

ETL is a term that refers to the processes that extracts information from an online transaction processing (OLTP) system and other sources, transforms it according to the needs of the Data Warehousing environment (this can also include quality requirements), and loads it into the Data Warehouse.

The *extraction process* takes the data from various sources. Each of these source systems can use a different data organization, such as relational databases, hierarchical databases, network databases, flat files, and even unstructured data. Depending on the source of the data, you might need different techniques for extraction. The extraction process then converts the data into a format that the transformation process can use.

The *transformation process* applies various rules and functions to the extracted data to change the source data into a form that is useful for the data warehouse. This can include conversion to the required data structure, aggregation, converting code, joining data from multiple sources to derive new values, and so forth. The transformation process can also include *cleansing* of data, such as removing duplicates and standardizing data formats.

In the *load process*, the transformed data is stored into the target, the Data Warehouse. When the Data Warehouse is built for the first time, data needs to be extracted from various sources, transformed, and loaded one time. This is called the initial load. After the initial load,

the Data Warehouse has to be kept relevant by updating the data from the all the sources. This process is known as *data refresh*.

This process includes the following basic stages:

<i>Extract</i>	Retrieve the data from its source. Often, you have to consolidate data from different sources, and in many cases this data is non-relational.
<i>Transform</i>	Make the data fit for operational need. The data quality aspects are usually considered critical.
Load	Put the data into the end target database. The load is often considered the most time consuming phase in the ETL process.

Note: Although the System z10 mainframe has some outstanding design principles in order to process large amounts of data, ETL based batch jobs can bring a lot of complexity to a mainframe environment. The complexity can influence the agility as well, and in many situations it already has. In 12.2.1, “Overcoming operational inefficiency with ETL” on page 193 we take a look at the operational inefficiency some of us experience today.

Traditionally, for data refresh, Data Warehouses extract data from the operational storage through batch processes executed within maintenance windows. The batch extract jobs are run perhaps once a month, but could be run as often as once a week, depending on the business requirement and batch window availability. However, the need to obtain the most recent data is becoming more and more important.

Another challenge in the batch update is the volume of data. A monthly data refresh will have to handle a large volume of data. This will require a much longer batch window. A continuous *trickle feed* might be a better option in this situation. Trickle feed is common “jargon” terminology for continuously feeding the Data Warehouse. As soon as new data is generated in the OLTP system, it is captured and passed to the ETL process in near real time. Using the trickle feed approach, the information in a Data Warehouse is kept as recent as possible and you avoid a batch window bottleneck.

Some of the data extracted from the OLTP window might not need a complex transformation. It can be loaded directly to the Data Warehouse with only minor modifications. In this case, *data replication* might be another option to consider. Data replication captures data from the data source as soon as it is generated, performs minor modifications, and then inserts it into the target database.

In most Data Warehouse environments, the ETL process is a combination of batch ETL, near real time ETL, and replication solutions, depending on the business needs.

The locations where you run the ETL processes are an important consideration. The extract process preferably runs on the platform where the source data is located. The compute-intensive transformation process could run on a different platform, but this implies moving a large amount of data between platforms. Finally, if load processing runs on a different platform from the target system or where transformation took place, data has to be moved again. There are many techniques available for data movement. Whichever technique you employ for data movement, this is one of the biggest challenges in a Data Warehousing environment.

Because most of the source data comes from the OLTP environment, we strongly suggest hosting your Data Warehouse on the same platform and running the ETL process on that platform. The System z platform has sufficient capabilities to support this kind of OLTP, ETL and Data Warehousing workloads together in one or more logical environments.

ETL processing can be quite complex. There are many operational challenges, such as meeting the service level agreements (SLAs), scaling to the data volumes, and completing within the available batch window. Organizations can develop in-house applications for ETL processing, but IBM offers a rich set of tools as well for System z. These tools support most of the data organizations you will find today. In addition, they support complex data transformations and cleansing requirements. The tools exploit DB2 LOAD or parallel INSERT capability to load a large volume of data efficiently.

In the next section, we discuss ways to improve batch agility, particularly for the extraction phase of the ETL process.

12.2.1 Overcoming operational inefficiency with ETL

The design of the batch process for ETL is a very important matter and if not handled appropriately it could cause significant operational problems over time. In mainframe environments we often see huge amounts of data being handled by ETL-like batch jobs. The volume can exceed anyone’s expectations. The data growth curve is only getting steeper for many of the mainframe environments around the world.

You might consider a sort of “scaling strategy” to break up the batch jobs so that your are less dependent on one single long-running batch job and can move gradually towards a continuous transformation and update process. This way of thinking can give an organization a better chance of meeting today’s business demand of highly accurate and available data.

Loading the data into the database is in general the most time consuming part of an ETL process and this affects the overall time-performance-ratio of the batch job. Mainframe environments can reduce the elapse time by using bulk processing of flat files instead of doing standard SQL database processing that includes transactional integrity.

First, let us explore some solutions that can help us with extracting the data in a faster and more scalable way, so it does not lock our master data that is used for OLTP work.

Data extraction improvement options

There are several improvements we can implement to establish a greater operational efficiency. The first approach we would like to discuss, are the potential improvements we can make to the data extraction stage of ETL. Figure 12-3 shows a technique that can be used to extract data, using tooling to read the data from the log and place it in a transaction queue for further parallel processing. This is just one example of a technique to move data from a source to a target environment, but it could give a mainframe environment a huge batch agility advantage.

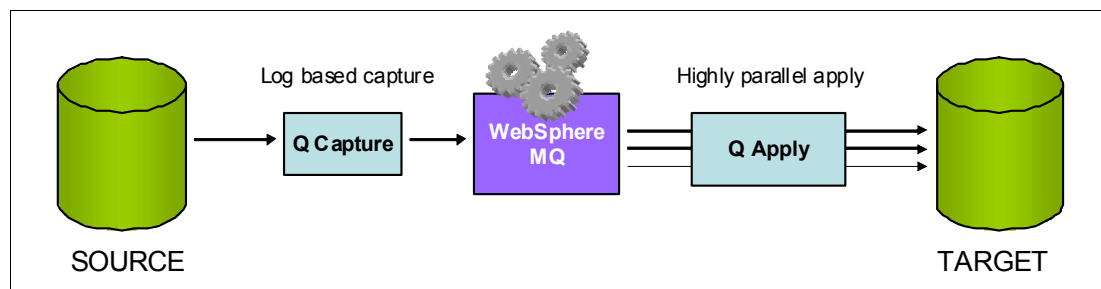


Figure 12-3 Data extraction tools - Example

Organizations that face pressure from business users that require extended online application availability, and at the same time request that more jobs and reports are processed, can look

at several IBM InfoSphere solutions for their problems. InfoSphere solutions can help solve the following issues:

- ▶ Relative simple and straightforward continuous replications of tables
- ▶ Help to reduce nightly batch window for data integration
- ▶ File locking during data extraction, which prevents application availability to end-users
- ▶ A scalable data extraction architecture

Depending on the source data, you might need a different software product, which are low impact solutions that supports the following modes of replication:

Continuous mirroring Apply data changes at the target as it is generated at the source
Periodic mirroring Apply net changes on a scheduled basis
Refresh Apply a snapshot version of source system

The IBM InfoSphere Information Server suite of products provides a framework for services around data transformation, data quality assurance, metadata management, SOA services, and other operations. IBM InfoSphere Information Server provides the following key integration functions:

Understand data This function is about analyzing data to determine the meaning and relationships of information. Web-based tools can be used to define and annotate fields of business data. Monitoring functions can be used to create reports over time. A meta model foundation (and the IBM Metadata Server) helps in managing changes in the transactional (OLTP) data model and their implication in the warehouse model and derived reports.

Cleanse data We mentioned cleansing before as a key function. In a Data Warehouse environment, multiple data sources maintained by different applications are to be integrated. Consequently, the data values in the transactional system can be stored in different formats and independent systems are likely to duplicate data. The QualityStage® component in Information Server supports consolidation, validation, and standardizing of data from these multiple data sources and thereby helps to build a consistent, accurate, and comprehensive warehouse model.

Transform data Transforming data in the context of an ETL process is a major requirement for a comprehensive Data Warehouse solution. The DataStage component in Information Server provides a variety of connectors and transformation functions for this purpose. High speed join and sort operations as well as the parallel engine help in implementing high-volume and complex data transformation and movement. A comprehensive set of tools are offered to monitor, manage, and schedule ETL jobs.

Deliver data In addition to the components mentioned in this section, Information Server comes with many ready-to-use native connectors to various data sources, both located on distributed or mainframe systems.

Data extraction batch job procedures can be improved easily using an InfoSphere solution to deliver data. Figure 12-4 displays the IBM InfoSphere solutions that available to extract data from a source as a first step in the ETL process.

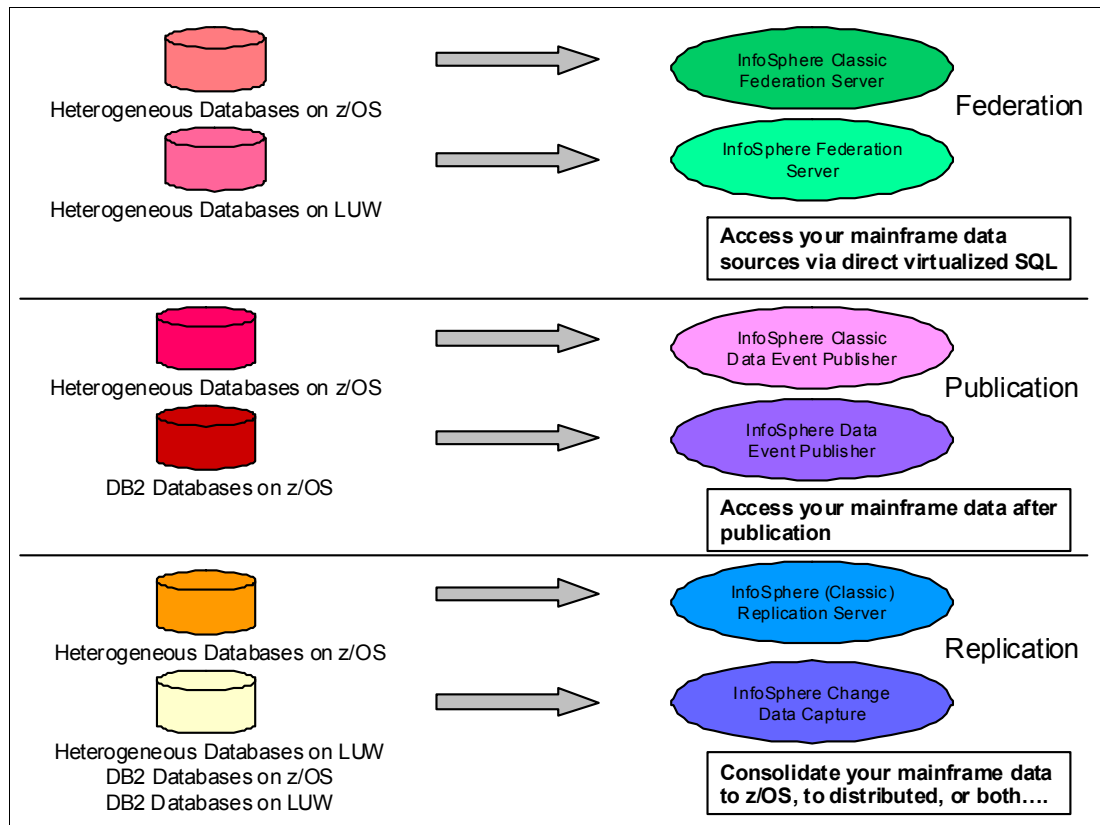


Figure 12-4 IBM InfoSphere data capture solutions

Depending on the source and target type, you might need a different product solution from one of the following categories:

- ▶ Federation
- ▶ Publication
- ▶ Replication

Federation of data

A Data Warehouse can collect, consolidate, and aggregate data from various sources and various source types. Not all sources are relational data sources, such as DB2 for z/OS is. Some data might reside on System z in the following sources:

- ▶ IMS
- ▶ VSAM data sets
- ▶ ADABAS
- ▶ IDMS
- ▶ DATACOM
- ▶ Flat files

This data (or at least aggregates of it) needs to be integrated into the Data Warehouse so that all information becomes available in a single place.

InfoSphere Classic Federation can access this data in a way that makes it unnecessary to set up a specialized extraction process for each of the different data sources (such as hierarchical or relational databases).

The InfoSphere Classic Federation Data Server is installed where the data sources reside. A connector is configured and started for each data source that needs to be accessed. These connectors are used by the data server to access the source data from IMS, VSAM, ADABAS, or other supported sources. The data server maps the various source data structures to a relational structure. So all data accessed through the InfoSphere Classic Federation Data Server looks like one relational database, even if the source is structured hierarchically.

Mapping between source and target definitions is done with the Classic Data Architect, which is an Eclipse-based workstation tool. It allows importing IMS DBD or COBOL copybooks to obtain the structure definitions of source data. The user can select the information to include in a target table, which is simulated as a relational structure by the data server. No further configuration or coding is necessary.

An ETL server can access this data through JDBC or ODBC to move it to the Data Warehouse in the form of staging tables or aggregates. The ETL Server and the InfoSphere Classic Federation Data Server are primarily used for either an initial load or a full reload of the Data Warehouse. Data in existing data sources can be updated as well, but this is usually not required within a Data Warehouse environment.

Publication of data

After the data is loaded into the Data Warehouse, it usually must be updated incrementally. You do not need to replace the entire content. InfoSphere Data Event Publisher and InfoSphere Classic Data Event Publisher, respectively, are used to detect the changes in DB2 sources and existing data sources and to provide the information about the changes to the ETL server.

The Classic Data Event Publisher and the Classic Federation have much code in common. If you installed and configured Classic Federation before, you can extend this configuration to allow event publishing. You must map existing data structures to relational tables by using Classic Data Architect. You can even configure the server so that this information is used for federation and event publishing at the same time, with just one running data server.

Instead of using a connector to directly access the existing data sources, a change capture agent is used to detect all manipulations to the data source. In some cases, this is a logger exit (such as IMS, for example). In other cases it is a log reader. The *Change Capture Agent* sends the change information to the correlation service, which maps the existing data structure to a relational structure that is designed with Classic Data Architect. Distribution Service and Publisher are used to send the information about the changes through MQ to the DataStage data integration server. The DataStage server is able to read the MQ messages directly from the queue, just as they are transmitted. These events can then be stacked up in staging tables and applied to the data warehouse in a composite update that is run in a batch window.

Replication of data

Data replication is all about moving data between two systems that are the same, normally used to keep data in a primary and secondary system consistent for disaster recovery or high availability or to offload data to a secondary system for workload balancing, data distribution or offloading queries.

IBM offers two types of solutions to replicate data:

- ▶ InfoSphere Change Data Capture
- ▶ InfoSphere Replication Server

InfoSphere Change Data Capture uses log-based change data capture technology to provide scalable, high performance and heterogeneous data integration without impacting source systems. IBM InfoSphere Change Data Capture is a high-performance, low latency, real-time data integration solution that enables customers to easily sense and respond to relevant business data changes throughout the enterprise. These solutions provide the following functions:

- ▶ Real-time data integration without impacting system performance
- ▶ Transactional integrity
- ▶ A full range of supported platforms and databases to work within existing environment
- ▶ Integration with Information Server to provide a single solution for real-time data integration

InfoSphere Replication Server takes advantage of the WebSphere MQ strength, to guarantee on-time delivery of data, and to queue up work. InfoSphere Change Data Capture however delivers two replication models, queue-based and SQL-based, in one solution, providing asynchronous log-based replication that maximizes flexibility and function.

The main difference between InfoSphere Change Data Capture and InfoSphere Classic Replication Server, is that InfoSphere Change Data Capture on the mainframe only supports DB2 for z/OS. And Classic Replication Server does support other heterogeneous databases on z/OS. InfoSphere Change Data Capture was acquired by IBM from DataMirror®. InfoSphere Change Data Capture has proved to be very scalable with high performance, with its support for parallel stage processing.

Note: Improving the extraction of source data extends the ETL investment, because the low impact/latency benefit combined with parallel load capabilities allows organizations to process more data outside of traditional batch windows.

InfoSphere Change Data Capture for batch window reduction

Figure 12-5 displays a simple ETL process where file locking for data extraction is eliminated by implementing an InfoSphere solution that provides continuous application availability to users even while changed data extraction is occurring.

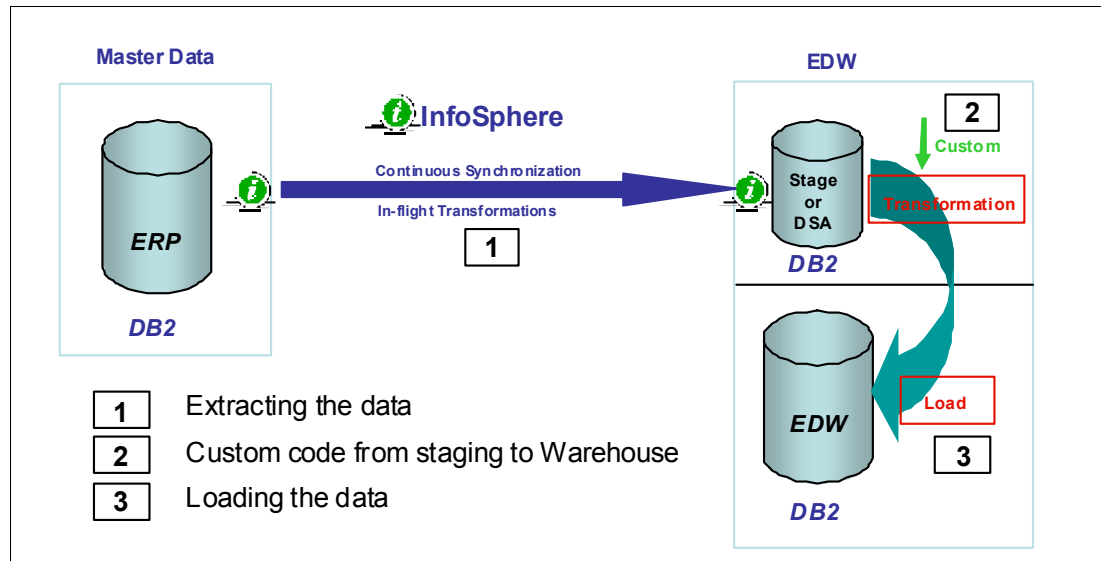


Figure 12-5 ETL example

You can implement some in-flight transformation capabilities that simplify or eliminate additional batch data extraction jobs. Particularly, InfoSphere Change Data Capture provides functionality to do filtering, table mapping, and data translations. It also provides the option to implement a User Exit to execute custom business logic, which can be implemented in C/C++, Java or stored procedures.

Depending on further steps in the ETL process, by implementing more flexible data extraction capabilities, it can provide users with the ability to access data throughout the day for reporting and trends analysis so they can make informed business decisions and increase revenue opportunities. In the next section, we discuss ways to improve the transformation and load stages to accelerate the ETL process.

Note: The idea behind all this is to extract the master data without impacting the OLTP work and by putting the data into a Data Storage Area (DSA) or staging tables for further ETL processing. This will contribute to the agility of the batch jobs and it can make a significant contribution in reducing the elapse time of the ETL batch jobs.

InfoSphere Change Data Capture is probably the best solution if you want a high performance, scalable, log-based data replication for single database environments, because it has a low impact on source or production systems. You can use it to provide a solid data staging environment that can be used for further ETL processing to build a Data Warehouse.

As mentioned previously, you can implement InfoSphere Change Data Capture using one of the following mechanisms:

- ▶ SQL replication, as shown in Figure 12-6, works as follows:
 - Capture program reads changed data from the DB2 log.
 - Changed data is staged in DB2 tables (CD tables).
 - Changes pulled (or pushed) to target tables by Apply program.
 - All done through DB2 SQL and DB2 client-server infrastructure.

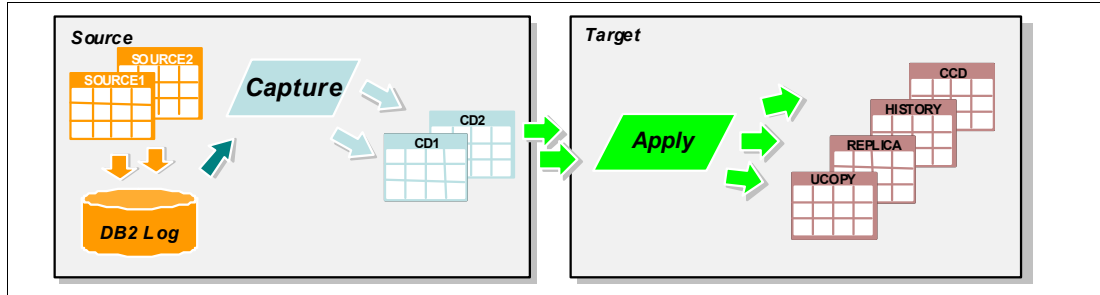


Figure 12-6 SQL replication with InfoSphere Change Data Capture

- ▶ Q replication, as shown in Figure 12-7, works as follows:
 - Capture program reads changed data from the DB2 log.
 - Data is put directly into WebSphere MQ queues, without staging.
 - WebSphere MQ delivers data to system where Apply program runs.
 - Apply program pulls data from queues and applies to target tables.

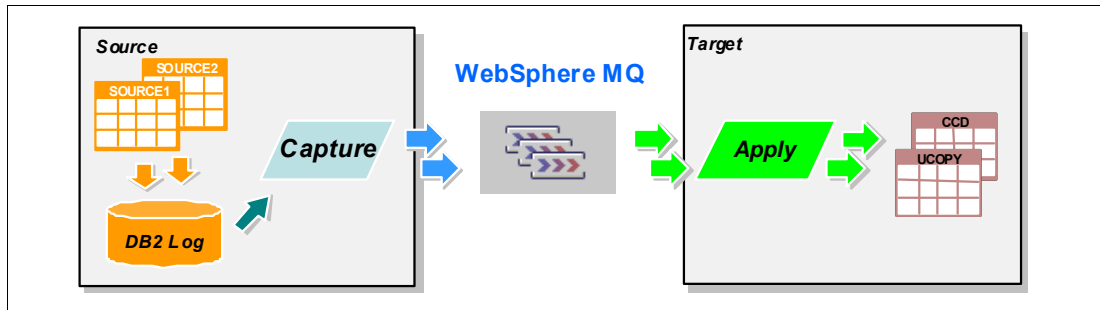


Figure 12-7 Q replication with InfoSphere Change Data Capture

With the solution shown in Figure 12-7, you take advantage of WebSphere MQ strengths, with guaranteed one-time delivery of data and SSL security options for encryption, digital signatures, and so forth. This solution also provides a secure method to move data between companies and firewalls. In addition, an outage at one site does not prevent progress on the other. For example, the capture and MQ continue even when target system down. You can also queue work for the target environment.

- Q replication also includes the following advantages:
- Transactions are rebuilt in memory.
 - Only committed transactions are put on queues.
 - Each transaction is a separate MQ message.
 - This solution allows for highly parallel queue apply, as shown in Figure 12-8.

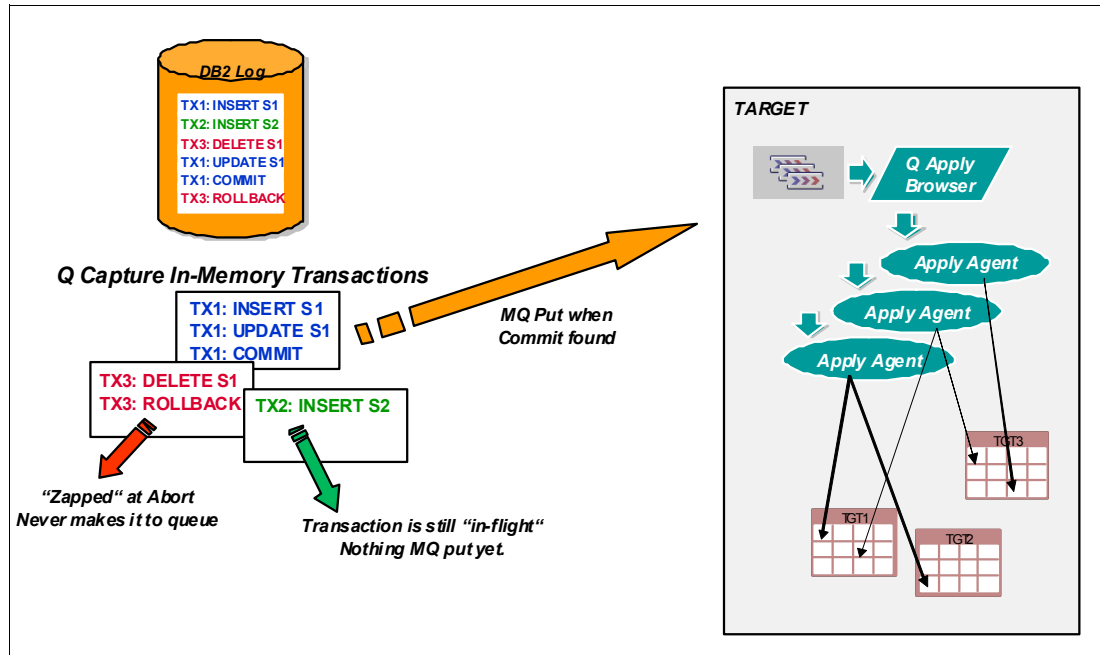


Figure 12-8 Parallel Q apply

12.2.2 ETL Accelerator with DataStage

Before taking a single batch job process apart, it is important that you have a good understanding of all the dependencies your job might actually have. The ETL process design is in many times procedural by nature and can exist of hundreds of operations. To provide the business with agile batch capabilities, we might need to redesign the ETL process to become highly efficient, maintainable and off-course scalable.

IBM InfoSphere DataStage product is part of the Information Server solution and it provides functionality for the transformation and movement of data. In addition, it allows for codeless and visual design of data flows. InfoSphere DataStage also includes many built-in transformation functions.

One of the main functionalities of InfoSphere DataStage is to perform transformations in batch or real time. The data source can be anything, from indexed files, to relational databases and message queues. Data transformation and movement is the process by which source data is selected, converted, and mapped to the format required by targeted systems. The process manipulates data to bring it into compliance with business, domain, and integrity rules and with other data in the target environment.

InfoSphere DataStage manages the data that is received on a periodic or scheduled basis. For mainframe sites this could imply that we have to process massive data volumes. Using the parallel processing capabilities of multiprocessor hardware platforms, InfoSphere DataStage can scale to satisfy the demands of ever-growing data volumes, stringent real-time requirements, and ever shrinking batch windows.

Note: InfoSphere DataStage is particularly useful when you are dealing with many transformations and significant differences between the source and target data model.

InfoSphere DataStage contains more than 300 pre-built functions that can be exploited to do data validations, or all sorts of business derivations and calculations on data. The data flows

are composed using the Information Server Client Package interface, with simple point-and-click techniques you can draw a schema that represents your ETL process requirements.

An InfoSphere DataStage job consists of individual stages where every stage is a step in the process, for example extract data from source or to transform it. Parallel jobs are supported that run inside the InfoSphere DataStage engine and have the capability to support parallel processing as well.

The runtime engine for InfoSphere DataStage is primarily available for Linux for System z and distributed platforms. There are however two older z/OS versions available, called InfoSphere DataStage MVS Edition and InfoSphere DataStage Enterprise for z/OS. At the time of writing this book, both versions are still available. It is good to know however that these z/OS versions are still available for product order, but further development of these products is uncertain. The way forward is InfoSphere DataStage for Linux for System z.

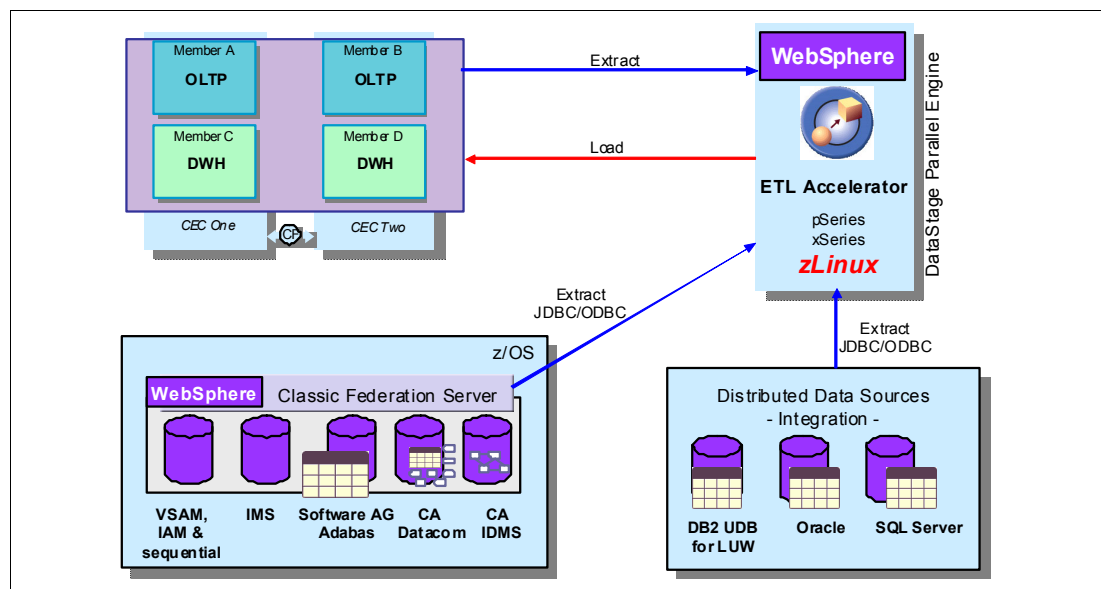


Figure 12-9 Initial load of the Data Warehouse

Figure 12-9 displays the following ETL architecture for building the initial Data Warehouse.

- ▶ Extract, Transform, and Load is done using an ETL Accelerator, which in its current implementation is represented by InfoSphere DataStage (IBM Information Server) on a secondary system. This could be Linux for System z.
- ▶ The data is extracted from the OLTP information of the data sharing group, transformed by InfoSphere DataStage and then loaded into the Data Warehouse tables again.
- ▶ Distributed data sources are directly integrated into InfoSphere DataStage.
- ▶ In this example, the existing data sources are integrated through the WebSphere Classic Federation Serve so that data is extracted from InfoSphere DataStage directly out of the data sources such as IMS and VSAM.

Note: Running InfoSphere DataStage on Linux for System z enables us to use HiperSockets and z/OS Batch Pipes, which can significantly increase the speed of loading the data into the Data Warehouse.

Figure 12-10 shows the use of the BatchPipes® utility, which you can use to connect products on Linux on System z with products on z/OS.

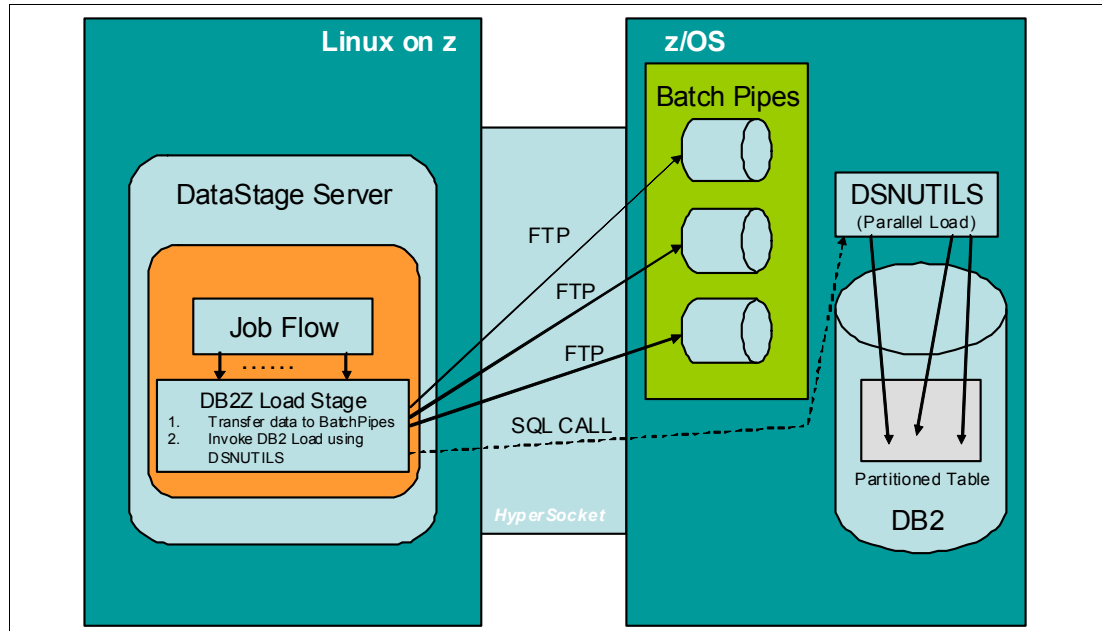


Figure 12-10 DB2ZLoad with BatchPipes

InfoSphere DataStage integration jobs extract and transform data based on defined data sources. The jobs eventually write the result to a data set in the file system or insert records into a database in DB2 for z/OS by using the LOAD utility.

The LOAD utility can only read from a data set, until the writing to the data set is completed. So the ETL job cannot write while the LOAD job processes records.

To offer a way for both “writer” and “reader” jobs to run concurrently, you can use BatchPipes functionality. The use of batch pipes can drastically shorten the Load phase, when used in combination with System z hardware HyperSockets. Running in a test environment demonstrated a load time reduction as high as 40%.

For more information about the BatchPipes utility, see the following resources:

- ▶ Chapter 16, “Increasing concurrency by exploiting BatchPipes” on page 309
- ▶ *Enterprise Data Warehousing with DB2 9 for z/OS*, SG24-7637

After the Data Warehouse is loaded, only incremental updates are performed. The changes on the original data sources (z/OS and distributed) are captured by a publication product, such as WebSphere Data Event Publisher, or a replication product, such as InfoSphere Change Data Capture, and sent through WebSphere MQ to the DataStage ETL Accelerator.

Special InfoSphere DataStage procedural steps, which the product calls *stages*, take the change information from WebSphere MQ, as shown in Figure 12-11, and updates or inserts are performed on the Data Warehouse data.

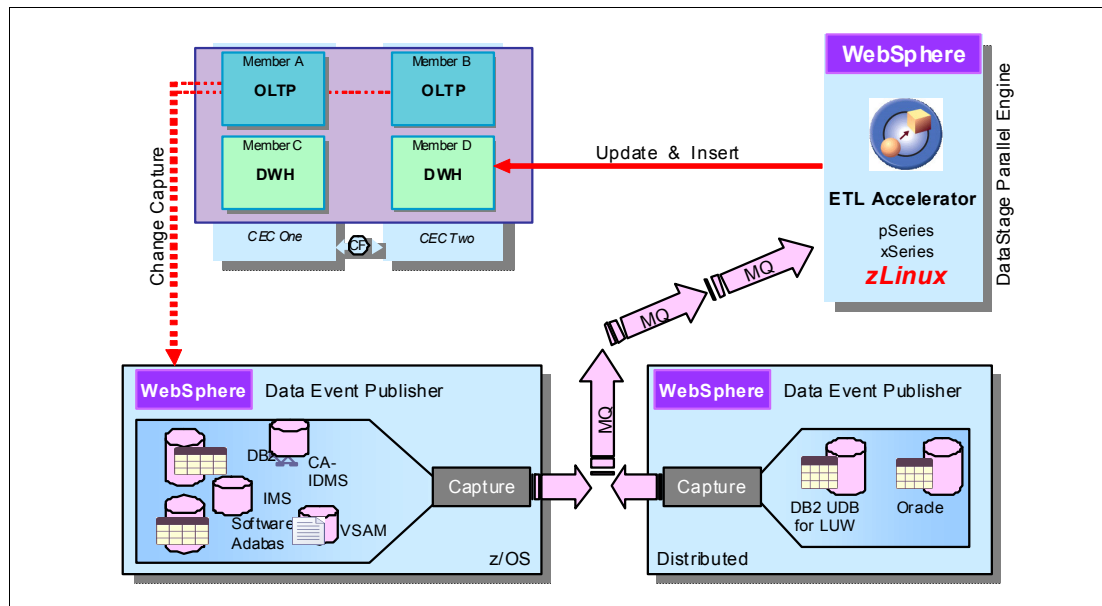


Figure 12-11 Incrementally updated Data Warehouse

InfoSphere DataStage Designer

InfoSphere DataStage Designer is a GUI application that can be used to design InfoSphere DataStage applications, which are called *jobs*. You define sources, business rules, and a target in a job, and you can do all the ETL steps using these jobs. The GUI also supports testing, debugging, and compiling of the job. Figure 12-12 shows an example of an InfoSphere DataStage job.

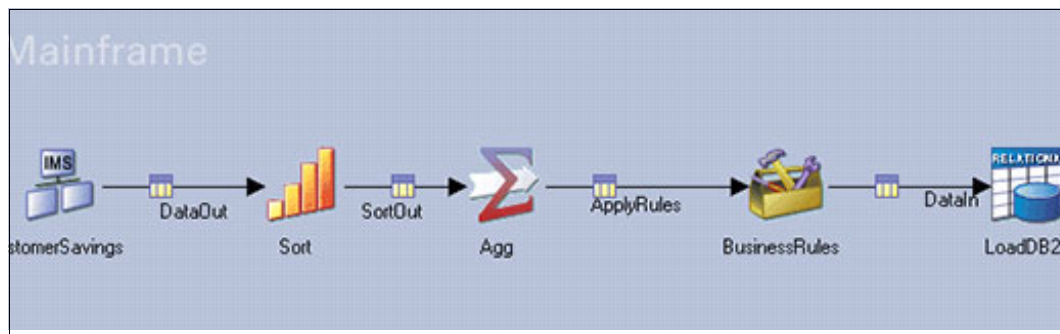


Figure 12-12 InfoSphere DataStage job

A single InfoSphere DataStage job consists of one or more *stages*. Each stage represents a processing step that is required for this job. Stages are connected using links, which represent the flow of data between different stages. The product comes with a lot of stages that have built-in functionality. Special stages are also available that you must install separately. For example, InfoSphere DataStage Transformation Extender is orderable as an add-on product. You can contain a group of stages in a *container* to allow for modularization and a more simple job design.

Batch jobs

An IBM InfoSphere DataStage job consists of individual stages that are linked together, which describe the flow of data from a data source to a data target. A stage usually has at least one data input or one data output. However, some stages can accept more than one data input, and output to more than one stage. Each stage has a set of predefined and editable properties that tell it how to perform or process data. Properties might include the file name for the sequential file stage, the columns to sort, the transformations to perform, and the database table name for the DB2 stage. These properties are viewed or edited using stage editors. Stages which have an iconic representation in the Designer are added to a job and linked together using the Designer.

A parallel InfoSphere DataStage job incorporates two basic types of parallel processing, *pipeline* and *partitioning*. Both of these methods are used at run time by the Information Server engine to execute the job. This execution is done automatically. To the InfoSphere DataStage developer, this job appears the same on the Designer.

Pipeline parallelism All stages run concurrently, even in a single-node configuration. As data is read from the source, it is passed to the Sort and Aggregate stage (see Figure 12-12 on page 203) for transformation, where it is then passed to the DB2 target. Instead of waiting for all source data to be read, as soon as the source data stream starts to produce rows, these rows are passed to the subsequent stages.

All stages in our example operate simultaneously regardless of the degree of parallelism of the configuration file. The Information Server Engine always executes jobs with pipeline parallelism. If you ran the example job on a system with multiple processors, the stage reading would start on one processor and start filling a pipeline with the data it read. The transformer stage starts running as soon as there was data in the pipeline, processes it, and fills another pipeline. The stage writing the transformed data to the target database would similarly start writing as soon as there was data available. Thus, all three stages are operating simultaneously.

Note: You do not need multiple processors to run in parallel. A single processor is capable of running multiple concurrent processes. Unfortunately, with InfoSphere DataStage Enterprise for z/OS this would result in multiple Address Spaces being initiated. This is an undesirable side effect, which leads to higher CPU consumption and slower performance.

Partition parallelism When large volumes of data are involved, you can use the power of parallel processing to your best advantage by partitioning the data into a number of separate sets, with each partition being handled by a separate instance of the job stages.

Partition parallelism is accomplished at run time, instead of a manual process that would be required by traditional systems. The DataStage developer only needs to specify the algorithm to partition the data, not the degree of parallelism or where the job will execute. Using partition parallelism the same job would effectively be run simultaneously by several processors, each handling a separate subset of the total data. At the end of the job the data partitions can be collected back together again and written to a single data source.

Note: There is also the option to combine pipeline and partition parallel processing to achieve even greater performance gains. In this scenario you would have stages processing partitioned data and filling pipelines so the next one could start on that partition before the previous one had finished.

For more information about parallel job design, see *Parallel Job Developer Guide*, LC18-9891, which demonstrates how to design and run InfoSphere DataStage parallel jobs as well as basic QualityStage functionality.

InfoSphere DataStage for Linux for System z can be an excellent runtime environment to replace complex ETL stand-alone batch jobs and to become more agile in your current IT environment. A lot of the work can be done without burdening the OLTP environment. In addition to that, with InfoSphere DataStage a more efficient ETL process can be arranged that takes advantage of out-of-box parallelism techniques.



Create agile batch by optimizing DB2 access

Batch windows are defined as “the time when batch jobs are scheduled and no online transactions are executed.” However, how do you explain Web downtime to your customers for 4 hours during peak times at night because of a batch window? Looking at continuous operations and high availability, batch windows are used less today because of business needs. Thus, batch processes must be designed to allow them to run concurrently with online transactions.

Because batch programs and online transactions often share the same data, their underlying programs must consider the specific conditions in terms of parallel processing. For example, multiple parallel processes can read the same data from the database, but if one process changes data, the data is locked until the corresponding unit of work ends. During this time, other processes that need the same data must wait.

Many batch jobs on z/OS access high volumes of DB2 data, and many times this data is the same data that is used in OLTP transactions. Running heavy batch jobs that access millions of DB2 records during the OLTP window can be a daunting task, because OLTP transactions that require sub-second response time might be confronted with locks on the data.

The waiting time due to database locks is not the only issue. Running batch and online in parallel causes a system to become more busy. If such a system receives its performance limits, your service level agreements (for example regarding response time) can no longer be guaranteed. To prevent a system from running at its limit, you need to treat the available resources as economically as possible.

In Chapter 12, “Create agile batch by optimizing the Information Management architecture” on page 189, we explained ways to extract or replicate data to a Data Warehouse so that at least batch jobs could use a Data Warehouse for queries. In certain cases, you cannot run batch programs on a data extract, and the DB2 master data needs to be accessed for queries, updates, or both.

In situations where you need to run a batch program any time, including during the OLTP window, and also run this batch program on master data, it becomes critical to implement data access in both the OLTP transactions and the batch program in the most efficient manner. Access to DB2 must be carefully designed and configured keeping in mind the following principles:

- ▶ Normally batch programs process a huge amount of data. Thereby often, depending on the business logic, some data is static and does not change during the program run (for example company address information). It is recommended to read such data only once from the database and cache it somewhere for further processing. This will prevent your system from running unnecessary round trips to the database.

In 13.1, “Data caching” on page 209, we discuss these considerations.

- ▶ SQL is a flexible and powerful tool to access databases. A non-conscious developer could theoretically write a one-line query that could hang up the performance of all your applications accessing the database. Writing smart and well-performing queries is not trivial.

In 13.2, “Optimizing data access using SQL functionality” on page 210, we provide some best practices to apply.

- ▶ In a DB2 for z/OS environment there is quite a bit of configuration and there are many ways to optimize performance. We just name a few:
 - When accessing DB2 tables with static SQL, a PLAN is created and this PLAN has several options you can configure. Making the right decisions is important.
 - Running DB2 utilities frequently keeps DB2 databases “lean” and fast.
 - Configuring DB2 system parameters can greatly influence performance and parallel processing with the database.
 - Choosing the correct checkpoint frequency and commit scope should be a key aspect in any DB2 application design.
 - When using Java, additional design and configuration decisions have to be made. The Java DataBase Connectivity (JDBC) drivers have additional configuration options.

In 13.3, “Optimizing data access using system functionality” on page 212, we discuss a variety of key considerations in optimizing DB2 access from a configuration point of view.

Note: In this book, we cannot mention all aspects and possibilities in relation to optimized DB2 data access. You can find detailed information about optimizing database access in the following resources:

- ▶ *DB2 UDB for z/OS: Design Guidelines for High Performance and Availability*, SG24-7134
- ▶ *DB2 9 Performance Monitoring and Tuning Guide*, SC18-9851
- ▶ *DB2 9 for z/OS Performance Topics*, SG24-7473

13.1 Data caching

Caching of data is one of the techniques to speed up the application and lower the database workload. After data is cached, it is usually faster to get access to that data again than retrieving it from the database again. Usually, cached data is stored in buffer storage. We can distinguish between hardware and software caching.

Data cannot stay in a cache forever, however, and a cache needs to be “maintained,” meaning that data in the cache needs to be cleaned up frequently. One reason for this might be that the available memory in the buffer is no longer sufficient and more new data has to be stored. Another reason might be that current data is no longer valid. Another process could have changed data in the database but hasn’t updated the cached data. In this case, such invalid data has to be removed from the cache.

There are various techniques available for removing data from caches.

- ▶ Least Recently Used (LRU)
The entry which has not been accessed for the longest time will be removed.
- ▶ First In First Out (FIFO)
The oldest entry will be removed.
- ▶ Least Frequently Used (LFU)
The least read entry will be removed.
- ▶ Random
A random entry will be removed.
- ▶ Climb Strategy
New data will always be stored at the lowest level. Every access of cached data will increase the level of this data. If free space is needed, data on the lowest level will be removed.

Due to their advantages caches are used in most places within the system. This applies to both the software and the hardware level. The following list contains examples of caches, which are used for faster processing, delivery or storage of data. A full list of available caches is beyond the scope of this book.

- ▶ At the hardware level, caches are used in many ways (for example CPU, hard disk controller).
- ▶ In DB2 several caches are used. For example, data that is requested by an application is also stored in DB2 buffer pools. Using dynamic statement cache can accelerate execution of dynamic SQL statements.
- ▶ WebSphere Application Server uses caches in many different areas. For example, DB2 connections are cached in a connection pool to avoid rebuilding connections every time a WebSphere transaction is scheduled. Also Enterprise Java Beans are stored in its own cache to prevent bean instantiation for each call.
- ▶ In order to accelerate transaction start processing, programs on the mainframe could be kept in a *Preload* stage. This prevents loading load modules every time an online transaction is scheduled.
- ▶ It might be a good idea to implement a self-written caching mechanism inside program.

DB2 itself uses a lot of its own caching mechanisms. However, even if the requested data is still in the buffer pool and does not need to be read from hard disk, there might still be reasons for using a separate application cache, as follows:

- ▶ As long as data is kept in an application cache, further access to DB2 for this data is not required.
- ▶ If the data, for whatever reason, does not exist in the buffer pool anymore, you would still have it in the application cache.
- ▶ If DB2 uses a bad access path for a query, further calls of this query would result in the same overhead.

In general, the sooner that a cache is used, the more beneficial it is to performance.

13.2 Optimizing data access using SQL functionality

Data, which almost never changes during program execution time, is most appropriate for caching. Other data must be read each time directly from the database, when needed. This data should always be accessed in an optimal way to speed up processing and to minimize claiming of system resources, which is especially important when running batch and online in parallel.

To read and manipulate data SQL statements are sent to the database. Thereby, SQL has many integrated functionality to efficiently access data. To get optimal performance all features of SQL should be used. You need to design an application to access data optimally.

As a suggestion, we show only a few examples of how you can optimize data access using SQL functionality. We discuss the following functions:

- ▶ Joining tables
- ▶ Using SELECT FROM INSERT/UPDATE/DELETE/MERGE
- ▶ Multi-row processing
- ▶ Explain SQL statements

13.2.1 Joining tables

Often, data is stored in different tables with dependencies between them. As an example, table DEPARTMENT contains information about departments (such as DEPTNO and LOCATION) and table EMPLOYEE contains information about employees of a company (such as EMPNO, LAST_NAME, and FIRST_NAME). Because each employee is working in a department, EMPLOYEE also contains the corresponding number of his department in column WORKDEPT.

If a report needs to list data from multiple dependent tables, it is better to fetch all data with one single SQL statement instead of accessing each table with a separate SQL statement as shown in Example 13-1.

Example 13-1 Using SQL Joins

```
-- Better use one single SQL cursor
SELECT D.*, E.*
FROM DEPARTMENT D
INNER JOIN EMPLOYEE E
    ON D.DEPTNO = E.WORKDEPT
ORDER BY D.DEPTNO, E.EMPNO;
```

```

-- Instead of using two different SQL cursor
SELECT D.*
FROM DEPARTMENT D
ORDER BY D.DEPTNO;

WHILE RowIsFound DO
  SELECT E.*
  FROM EMPLOYEE E
  WHERE E.WORKDEPT =:D_DEPTNO
  ORDER BY E.EMPNO;
END

```

13.2.2 Using SELECT FROM INSERT/UPDATE/DELETE/MERGE

By using the SQL syntax SELECT FROM INSERT/UPDATE/DELETE/MERGE, you can reduce the number of needed SQL statements, which is particularly useful in cases where tables contain columns which are automatically changed by DB2 (Trigger, Generated Always, and so forth).

Example 13-2 shows that a single SQL statement is sufficient to directly determine data, which is assigned by DB2 after an INSERT/UPDATE statement. It can, for the purpose of logging, also be relevant to receive contents of a column before the data is deleted. Without this functionality two separate SQL statements would be necessary.

Note: SELECT FROM INSERT requires DB2 for z/OS Version 8 and SELECT FROM UPDATE/DELETE/MERGE requires DB2 for z/OS Version 9.

Example 13-2 SELECT FROM INSERT/UPDATE/DELETE

```

CREATE TABLE T1 (COLa INTEGER, COLb TIMESTAMP WITH DEFAULT);

-- Retrieve current value for column "COLb" after inserting new data
SELECT COLb FROM FINAL TABLE (INSERT INTO T1 (COLa) VALUES (1));

-- Retrieve current value for column "COLb" after updating current data
SELECT COLb FROM FINAL TABLE (UPDATE T1 SET COLb = COLb + 1 year);

-- Retrieve value for column "COLb" before deleting data
SELECT COLb FROM OLD TABLE (DELETE FROM T1 WHERE COLa = 1);

```

13.2.3 Multi-row processing

Multi-row support by DB2 allows processing of multiple records with one single SQL statement. In this case, all data is transferred as one block to DB2 instead of transferring all data individually with separate SQL statements. You can use this technique, especially for remote access, to minimize network traffic and to reach a significant performance gain.

Example 13-3, instead of using many statements only one single SQL statement is necessary to insert different records in the database.

Example 13-3 Multi-row processing

```
01 T1-VARS.
    05 COLa          PIC S9(9) COMP-4 OCCURS 3 TIMES.
    05 COLb          PIC X(32)          OCCURS 3 TIMES.
    05 COLc          OCCURS 3 TIMES.
    49 COLc-LEN     PIC S9(4) COMP-4.
    49 COLc-DATA    PIC X(60).

INSERT INTO T1 (COLa, COLb, COLc)
VALUES (:HV-COLa-ARRAY, :HV-COLb-ARRAY, :HV-COLc-ARRAY :HV-IND-ARRAY)
FOR 3 ROWS
```

You can also take advantage of multi-row processing in environments like Java. As an example, using the JDBC method `addBatch` allows processing of multiple records as a block.

13.2.4 Explain SQL statements

DB2 provides a comprehensive explain facility that provides detailed information about an access plan that the DB2 Optimizer chooses for an SQL statement. This information that allows in-depth analysis of an access path is stored in separate explain tables in the database. By using tools such as Visual Explain, you can see a graphical display of a query access path.

This access path, especially in regard to performance aspects, is of crucial importance for the execution time of SQL statements. When modifying the access path, depending on the underlying data or table structure, the response time of an SQL statement could change significantly.

Now the developer can try to reach a better access path by changing the SQL statement without changing the requested result set. After every change, the developer can explain the SQL statement again and check whether the access plan has improved. With this procedure, the developer can identify the most appropriate SQL statement.

13.3 Optimizing data access using system functionality

Many functions are available on the mainframe to optimize data processing. In this section, we discuss techniques in the following categories:

- ▶ Optimizing using DB2 for z/OS functionality
- ▶ Optimizing using I/O features
- ▶ Optimizing using JDBC functionality
- ▶ Checkpoint and restart functionality

13.3.1 Optimizing using DB2 for z/OS functionality

DB2 on z/OS provides many functionalities to optimize data processing. Because of high integration between hardware and software, DB2 on z/OS exploits mainframe strengths, for example powerful I/O processing. Many different functions are available, but we discuss only a few in this book.

Note: Some of the following examples require DB2 for z/OS Version 9.

Clone DB2 tables

A *clone table* is a table which has the same structure as the base table. You can create a clone table by using the ALTER TABLE statement and setting the parameter ADD CLONE. After this procedure, two different tables with the same data structure exists in DB2.

Because these two tables are completely independent, you can, for example, load data in the clone table and process them without influencing data in the base table. By issuing a simple SQL EXCHANGE statement, you can switch between the base and the clone table. You can also repeat this command at any time.

As an example, we have to prepare data for a Data Warehouse. This data should contain information about the next business day. First, we load and prepare the data in the clone table. As mentioned, the normal online and batch process is not influenced, because they are processing data from the base table.

When the next business day starts, we switch the base and the clone table by issuing an SQL EXCHANGE statement. With this technique, our current online and batch environment could immediately process data for the next business day without any outage. Our former base table has switched to a “new” clone table and could also be used to load and prepare data for the next business day.

Using clone tables, you can reduce or even eliminate service outage caused by batch processes.

DB2 Locking and Isolation Level

Locking of data is one of the most important aspects when running programs in parallel. Locks are used by DB2 internally to ensure data consistency. Thereby, DB2 utilizes multiple kinds of locks, including transaction and LOB locks, latches, claims, and drains.

When a page or row is locked, the table, partition, or table space that contains it is also locked. In that case, the table, partition, or table space lock has one of the following intent modes:

- | | |
|------------|--------------------------------------------------------------------------------------------------------|
| S (Share) | The lock owner and any concurrent processes can read but not change the locked page or row. |
| U (Update) | The lock owner can read but not change the locked page or row. Concurrent processes can read the data. |
- U locks reduce the chance of deadlocks when the lock owner is reading a page or row to determine whether to change it, because the owner can start with the U lock and then promote the lock to an X lock to change the page or row.

X (Exclusive) The lock owner can read or change the locked page or row. A concurrent process cannot read or update the data, because he is not allowed to acquire S, U, or X locks on the page or row.

A concurrent process can access the data if the process runs with UR isolation level (see description of DB2 isolation levels below).

The following list contains the different modes for table, partition, and table space locks:

IS (Intent Share) The lock owner can read data in the table, partition, or table space but cannot change it. Concurrent processes can both read and change the data.

IX (Intent Exclusive) The lock owner and concurrent processes can read and change data in the table, partition, or table space.

S (Share) The lock owner and any concurrent processes can read but not change data in the table, partition, or table space.

U (Update) The lock owner can read but not change the locked data; however, the owner can promote the lock to an X lock and then can change the data. Processes concurrent with the U lock can acquire S locks and read the data, but no concurrent process can acquire a U lock.

SIX (Share with Intent Exclusive) The lock owner can read and change data in the table, partition, or table space. Concurrent processes can read data in the table, partition, or table space but not change it.

X (Exclusive) The lock owner can read or change data in the table, partition, or table space. A concurrent process can access the data if the process runs with UR isolation or if data in a partitioned table space is running with CS isolation and CURRENTDATA(NO).

For example, an SQL statement locates Paul Smith in a table of customer data and changes his address. The statement locks the entire table space in mode IX and the specific row that it changes in mode X.

Thus, lock compatibility is very important to understand DB2 concurrency regarding parallel processing. For example, assume one process holding an S-lock on a row and another process wanting to read the data. The second process first tries to also set an S-lock on the same row. Because these two locks are compatible (see Table 13-1), both processes could read the same data. But, if the other process would want to change this data, he first has to set an X-lock. These two locks are incompatible. Therefore, the second process has to wait until the first process releases his S-lock.

Compatibility for page and row locks is easy to define. Table 13-1 shows whether page locks of any two modes, or row locks of any two modes, are compatible (Yes) or not (No).

Table 13-1 Compatibility of page lock and row lock modes

Lock Mode	S	U	X
S	Yes	Yes	No
U	Yes	No	No
X	No	No	No

Compatibility for table space locks is slightly more complex. Table 13-2 shows whether table space locks of any two modes are compatible.

Table 13-2 Compatibility of table and table space (or partition) lock modes

Lock Mode	IS	IX	S	U	SIX	X
IS	Yes	Yes	Yes	Yes	Yes	No
IX	Yes	No	Yes	Yes	No	No
S	Yes	No	Yes	Yes	No	No
U	Yes	No	Yes	No	No	No
SIX	Yes	No	No	No	No	No
X	No	No	No	No	No	No

The duration of a lock is the length of time the lock is held. It varies according to when the lock is acquired and when it is released. For maximum concurrency, locks on a small amount of data held for a short duration are better than locks on a large amount of data held for a long duration. However, acquiring a lock requires processor time, and holding a lock requires storage; thus, acquiring and holding one table space lock is more economical than acquiring and holding many page locks. Consider that trade-off to meet your performance and concurrency objectives.

One important parameter which influences duration of locks is the DB2 *Isolation Level* for an SQL statement. The following list shows the different DB2 Isolation Level options.

Repeatable Read (RR) A row or page lock is held for all accessed rows, qualifying or not, at least until the next commit point. If the application process returns to the same page and reads the same row again, another application cannot have changed the rows nor have inserted any new qualifying rows.

Read Stability (RS) A row or page lock is held for rows or pages that are returned to an application at least until the next commit point. If the application process returns to the same page and reads the same row again, another application cannot have changed the rows, although additional qualifying rows might have been inserted by another application process.

Cursor Stability (CS) A row or page lock is held only long enough to allow the cursor to move to another row or page. For data that satisfies the search condition of the application, the lock is held until the application locks the next row or page. For data that does not satisfy the search condition, the lock is immediately released.

Uncommitted Read (UR) The application acquires no page or row locks and can run concurrently with most other operations. But the application is in danger of reading data that was changed by another operation but not yet committed.

Isolation level RS has the highest lock behavior and, therefore, the lowest possible concurrency. Alternatively, isolation level UR has the lowest lock behavior and the highest possible concurrency (but with danger of reading uncommitted data).

Tip: A good compromise to achieve good concurrency without risk of reading uncommitted data is to use isolation level CS. Remember, however, some application might need a higher isolation level. Based on this information, you need to decide which isolation level to use for your application.

You can set an Isolation Level for your SQL statements in different ways:

► SQL statement level

At the end of your statement, use WITH UR/CS/RS/RR to explicit set an Isolation Level for this single statement. This Isolation Level is used even if the package which contains this statement is bound with a different Isolation Level.

► DB2 package/plan level

When using traditional languages (such as COBOL and PL/I), you can use option ISOLATION(UR/CS/RS/RR) when running the BIND or REBIND command for the corresponding package or plan.

► Java native application

When running a Java native application you can choose the Isolation Level by using the method `setTransactionIsolation` of the current connection object. The following list shows the JDBC and the corresponding DB2 Isolation Level.

– <code>Transaction_Serializable</code>	DB2 isolation level RR
– <code>Transaction_Repeatable_Read</code>	DB2 isolation level RS
– <code>Transaction_Read_Committed</code>	DB2 isolation level CS
– <code>Transaction_Read_Uncommitted</code>	DB2 isolation level UR

Attention: JDBC level `Transaction_Repeatable_Read` corresponds with DB2 level Read Stability, whereas DB2 level Repeatable Read corresponds with JDBC level `Transaction_Serializable`.

► J2EE or Java EE application

Setting an Isolation Level in J2EE or Java EE application environments depends on your current environment (J2EE or Java EE application, application server provider). In this book, we cannot describe all variants. Therefore, to set an Isolation Level, look into your application environment documentation.

Tip: JDBC level `Transaction_Repeatable_Read`, which corresponds to DB2 level RS, is the default Isolation Level for a WebSphere application. To change this default, you can use custom property `webSphereDefaultIsolationLevel` at the data source configuration.

Lock avoidance

Under certain circumstances, if DB2 can determine that the data it is reading has already been committed, it can avoid taking the lock altogether. Lock avoidance will increase concurrency, decrease the lock, and unlock activity and associated CPU resource consumption.

Therefore, effective lock avoidance is very important in all environments. The number of unlock requests per commit is a good indicator of the effectiveness of lock avoidance. If perfect lock avoidance is achieved, a transaction only takes locks on the resources that it wants to modify, and releases all of them in a single unlock request at commit time.

If the number of unlock requests per commit is greater than 1 (and it always is in the real world), it indicates that, as the rows were being processed, the transaction had to acquire and release some S-locks. The general rule-of-thumb is that the number of unlock requests per commit should be less than 5.

Fast table append

When creating a DB2 table, you can also define a clustering index. If present, DB2 tries to save the data in the same order as defined by the clustering index. Therefore, if new data is inserted or loaded into the table, DB2 first has to search for the right place, which be an expensive, time-consuming operation.

With DB2 for z/OS V9 it is now possible to force DB2 to save new data always at the end of the table. Especially when processing a huge amount of data (INSERT or LOAD), this could speed up your processing.

To activate or deactivate this functionality, you can use the SQL ALTER TABLE statement to set the APPEND YES/NO options. You can also set this option directly when creating the table.

Index on expression

Indexes are often used to speed up SQL processing. However, with DB2 for z/OS V8 an index cannot be used when running an SQL statement with expressions, as shown in Example 13-4.

Example 13-4 Select all data for year 2003

```
SELECT * FROM TABLE T1 WHERE YEAR( DATE_COLUMN ) = 2003
```

With DB2 for z/OS V9, you can now define an index to speed up SQL processing when using expressions, as shown in Example 13-5.

Example 13-5 Create Index on expression

```
CREATE INDEX T1.YEAR_2003 ON T1  
(YEAR( DATE_COLUMN ) ASC)
```

Online REBUILD index

If no adequate index exists, it can take a long time for DB2 to identify all data for a query. In this case, DB2 has to scan the whole table to get all data requested by the query. Often, to speed up this query another index has to be created. If a lot of data is stored in the table, you need to first create the new index with the DEFER YES option.

By using this option, the index is defined only in DB2, but it is not physically created. Afterwards, run the REBUILD INDEX utility to create index data and make this new index available for SQL processing. With DB2 V9, you can now run REBUILD INDEX during online and batch processing.

Detecting unnecessary indexes

As mentioned previously, you can use an index to speed up SQL processing. Alternatively, every index must also be maintained by DB2. However, what about an index, which was created for a specific SQL statement and this statement is not used anymore?

Because a not used index is an avoidable overhead for heavy batch inserts or updates, you need to remove this index. This process is made easier with DB2 V9 and Real Time Statistics (RTS) using the LASTUSED field in the SYSIBM.SYSINDEXSPACESTATS table.

Buffer pool size

The size of a buffer pool plays a crucial role in the performance of queries. After read data is cached in memory the first time, it can be processed faster a second time. When the data is still in the buffer pool, it does not need to be read from the hard disk again. In this case, expensive I/O processing is not necessary.

Often, in the case of batch processing lots of data is read from or written to DB2. Thereby, a too small buffer pool can quickly decrease performance of a running batch program. With DB2 V9 the maximum size of a buffer pool has increased, so that it is now possible to create very large buffer pools.

By the way, the underlying DB2 page size is also relevant to classify the maximum size of a buffer pool. For example, if you increase your DB2 page size from 4 KB to 8 KB and do not increase your buffer pool size, then you are only able to cache half as many pages in the buffer pool. This can have an impact on the buffer pool hit ratio. If you do plan to use larger DB2 page sizes, we recommend that you also review your DB2 buffer pool sizes. Always ensure you have enough real storage available to back any increase in buffer pools, to avoid any paging to disk.

RUNSTATS and REORG

If you send an SQL statement to the database, the DB2 internal Optimizer analyzes the SQL statement and specifies an access path. To find the best access path for your SQL statement, the Optimizer uses internal information (like indices implemented on the table, frequent values for special columns). To create this statistical information you have to run the DB2 RUNSTATS utility.

Because of the effort, DB2 does not automatically update this statistics if data is changed. But only with correct statistical information the Optimizer could choose a good access path. Therefore, it is recommended to run RUNSTATS after your data has significantly changed.

If you run RUNSTATS with the SHRLEVEL(CHANGE) option, you do not need to stop online or batch processing. When using static SQL, you also have to rebind the corresponding DB2 packages with the REBIND utility.

Further, to achieve best performance for your data access, you also should run the REORG utility after your data has significantly changed. After starting this utility, DB2 reorganizes the data in the database (for example sort all rows as defined by the clustering index). Similar to using RUNSTATS, it is not necessary to stop online or batch processing to reorganize data.

13.3.2 Optimizing using I/O features

Performance is of course one of the most important parts of every application. As batch is mostly I/O bound, optimal I/O performance is extremely relevant to achieve fast batch processing. Especially the Mainframe has always provided powerful I/O functionality.

The following examples show only some technologies and configurable options to achieve best I/O performance with DB2 for z/OS.

MIDAW

The Modified Indirect Data Address Word (MIDAW) facility is integrated in System z processors to improve FICON® performance, especially when accessing DB2 databases. This facility is a new method of gathering data into and scattering data from discontinuous storage locations during an I/O operation.

The use of MIDAWs will not cause the bits to move any faster across the FICON link, but they reduce the number of frames and sequences flowing across the link, which makes the channel more efficient.

Thereby, the most benefit occurs with Extended Format data sets that have small block sizes. Because DB2 depends on Extended Format data sets to stripe the logs, or to enable data sets to be larger than 4 GB, DB2 is a major beneficiary from the MIDAW facility.

DSNZPARMs

With DSNZPARMs, you have many different settings available to configure a DB2 system to your needs (like activate dynamic statement caching, maximum size of statement cache and much, much more). Thereby, some of these DSNZPARMs relate to storage functions:

- ▶ Data Set VSAM Control Interface (CI) size

With V7, DB2 uses only the VSAM CI size of 4 KB. If DB2 page size is 8, 16, or 32 KB, DB2 treated the page as a chain of 2, 4, or 8 CIs.

Since V8, setting the DSVCI parameter to YES allows DB2 to use VSAM CI sizes of 8, 16, and 32 KB and therefore to synchronize DB2 page size and VSAM CI size.

With DSVCI=YES, VSAM splits 32 KB CI over two blocks of 16 KB in order to span two tracks and not waste space within a single track usage (48 KB). A 16 KB block is not used every 15 tracks (CA size) because the CI cannot span a CA boundary.

- ▶ Sequential processing and disk cache

The parameter SEQCACH is used to determine whether data in the disk cache is to be used in sequential mode (BYPASS) or sequential detected mode (SEQ). Although BYPASS, the default, is still a valid option for detected mode, cache is not bypassed because the 3990 controllers were in use.

For the later disks (RVA, ESS, or DS8000®), BYPASS uses sequential detect, and SEQ uses explicit command. The differences are as follows:

- SEQ (explicit) puts tracks on the accelerated list and starts pre-staging for next I/O. Operations are done on extent boundaries or stage groups. Explicit SEQ reacts faster and can end sooner than using the detect mechanism (BYPASS).
- BYPASS (sequential detect) stages data to the end of a stage group.

The recommendation is to set SEQCACH to SEQ.

- ▶ Utilities use of disk cache

The parameter SEQPRES specifies whether (YES) or not (NO, default) DB2 utilities that scan a non-partitioning index followed by an update should allow data to remain in cache longer when reading data. If you specify YES, DB2 utility prefetch reads remain in cache longer, possibly improving performance of subsequent writes of large non-partitioned indexes.

Similar to what we discussed for sequential processing and disk cache (SEQCACH), the recommendation is to set SEQPRES to YES.

13.3.3 Optimizing using JDBC functionality

It is undisputed that in recent years more and more applications have been developed in Java, making more and more SQL statements communicate using JDBC with the database. Therefore, many features have been implemented in JDBC to optimize this communication.

The following examples show only some possibilities, to optimize data access for Java applications:

- ▶ Use batch update API (`addBatch` or `executeBatch`) for larger amount of inserted, updated, and merged rows, which can speed up processing. See Example 13-6.

Example 13-6 Using JDBC `addBatch`

```
PreparedStatement ps = con.prepareStatement("INSERT INTO TB1 (COL1) Values(?)");
for (int i=0; i<100; i++) {
    ps.setShort(1, i);
    ps.addBatch();
}
ps.executeBatch();
```

- ▶ Progressive Locator support (dynamic data format)

Based on actual size, this new support first determines the most efficient mode to return LOB/XML data. You can activate or deactivate this support by setting JCC properties:

- `progressiveStreaming` = yes/no (default is yes)
- `streamBufferSize` = value (default 1 M)

As mentioned, the method for returning LOB/XML data is based on actual size.

- For LOB/XML data size smaller than 12 KB, the data is inlined similar to varchar data.
- For LOB/XML data size between 12 KB and `streamBufferSize`, the data is chained to the query result.
- For LOB/XML data size between `streamBufferSize` and 2 GB a large object locator is returned.

- ▶ Java data types should match DB2 data types

To avoid data type cross-conversion you should use the correct Java for your DB2 data type (for example, `java.math.BigDecimal` for DB2 DECIMAL data type).

- ▶ Only select and update columns as necessary

A Java object is created for every retrieved column.

- ▶ Store numbers as numbers

Numeric data should always be stored in DB2 numeric data types. If you store such data in character data types, you get some disadvantages:

- DB2 does not check automatically whether this character data only contain numbers.
- SQL numeric functions, such as `AVG`, cannot be used directly.
- Because Java applications always are executed in Unicode, DB2 character data which is stored in EBCDIC/ASCII has to be converted. Because numbers are not dependent on an encoding schema, no conversion is necessary.

- ▶ Turn `autocommit` off for native Java processing

If you did not change the default behavior and run a native Java application, a “commit” is sent to the database after every single SQL statement. This commit is extremely expensive and can result in data inconsistency if the entire business transaction must be rolled back but some data changes are already committed.

You can use the JDBC method `setAutoCommit` to turn on or off `autocommit`, as shown in Example 13-7.

Example 13-7 Java turn autocommit off

```
con.setAutoCommit(false);
```

Note: You can find more information about Java and DB2 optimization in the following resources:

- ▶ *Application Programming Guide and Reference for Java*, SC18-9842
- ▶ *DB2 for z/OS and OS/390: Ready for Java*, SG24-6435
- ▶ *DB2 for z/OS and WebSphere: The Perfect Couple*, SG24-6319

13.3.4 Checkpoint and restart functionality

Normally batch programs process a huge amount of data. Furthermore, it is not unusual that data from many different systems (like DB2, MQ, IMS databases) are read or changed by one single batch process. To prevent data inconsistency, all participated systems must commit at the same time.

To achieve optimal performance for a batch job, it is recommended not to write a checkpoint after each input data. However, it is not useful to commit all changes at the end of a batch run with only one single checkpoint either. In this case, changed data is blocked for the entire period of the batch job and cannot be accessed by other processes (such as online processing). It would also generate a significant overhead for the system (for example management of locks). Also, in case of an error, the necessary rollback process could become considerably more expensive.


Therefore, it is important to write checkpoints at reasonable intervals. Further, in order to flexibly respond to current circumstances, you need the possibility to influence checkpoint settings at start or during batch processing.

You should have the opportunity:

- ▶ To write checkpoints after a specified number of processed data
- ▶ To write checkpoints after a specified period of time. Thereby, after reaching this time limit a checkpoint must be written, even if the specified number of processed data has not been reached. This prevents you from locking your data too long.
- ▶ To easily and dynamically change current settings.

Furthermore, the batch program needs a restart functionality. If an error occurs during a batch run and the program aborts, all changes to the last checkpoint are committed in the system. But because the program is not yet finished, it needs to be started again. In this case, it is important that the program continues processing immediately after the last checkpoint. It must, in any case, ensure that already processed data is not reprocessed. Therefore when restarting, batch programs often have a separate logic implemented to ensure the correct processing after the last checkpoint.

Note: As previously mentioned, it is not the scope of this book to describe all system features and SQL-based opportunities to establish an optimal database access. However, these few examples should provide a good idea of the impact achieved with optimization.



Create agile batch by implementing trigger mechanisms

Today, obtaining information about demand is critical from a business point of view. Demand also affects batch workloads. For example, consider the following scenarios:

- ▶ Creating business reports with heavy input data amounts that are produced on demand just by a mouse click.
- ▶ Executing urgent transactions immediately although they are normally scheduled for a certain batch window later.

In addition to the aspect that data must be prepared for well, as described in Chapter 12, “Create agile batch by optimizing the Information Management architecture” on page 189, we also need to consider how the batch itself can be triggered. Therefore, in this book, we provide different approaches to how batch can be submitted flexibly.

In this chapter, we discuss the following scenarios:

- ▶ How to submit jobs with Java, for example from a WebSphere application, in 14.1, “Job submission with native Java technology” on page 224
- ▶ How to trigger jobs running in a WebSphere XD Compute Grid, in 14.2, “Using WebSphere XD Compute Grid trigger mechanisms” on page 234
- ▶ How to trigger jobs on demand with the help of Tivoli Workload Scheduler, in 14.3, “Exploiting enhanced features of Tivoli Workload Scheduler for z/OS” on page 263
- ▶ How to trigger a DB2 stored procedure that is used for batch purposes, in 14.4, “Triggering a DB2 stored procedure” on page 270

Furthermore, we explain how to use WebSphere Application Server as a GUI to submit jobs in different ways.

14.1 Job submission with native Java technology

The first option to schedule z/OS batch jobs in a dynamic way is to use Java technology. Java on z/OS provides a special API that allows to submit jobs directly from of a Java application. To demonstrate how you could use this API, we created a sample J2EE application. Figure 14-1 shows how this application works. A simple servlet is acting as a Web interface to the user.

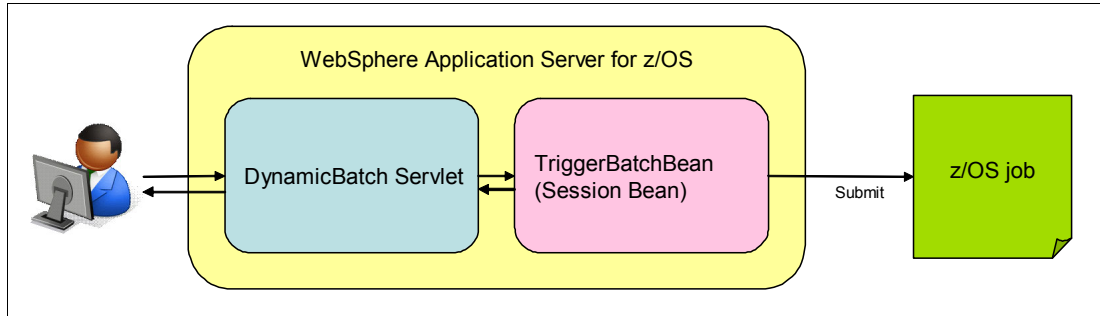


Figure 14-1 Dynamic batch sample application

Figure 14-2 shows how this servlet looks from a user perspective. It consists of a form with two parameter input fields as well as a Submit and Cancel button. Because this application is intended as a template for further chapters, it is a modular, which means that we use the parameters in other chapters for different purposes.

The screenshot shows a web browser window with the title 'Welcome to the z/OS batch submitter application'. Below the title, there is a message: 'Please enter all required parameters.' This is followed by two input fields labeled 'Parameter 1:' and 'Parameter 2:'. At the bottom of the form, there are two buttons: 'Submit' and 'Cancel'.

Figure 14-2 Sample application welcome page

By click **Submit**, the servlet calls a Session Bean called `TriggerBatchBean`, which contains the logic to call a job.

Note 1: In the template version of the application, the session bean returns only the value of the two parameters.

Note 2: We use WebSphere Application Server for z/OS Version 6.1 in our scenario to ensure compatibility with the J2EE 1.4 specification. Although we could have used Java EE 5.0, we wanted to ensure that a broad range of users can use this sample.

You can find the servlet and session bean source code in “Dynamic batch Web application” on page 445.

We also provide this application as a Rational Application Developer project. See Appendix C, “Additional material” on page 453 for more details. This project is be the base for the next sections. However, if you do not have Rational Application Developer, you can also use another J2EE development tool.

14.1.1 Developing the code

With the help of this J2EE application template, we can implement the job submission EJB based on the JZOS Toolkit `MvsJobSubmitter` class from the following Web site:

<http://www.ibm.com/servers/eserver/zseries/software/java/products/jzos/jzossamp.html>

This class is part of the IBM Java SDK for z/OS in all current releases.

Note 1: We includes instructions about how to download the entire EJB project that we discuss in this section in Appendix C, “Additional material” on page 453.

Note: Keep in mind that because we use a Web application to submit a z/OS batch job, we do not know how long a submitted job will run. Thus, we do not wait for the job output in the Web application and only submit it. Nevertheless, to get the results to the user, we propose to implement some kind of notification (for example through e-mail) that informs the user after the job of the results.

To implement the job submission, follow these steps:

1. First, switch to the J2EE Perspective in Rational Application Developer. Then, import the three projects of the application Dynamic Batch application template in Rational Application Developer by selecting **File** → **Import** → **General** → **Existing Project into Workspace** → **Next**. Choose **Select archive file**. Browse for the `DynamicBatchRADProject.zip` file of the additional material of this book, and press **Finish**.
2. Create a new class called `MvsJobSubmitter.java` and a class called `MvsJob.java` in the `com.ibm.itso.sample` package to submit z/OS batch jobs from the `TriggerBatchBean`. The Rational Application Developer Project Explorer should now look like that shown in Figure 14-3.

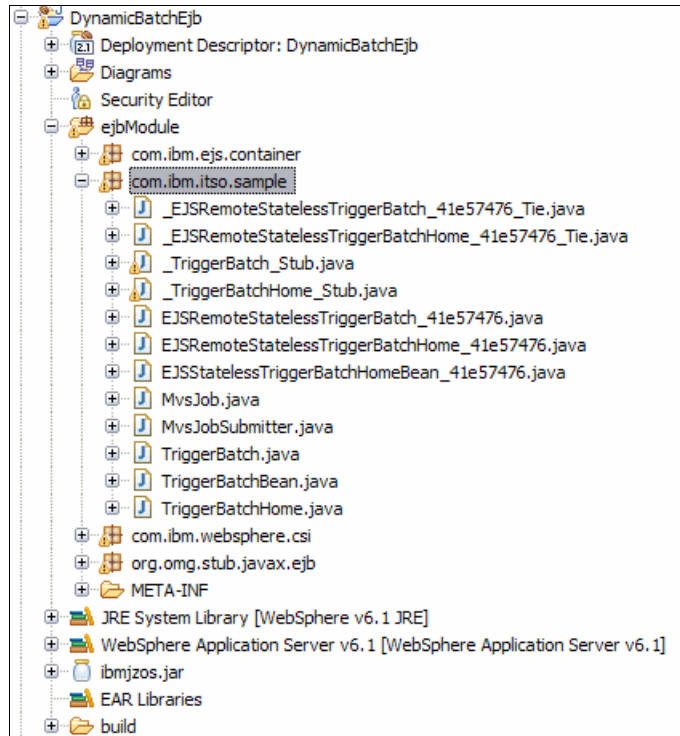


Figure 14-3 Rational Application Developer Project Explorer

3. In the `MvsJobSubmitter.java` file, insert the code shown in Example 14-1.

Example 14-1 MvsJobSubmitter.java

```

package com.ibm.itso.sample;

/*
=====
* Licensed Materials - Property of IBM
* "Restricted Materials of IBM"
* (C) Copyright IBM Corp. 2005. All Rights Reserved
*
=====
*/
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.Writer;
import java.util.Iterator;
import java.util.Properties;
import java.util.StringTokenizer;

import com.ibm.jzos.Exec;
import com.ibm.jzos.FileFactory;
import com.ibm.jzos.RcException;
import com.ibm.jzos.ZUtil;

public class MvsJobSubmitter {

    public static final String SUBMIT_JOB_CMD = "submitJob";

```

```

Exec exec;
BufferedWriter intrdrWriter;

/**
 * Submits a JCL read from a file
 * Argument can be a Unix file/path name or a "//dataset.name".
 */
public void run(String jobName) throws IOException {
    MvsJobSubmitter submitter = new MvsJobSubmitter();
    // copy the file to the InternalReaderWriter (and close that writer)
    copyFile(jobName, submitter.getInternalReaderWriter());
    submitter.submitJob();
}

/**
 * Construct an instance, which causes the "submitJob" Rexx script to
 * be started to receive our JCL.
 */
public MvsJobSubmitter() throws IOException {
    executeSubmit(); // starts the child process
}

/**
 * Answers a writer that can be used by the client to
 * send JCL to the Rexx submitJob process.
 * Clients should either flush() or close() this Writer
 * to ensure that buffered data is actually sent to the child process
 * @return Writer
 * @throws IOException
 */
public Writer getInternalReaderWriter() throws IOException {
    if (intrdrWriter == null) {
        intrdrWriter = exec.getStdinWriter();
    }
    return intrdrWriter;
}

/**
 * Answers an MvsJob object which is a simple bean holding the
 * submitted jobname and userid. The method should be called
 * once after writing jcl to getInternalReaderWriter() and closing
 * @return MvsJob if submitted successfully, otherwise an exception is thrown
 *
 * @throws IOException if there is an error communicating with the child Rexx
process
 * or if the child process had an error submitting the process.
 */
public MvsJob submitJob() throws IOException {
    if (exec == null) {
        // can only do this once!
        throw new IllegalStateException("Child process already stopped?");
    }
    MvsJob mvsjob = null;
    String line;
    String firstLine = null;
    while ((line = exec.readLine()) != null) {
        if (firstLine == null) firstLine = line;
        if (mvsjob == null) {

```

```

        StringTokenizer tok = new StringTokenizer(line);
        if (tok.countTokens() == 3) {
            if (tok.nextToken().equalsIgnoreCase("Submitted:")) {
                String jobname = tok.nextToken();
                String jobid = tok.nextToken();
                mvsjob = new MvsJob(jobname, jobid);
            }
        }
    }
}
int rc = exec.getReturnCode(); // wait for process completion
exec = null;
if (mvsjob == null) rc = 3; // somehow we got a good return code with invalid
output
if (rc != 0) {
    throw new RcException("Rexx 'submitJob' process failed: " + firstLine, rc);
}
return mvsjob;
}

/**
 * Start the "submitJob" child process.
 */
protected void executeSubmit() throws IOException {
    exec = new Exec(getSubmitCommand(), getEnvironment());
    exec.run();
}

/**
 * Return the command to be executed via Runtime.exec(). This is
 * the Rexx script 'submitJob'. By default, this script needs to be present
 * in the current PATH. However, if the System variable jzos.script.path
 * is defined, it will be used to prefix 'submitJob'.
 */
protected static String getSubmitCommand() {
    String cmdPath = System.getProperty("jzos.script.path", "");
    if (cmdPath.length() > 0 && !cmdPath.endsWith("/")) {
        cmdPath = cmdPath + "/";
    }
    return cmdPath + SUBMIT_JOB_CMD;
}

/**
 * Return the environment to use for the child process. This is
 * the current environment with _BPX_SHAREAS and _BPX_SPAWN_SCRIPT set to "YES"
 * so that the child process will execute in the same address space.
 */
protected static String[] getEnvironment() {
    Properties p = ZUtil.getEnvironment();
    p.put("_BPX_SHAREAS", "YES");
    p.put("_BPX_SPAWN_SCRIPT", "YES");
    String[] environ = new String[p.size()];
    int i=0;
    for (Iterator iter = p.keySet().iterator(); iter.hasNext();) {
        String key = (String)iter.next();
        environ[i++] = key + "=" + p.getProperty(key);
    }
    return environ;
}
}

```



```

/**
 * Copy a file given its name to a Writer.
 * After copying or on error, close both files.
 */
protected static void copyFile(String filename, Writer writer) throws IOException {

    BufferedReader rdr = null;
    try {
        rdr = FileFactory.newBufferedReader(filename);
        String line;
        while ((line = rdr.readLine()) != null) {
            writer.write(line);
            writer.write("\n");
        }

    } finally {
        writer.close();
        if (rdr != null) {
            rdr.close();
        }
    }
}
}

```

4. Next, insert in the `MvsJob.java` file the code shown in Example 14-2.

Example 14-2 MvsJob.java

```

/*
=====
 * Licensed Materials - Property of IBM
 * "Restricted Materials of IBM"
 * (C) Copyright IBM Corp. 2005. All Rights Reserved
 *
=====
*/
package com.ibm.itso.sample;

/**
 * Simple bean which holds an MVS jobname and id.
 */
public class MvsJob {

    String jobname;
    String jobid;

    public MvsJob(String name, String id) {
        this.jobname = name;
        this.jobid = id;
    }

    public String getJobid() {
        return jobid;
    }

    public String getJobname() {
        return jobname;
    }
}

```

```

public String toString() {
    return "" + jobname + "(" + jobid + ")";
}
}

```

5. Because the JZOS Toolkit libraries are not included in the Rational Application Developer Build Path, you now have to download the `ibmjzos.jar` file from the z/OS system using FTP to the root directory of the Rational Application Developer EJB project on the local workstation. This SLA file is stored in `<Java_Home>/lib.ext` on z/OS.
6. After you download the file, click the project, and press F5 to refresh the content. Now the `ibmjzos.jar` file displays in the project. Include this file by right-clicking the project, and then clicking **Build Path** → **Configure Build Path** as shown in Figure 14-4.

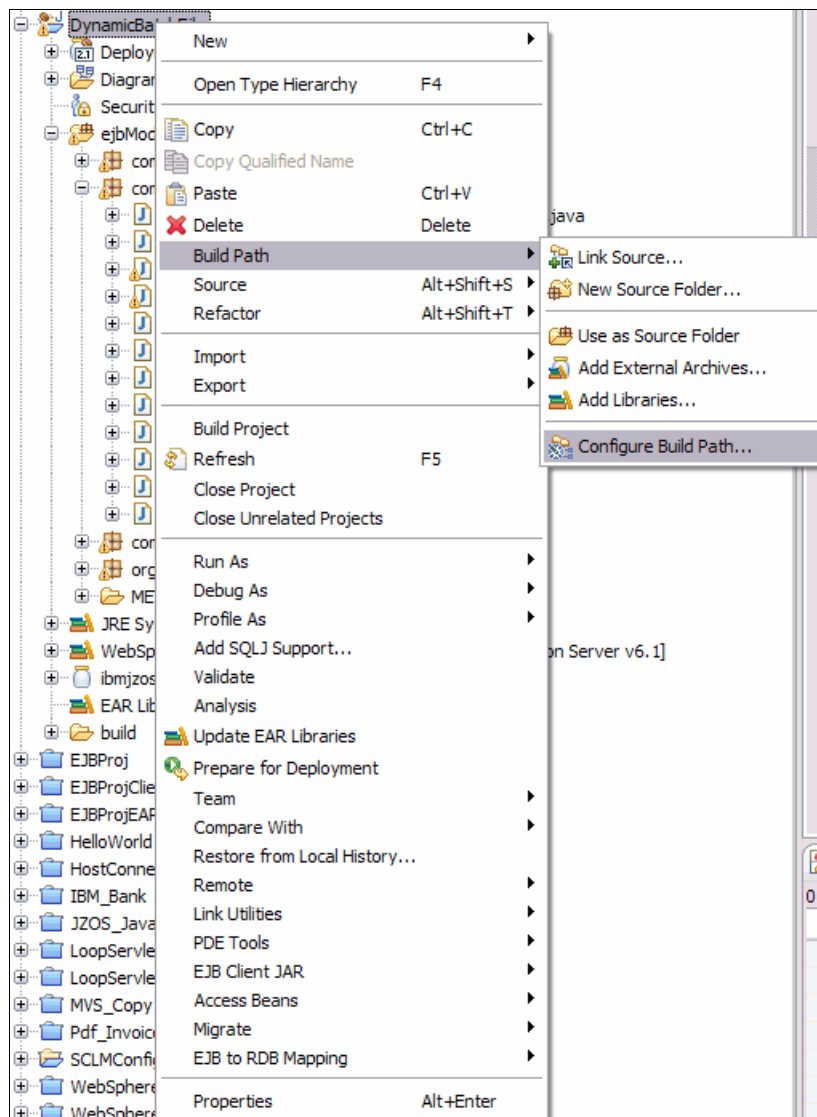


Figure 14-4 Configuring Rational Application Developer Build Path

7. In the JAR Selection window, select **Add JARs**. select the `ibmjzos.jar` file, as shown in Figure 14-5.

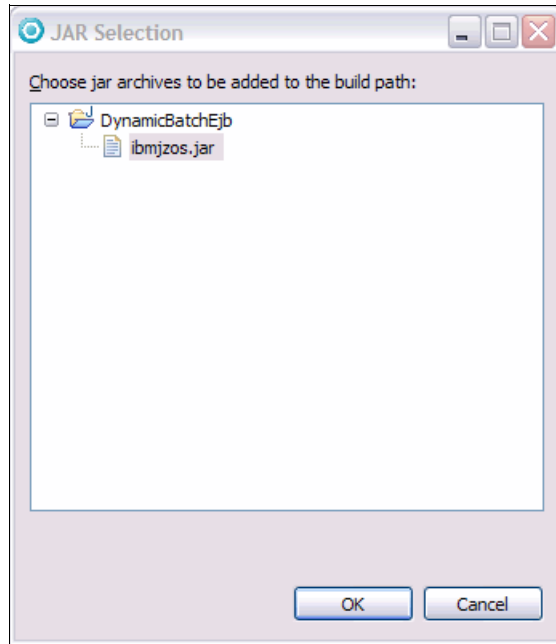


Figure 14-5 Adding `ibmjzos.jar` file to Build Path

8. Click **OK**. Apply the settings by pressing **OK** again. Now, all code errors disappear in Rational Application Developer.
9. With the help of the two recently created Java classes, you can submit a z/OS job by passing a job name as an MVS data set name in the form of `//dataset.name` or as a UNIX file path to the `run()` method in `MvsJobSubmitter.java`. To call this method by the `TriggerBatchBean`, we open the `TriggerBatchBean.java` file in the Project Explorer.
10. Then, change the `public String execute(String[] parameters)` as shown in Example 14-3. Because you need only one parameter from the servlet form (the job name), perform a check within this method as to whether the parameters are valid. If yes, call the `MvsJobSubmitter.run()` method, and return to the servlet that the job was submitted. If not, return to the servlet that the parameters were wrong.

Example 14-3 `TriggerBatchBean` execute method

```
public String execute(String[] parameters)
{
    String message="";
    for (int i=1; i<parameters.length; i++)
    {
        if (parameters[i].equals("")==false || parameters[0].equals(""))
        {
            message="Please enter the job name in parameter 1, the other
parameters have to left empty.";
        }
        else if (parameters[0].equals("")==false)
        {
            try {
                MvsJobSubmitter submitter = new MvsJobSubmitter();
                submitter.run(parameters[0]);
            }
        }
    }
}
```

```

        message="Job "+parameters[0]+" successfully submitted. Please
check output.";
    } catch (IOException e) {
        e.printStackTrace();
        message="Error: "+e.getMessage();
    }
}
}
return message;
}
}

```

11. The project is now ready for packaging. Select the following projects in the Project Explorer:

- DynamicBatch
- DynamicBatchEar
- DynamicBatchEjb

12. Then, right-click the select projects, and select **Prepare for Deployment** as shown Figure 14-6.

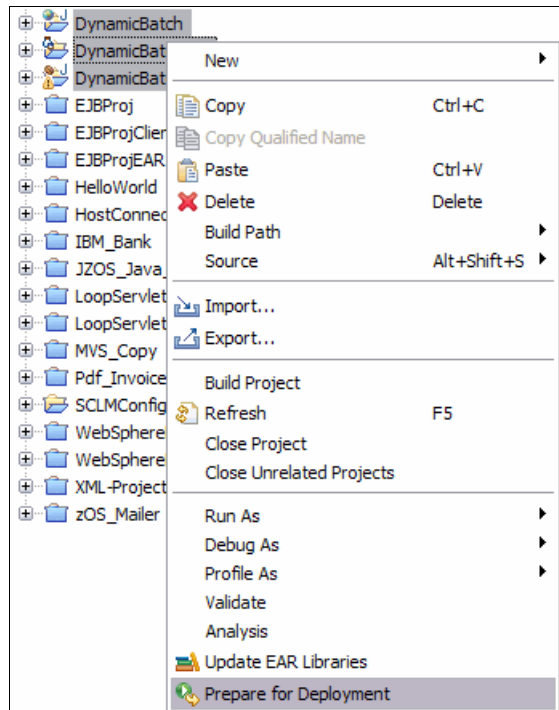


Figure 14-6 Preparing projects for deployment in Rational Application Developer

13. Next, right-click the DynamicBatchEar project and select **Export** → **Ear file** as shown in Figure 14-7.

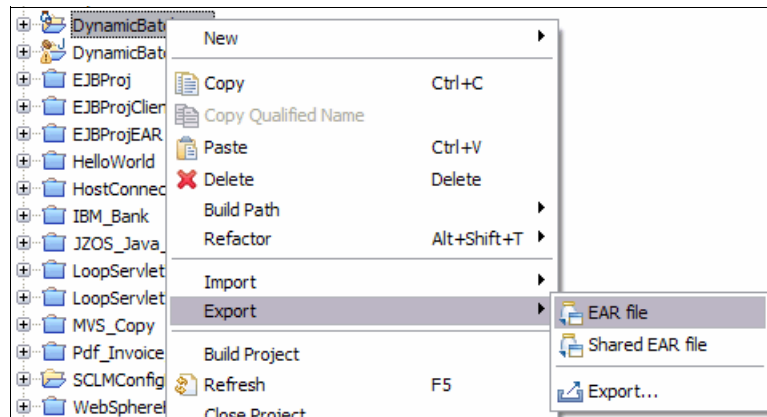


Figure 14-7 Export EAR file in Rational Application Developer

14. Browse for a file location where you want to store this EAR file, and select **Finish** (see Figure 14-8).

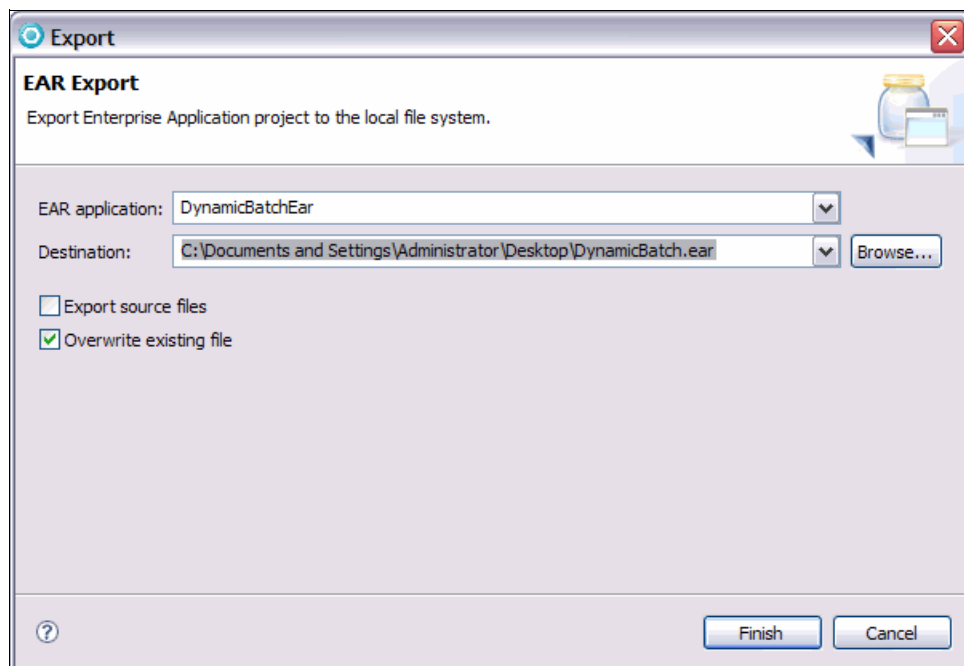


Figure 14-8 Exporting EAR file in Rational Application Developer

14.1.2 Configuring WebSphere Application Server z/OS for deployment

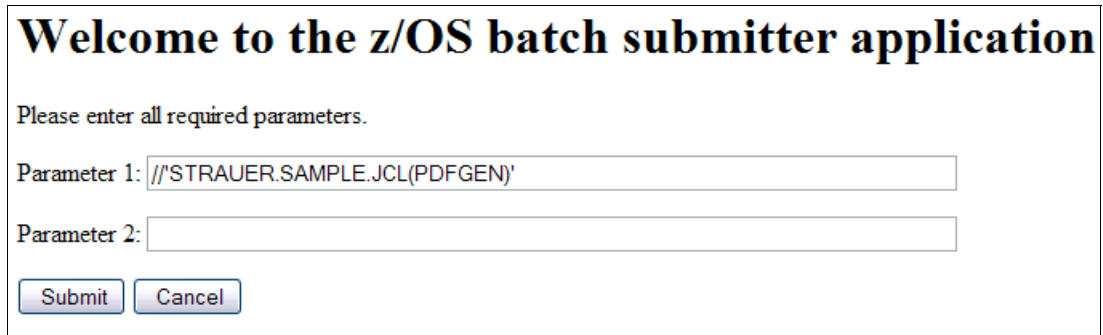
Before you deploy the EAR file to WebSphere Application Server for z/OS, you have to add a JVM parameter to the Servant:

1. Select **Servers** → **Application Servers** → **ServerName** → **Server Infrastructure** → **Process Definition** → **Servant** → **Java Virtual Machine**.
2. Under **Generic JVM arguments**, add:

```
-Djzos.script.path=${JAVA_HOME}/mvstools/samples/scripts
```

This parameter is needed to tell the JVM where to find certain REXX scripts that take care of the job submission that is initiated by the JZOS Toolkit classes.
3. Then, select **OK**, and **Save**.
4. Now, you can deploy the EAR file of the previous to WebSphere Application Server, and start the application.
5. After deploying the application, call the application by entering the following address in the browser:

```
http://<hostname>:<port>/DynamicBatch/DynamicBatch
```
6. In the browser, enter a job name as *Parameter 1*, and click **Submit** (see Figure 14-9). This job name can either be a MVS data set or a z/OS UNIX file.



Welcome to the z/OS batch submitter application

Please enter all required parameters.

Parameter 1:

Parameter 2:

Figure 14-9 Submitting a job with the sample application in native Java

7. Finally, verify the job results in SDSF.

Note: Because we are submitting jobs out of WebSphere Application Server, the submitted job will run under the user ID of WebSphere Application Server.

14.2 Using WebSphere XD Compute Grid trigger mechanisms

The WebSphere XD Compute Grid component provides ways of managing the scheduling and execution control of background activities in a grid computing environment. The various ways that you can manage your Compute Grid environment include using the Job Management Console (JMC), analyzing job logs, specifying job classes, and using classification rules:

- ▶ Using the JMC directly from a browser
- ▶ Using the command line interface
- ▶ Using WSGrid command-line utility
- ▶ Web services and EJB interfaces for the job scheduler

Using the JMC, you can:

- ▶ Submit jobs
- ▶ Monitor job execution
- ▶ Perform operational actions against jobs
- ▶ View job logs
- ▶ Manage the job repository
- ▶ Manage job schedules
 - A *job log* is a file that contains a detailed record of the execution details of a job. It is comprised of both system and application messages. Job logs are stored on the endpoints where the job runs and on the application server that hosts the job scheduler. Job logs are viewable through the JMC and from the command line.
 - A *job class* establishes a policy for resource consumption by a set of grid jobs. Through this policy, execution time, number of concurrent jobs, job log and job output queue storage can be controlled. Each job is assigned to a job class. A default job class is provided for jobs that do not specify a class.
 - *Classification rules* are saved in a configuration file named `gridclassrules.xml` under the configuration directory of WebSphere Application Server. In WebSphere XD, there is one `gridclassrules.xml` per cell, and the rules are ordered based on the priority element.

You can find more detailed information for job logs, job classes, and job classifications in the WebSphere XD Compute Grid Information Center at:

<http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r1/index.jsp>

14.2.1 The Job Management Console

WebSphere XD Compute Grid introduces the Job Management Console (JMC), a stand-alone Web interface for managing jobs. Interaction with the job repository is also possible, where jobs can be saved, removed or submitted to the repository. This console provides controlled access when security is enabled. Only authorized users who are granted the `lsubmitter` or `lradmin` roles through the WebSphere XD administrative console can be allowed access to the JMC.

Through the JMC, you can submit job schedules with a preferred processing time. Also, you can configure job schedules so that they can occur or recur on a specific time of day or week, and so on. When you are ready to submit a job, you can choose to delay the submission of it by specifying the start date and time of when you want to run the job.

To access the Job Management Console:

1. Configure the job scheduler.
2. Ensure that the job scheduler is running.
3. In a browser, enter:

```
http://<job_scheduler_server_host>:<port>/jmc
```

For example:

```
https://wtsc48.itso.ibm.com:9575/jmc/console.jsp
```

4. If an On-Demand Router (ODR) is defined in the cell, enter:

```
http://<odr_host>:80/jmc
```

To explain the functionality of the JMC, we use the Echo sample application from 8.7, “Developing a WebSphere batch application” on page 109. The Echo application is installed

in the WebSphere XD Compute Grid Execution Endpoint (GEE) and started. So, you can now submit batch jobs. First, open the JMC in the browser. The window shown in Figure 14-10 opens.

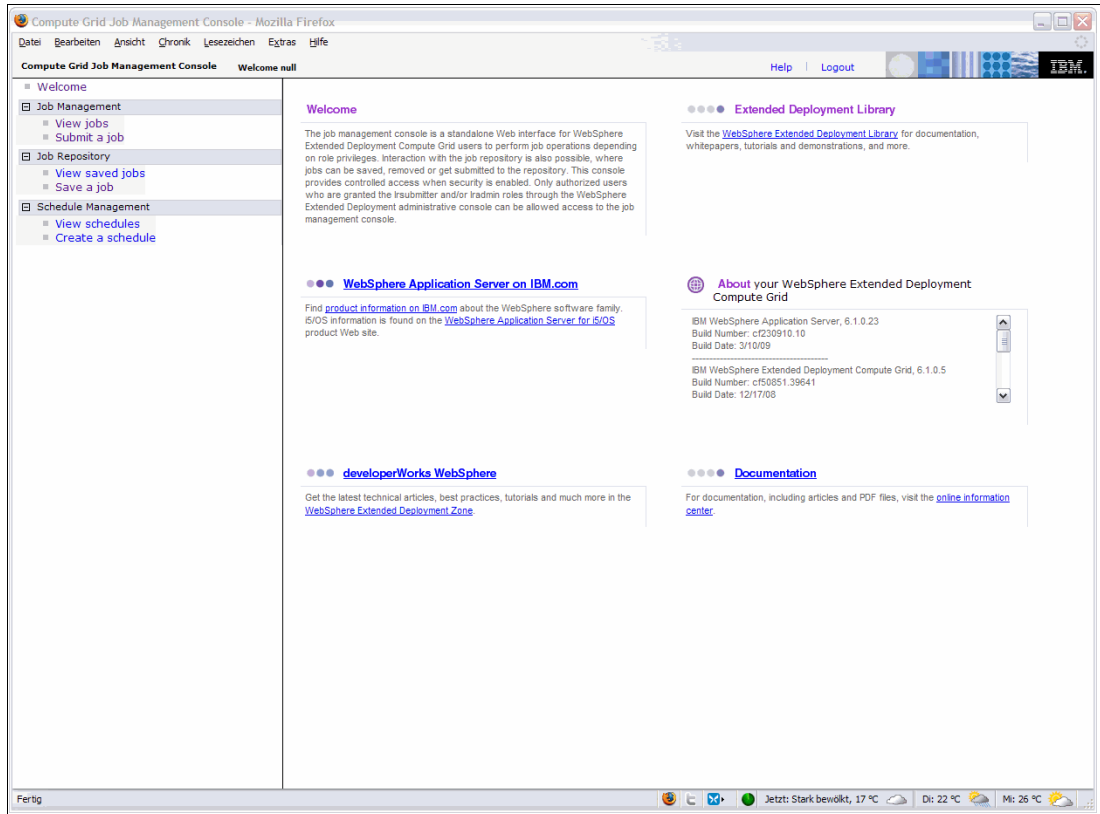


Figure 14-10 The WebSphere XD Compute Grid Job Management Console

Through the JMC, you can perform the following functions:

- ▶ Store and manage the job repository. Select **Save a job** under **Job Repository**. Enter a Job name and browse, for example, to the xJCL zOSEcho.xml in your workspace. See Figure 14-11.

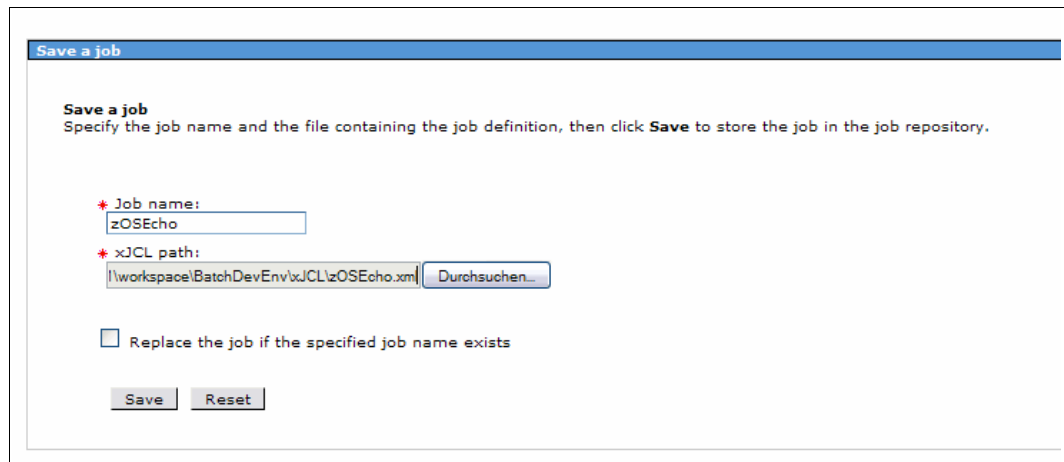


Figure 14-11 Store a job in the job repository

- ▶ View jobs in the job repository. Select **View saved jobs** under **Job Repository**. Your z0SEcho job should be stored in the job repository. See Figure 14-12.

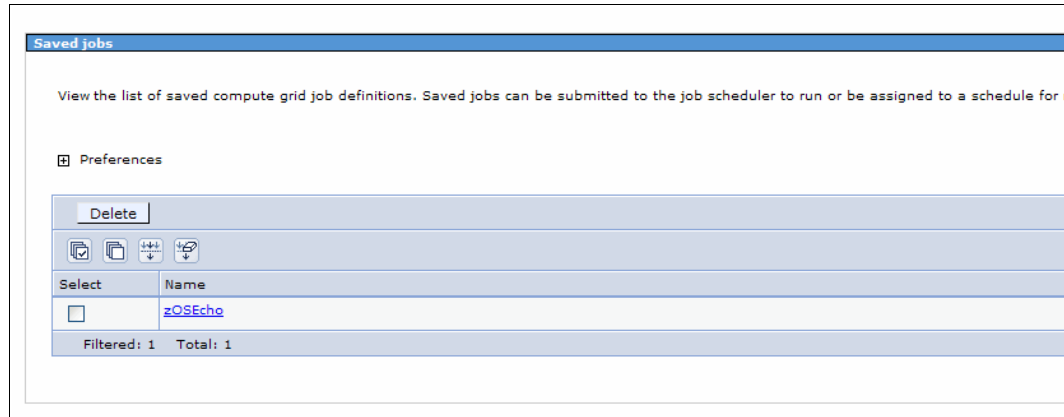


Figure 14-12 View jobs in the job repository

- ▶ Submit jobs. Select **Submit** a job under **Job Management**. Choose **Job repository** and browse to the job you want to submit, for example z0SEcho. Click **Submit**. See Figure 14-13.

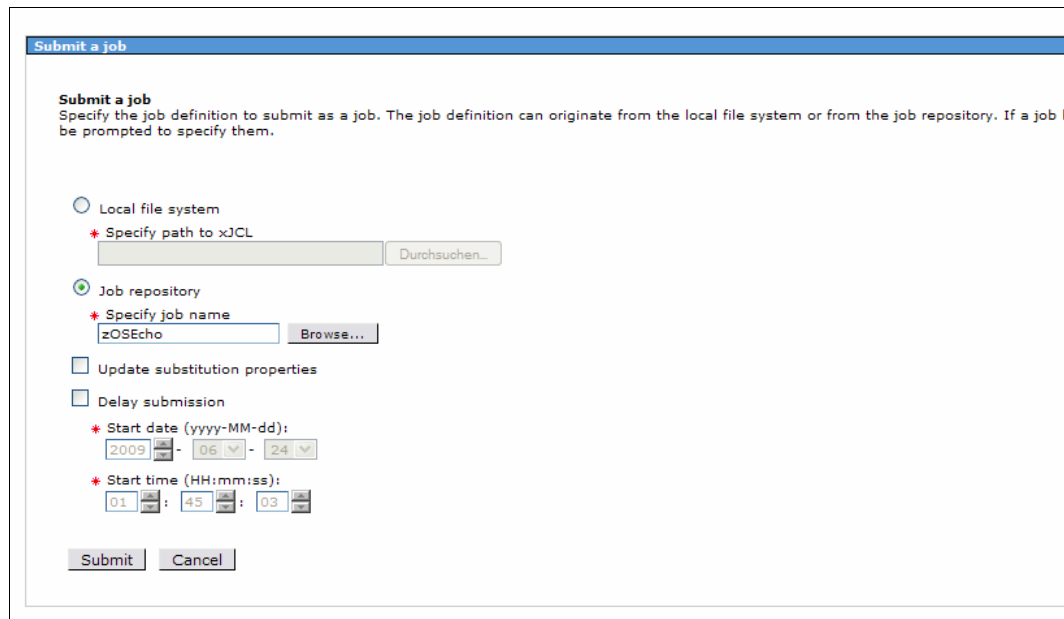


Figure 14-13 Submit job from the job repository with JMC

- View job logs. Select **View jobs** under **Job Management** and choose the job with <Job-name>:<job-id> that you want to view. You see the job output, and you can also download the job log as *.txt file. See Figure 14-14.

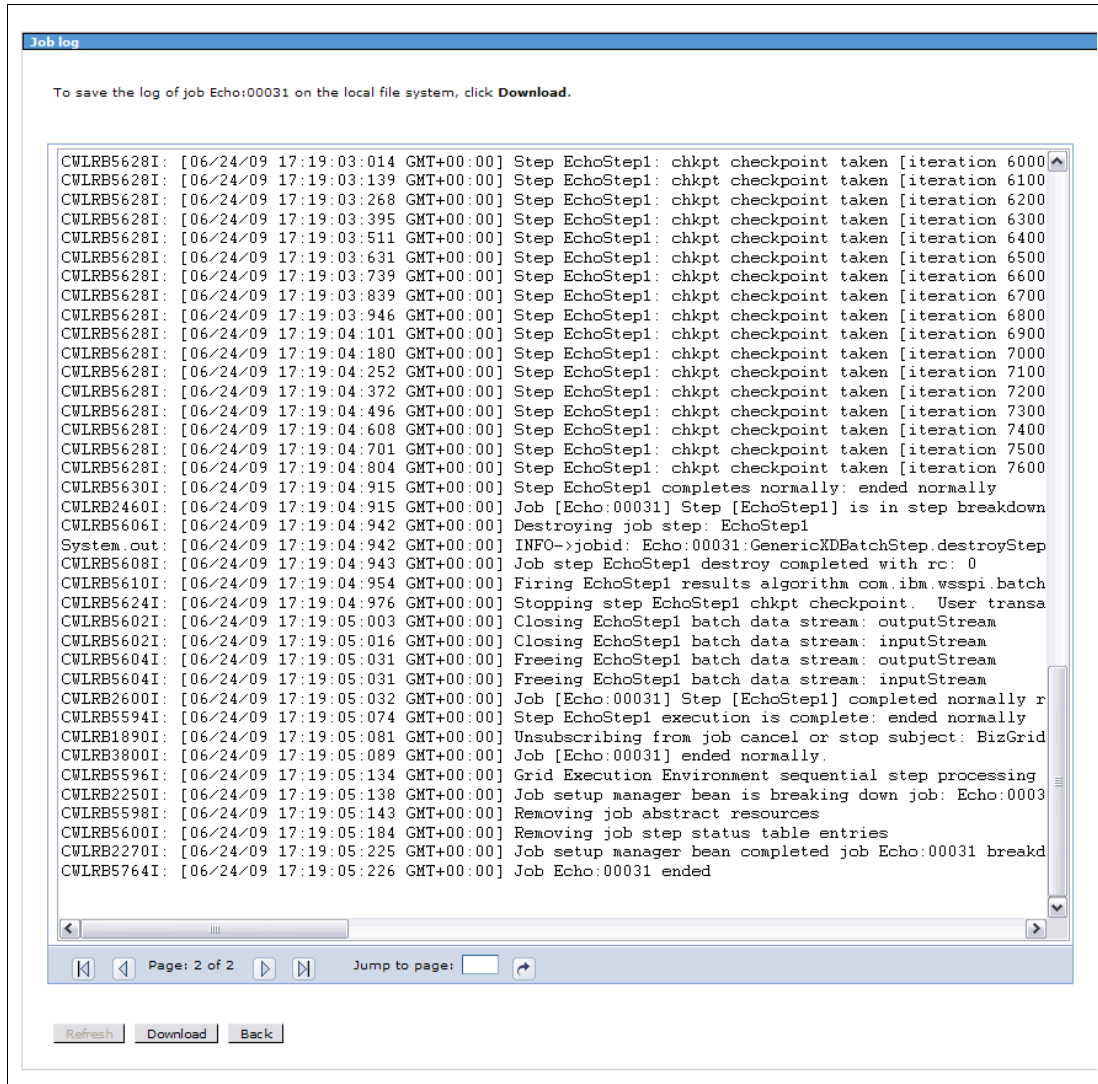


Figure 14-14 View jobs in the JMC

- ▶ Manage job schedules and view schedules. Select **Create a schedule** to set up a job schedule and under **View schedules** you can see all your scheduled jobs. See Figure 14-15.

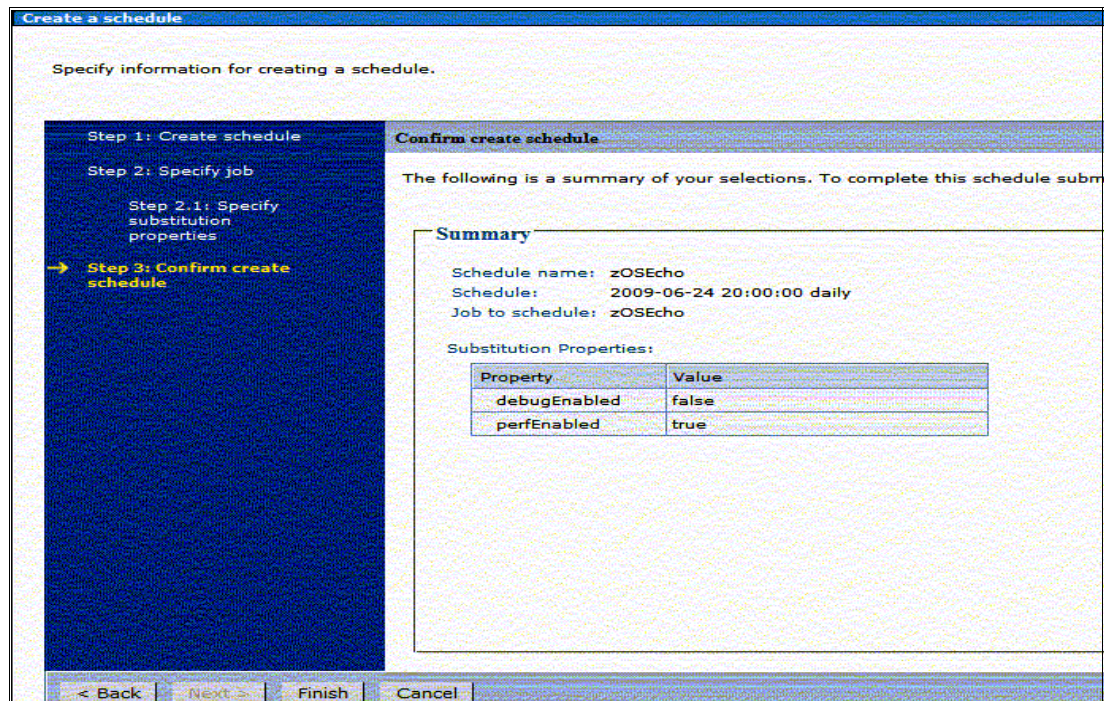


Figure 14-15 Schedule a job with JMC

- ▶ Monitor job execution
- ▶ Perform operational actions against jobs

To access the field help for the JMC, click **Help** in the upper, right corner of every job management panel. See Figure 14-16.

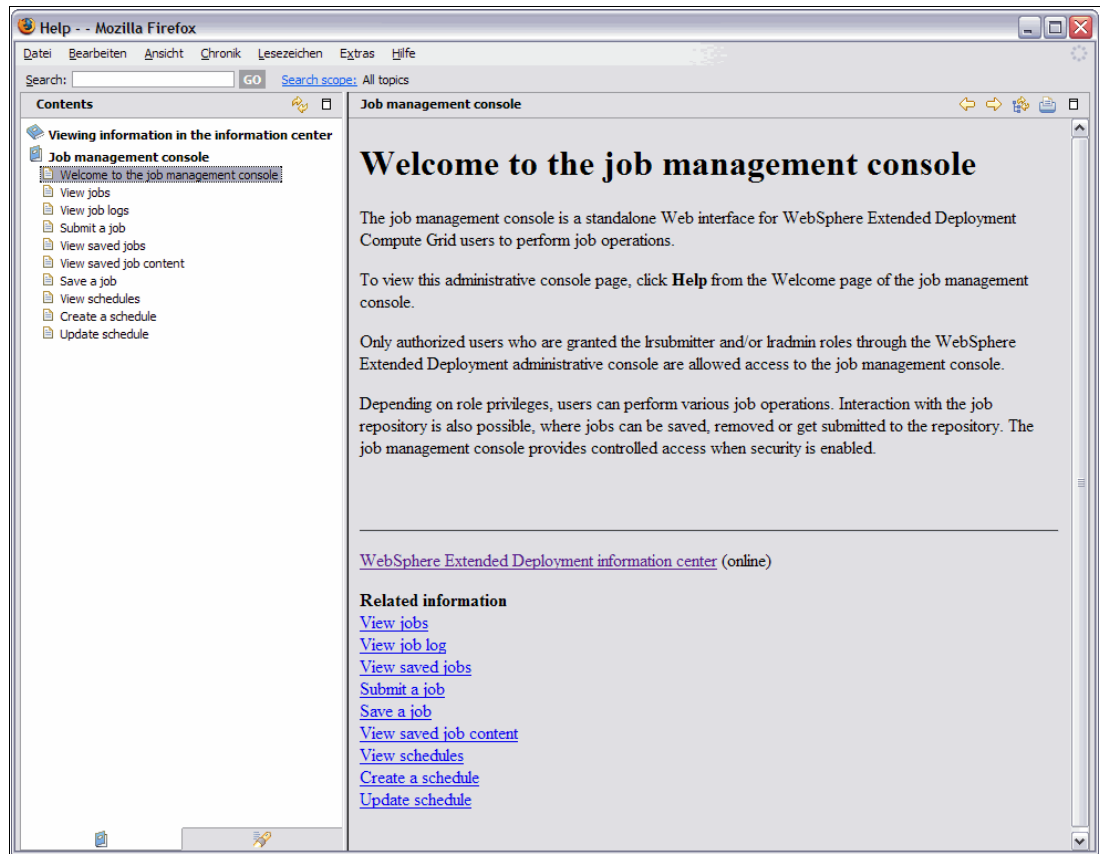


Figure 14-16 Help for JMC

14.2.2 The command-line interface

The command-line interface interacts with the job scheduler to submit and manipulate a WebSphere XD Compute Grid job. It is located in the `was_root/bin` directory as the `lrcmd.sh` script and can be invoked from any location in the WebSphere cell.

Use the `lrcmd` script to perform the following operations:

- ▶ Display usage information for `lrcmd`
- ▶ Submit a job to the job scheduler
- ▶ Cancel a previously submitted job
- ▶ Restart a job
- ▶ Purge job information
- ▶ Save an xJCL to the job repository
- ▶ Remove a job from the job repository
- ▶ Show xJCL stored in the job repository
- ▶ Show the status of a Compute Grid job
- ▶ Suspend a job
- ▶ Resume start of a previously suspended job
- ▶ Display the output for a job
- ▶ Displays the return code of a batch job
- ▶ Submit a recurring job request to the job scheduler
- ▶ Modify an existing recurring job request

- ▶ Cancel an existing recurring job request
- ▶ List all existing recurring job requests
- ▶ Show all recurring jobs of a request

You can find a complete list about how to invoke all operations in the WebSphere XD Compute Grid Information Center at:

<http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r1/index.jsp>

To find the utility, go to `<WAS_HOME>/bin` in the TSO OMVS session, for example, `/wasalconfig/V6R1/alcell/alnodea/AppServer/profiles/default/bin`. The `lrcmd.sh` script is in this directory. Example 14-4 shows an example of submitting a job with the command-line interface.

Example 14-4 Example of submitting a batch job

```
./lrcmd.sh -cmd=submit -job=z0SEcho -host=wtsc48.itso.ibm.com -port=9575
```

```
CWLRB4960I: Wed Jun 24 15:23:11 EDT 2009 : com.ibm.ws.batch.wsbatch : Job [Echo:00032] is submitted.
```

Example 14-5 shows the status coming back after the job has ended.

Example 14-5 Example of receiving the job status of a submitted batch job

```
<-- jobid=Echo:00032
```

```
./lrcmd.sh -cmd=status -jobid=Echo:00032 -host=wtsc48.itso.ibm.com -port=9575
```

```
CWLRB4940I: com.ibm.ws.batch.wsbatch : -cmd=status -jobid=Echo:00032 -host=wtsc48.itso.ibm.com -port=9575
```

```
CWLRB5160I: Wed Jun 24 15:26:08 EDT 2009 : Job [Echo:00032] execution has ended.
```

Example 14-6 shows an example of retrieving job output.

Example 14-6 Example of retrieving output of a batch job

```
./lrcmd.sh -cmd=output -jobid=Echo:00032 -host=wtsc48.itso.ibm.com -port=9575
```

```
CWLRB4940I: com.ibm.websphere.batch.wsbatch : -cmd=output -jobid=Echo:00032
```

```
CWLRB5000I: Wed Jun 24 15:26:55 EDT 2009 : com.ibm.ws.batch.wsbatch : response to output
```

```
CWLRB1740I: [06/24/09 19:23:15:578 GMT+00:00] Job [Echo:00032] is in job setup.
```

```
CWLRB1760I: [06/24/09 19:23:15:856 GMT+00:00] Job [Echo:00032] is submitted for execution.
```

```
CWLRB2420I: [06/24/09 19:23:16:440 GMT+00:00] Job [Echo:00032] Step [EchoStep1] is in step setup.
```

```
CWLRB2440I: [06/24/09 19:23:16:794 GMT+00:00] Job [Echo:00032] Step [EchoStep1] is dispatched.
```

```
CWLRB2460I: [06/24/09 19:23:25:020 GMT+00:00] Job [Echo:00032] Step [EchoStep1] is in step breakdown.
```

```
CWLRB2600I: [06/24/09 19:23:25:116 GMT+00:00] Job [Echo:00032] Step [EchoStep1] completed normally rc=0.
```

```
CWLRB3800I: [06/24/09 19:23:25:168 GMT+00:00] Job [Echo:00032] ended normally.
```

```
End
```

Figure 14-17 shows the JMC showing the status of the job submitted using the command line interface.

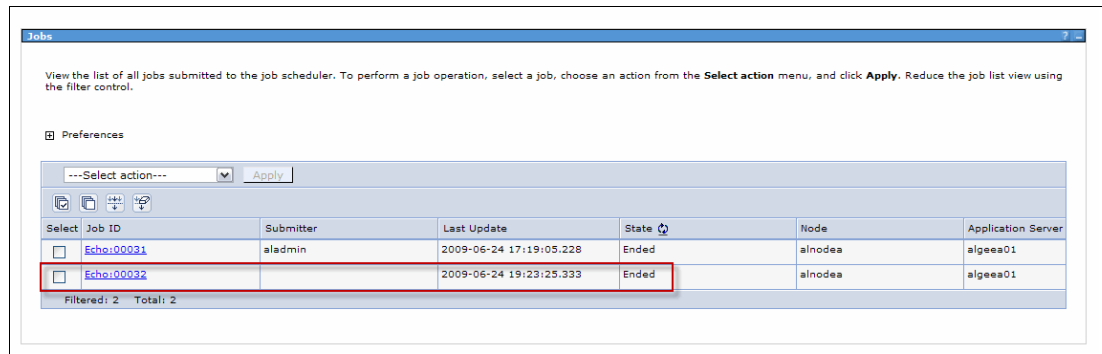


Figure 14-17 Run zOSEcho with command line utility: View submitted job from console in JMC

14.2.3 The WSGrid command-line utility

The WSGrid utility is designed to facilitate integration of WebSphere XD Compute Grid jobs with external workload schedulers. You can use this utility to sub-dispatch a WebSphere XD Compute Grid job.

The WSGrid utility is a client application of the Job Scheduler message-driven interface. Use the WSGrid utility to facilitate control of WebSphere XD Compute Grid batch jobs by external workload schedulers, such as Tivoli Workload Scheduler. The WSGrid utility is invoked by an external workload scheduler as part of a larger task choreography. The WSGrid utility submits a WebSphere XD Compute Grid batch job, then waits for its completion. The job log from the WebSphere batch job is written to the standard output stream of the environment in which the WSGrid utility was invoked.

WSGrid is a shell script that uses a JMS interface of the Job Scheduler to submit jobs into a WebSphere XD Compute Grid topology. Its function is to provide an integration point between the Job Scheduler and enterprise scheduling products such as Tivoli Workload Scheduler. Logs of jobs submitted by WSGrid are written to the output stream of the environment from which WSGrid is invoked. When the job completes, WSGrid returns with the return code of the batch job. To set up WSGrid and submitting jobs:

1. Invoke `wsgridConfig.py` to install the JobschedulerMDI application and associated JMS artefacts:
 - a. From a shell, change to the `<WAS_HOME>/<cell>/<node>/AppServer/bin` directory. In our case, we change to the following directory:


```
/wasalconfig/V6R1/alcell/alnodea/AppServer/profiles/default/bin
```
 - b. Invoke the command shown in Example 14-7 to run the `wsgridConfig.py` script to install the JobschedulerMDI application and associated JMS artifacts:

Example 14-7 Run the wsgridConfig.py script

```
./wsadmin.sh -f wsgridConfig.py -install -node alnodea -server allrsa01
-providers wtsc48.itso.ibm.com,9565
```

See Example 14-8 for the results.

Note: The SIB_ENDPOINT_ADDRESS port of each server or cluster member that hosts the Job Scheduler; in this case this is allrsa01 on node alnodea. You need to list the SIB_ENDPOINT_ADDRESS ports of *all* servers that host the job_scheduler. In the case of the environment, we have a only single server hosting the Job Scheduler.

Example 14-8 Output of the wsgridConfig.py script

```
/wasalconfig/V6R1/alcell/alnodea/AppServer/profiles/default/bin>./wsadmin.sh -f
wsgridConfig.py -install -node alnodea -server allrsa01 -providers
wtsc48.itso.ibm.com,9565
WASX7209I: Connected to process "dmgr" on node aldm using SOAP connector; The t
ype of process is: DeploymentManager
```

```
WASX7303I: The following options are passed to the scripting environment and are
available as arguments that are stored in the argv variable: "-install, -node,
alnodea, -server, allrsa01, -providers, wtsc48.itso.ibm.com,9565"
```

wsgridConfig.py is performing install using

```
target cluster or server : /Node:alnodea/Server:allrsa01/
MDI application name      : JobSchedulerMDI
JMS connection factory   : com.ibm.ws.grid.ConnectionFactory
JMS activation spec      : com.ibm.ws.grid.ActivationSpec
JMS input queue name     : com.ibm.ws.grid.InputQueue
JMS output queue name    : com.ibm.ws.grid.OutputQueue
JMS file store root      : /tmp/JobSchedulerBus
SIB identifier           : JobSchedulerBus
endpoint provider list   : wtsc48.itso.ibm.com:9565
```

```
ADMA5016I: Installation of JobSchedulerMDI started.
```

```
ADMA5058I: Application and module versions are validated with versions of deploy
ment targets.
```

```
ADMA5005I: The application JobSchedulerMDI is configured in the WebSphere Applic
ation Server repository.
```

```
ADMA5053I: The library references for the installed optional package are created
.
```

```
ADMA5005I: The application JobSchedulerMDI is configured in the WebSphere Applic
ation Server repository.
```

```
ADMA5110I: The application JobSchedulerMDI is installed as a hidden application
and will not be exposed via administrative interfaces such as GUI client, wsadmi
n or MBean Java API. In order to perform management operations on this applicat
ion, the application name must be known.
```

```
ADMA5005I: The application JobSchedulerMDI is configured in the WebSphere Applic
ation Server repository.
```

```
SECJ0400I: Successfully updated the application JobSchedulerMDI with the appConte
xtIDForSecurity information.
```

```
ADMA5005I: The application JobSchedulerMDI is configured in the WebSphere Applic
ation Server repository.
```

```
ADMA5011I: The cleanup of the temp directory for application JobSchedulerMDI is
complete.
```

```
ADMA5013I: Application JobSchedulerMDI installed successfully.
saving config...
```

```
Done saving..
wsgridConfig.py INFO: Configuration was saved and synchronized to the active nodes
starting JobSchedulerMDI

wsgridConfig.py INFO: installed JobSchedulerMDI to node/server alnodea/allrsa01
Done installing JobSchedulerMDI
createBus JobSchedulerBus
wsgridConfig.py INFO: created SIB JobSchedulerBus
saving config...
Done saving..
wsgridConfig.py INFO: Configuration was saved and synchronized to the active nodes
wsgridConfig.py INFO: created SIB member on target /Node:alnodea/Server:allrsa01/
saving config...
Done saving..
wsgridConfig.py INFO: Configuration was saved and synchronized to the active nodes
wsgridConfig.py INFO: created SIB destination com.ibm.ws.grid.InputQueue
saving config...
Done saving..
wsgridConfig.py INFO: Configuration was saved and synchronized to the active nodes
wsgridConfig.py INFO: created SIB destination com.ibm.ws.grid.OutputQueue
saving config...
Done saving..
wsgridConfig.py INFO: Configuration was saved and synchronized to the active nodes
wsgridConfig.py INFO: created JMS connection factory com.ibm.ws.grid.ConnectionFactory
saving config...
Done saving..
wsgridConfig.py INFO: Configuration was saved and synchronized to the active nodes
wsgridConfig.py INFO: created JMS queue com.ibm.ws.grid.InputQueue
saving config...
Done saving..
wsgridConfig.py INFO: Configuration was saved and synchronized to the active nodes
wsgridConfig.py INFO: created JMS queue com.ibm.ws.grid.OutputQueue
saving config...
Done saving..
wsgridConfig.py INFO: Configuration was saved and synchronized to the active nodes
wsgridConfig.py INFO: created JMS activation spec com.ibm.ws.grid.ActivationSpec
saving config...
Done saving..
wsgridConfig.py INFO: Configuration was saved and synchronized to the active nodes
```

You can check in the WebSphere administrative console the tasks that `wsgriidConfig.py` performs during the installation:

- J2C connection factory, `com.ibm.ws.grid.ConnectionFactory`, as shown in Figure 14-18.

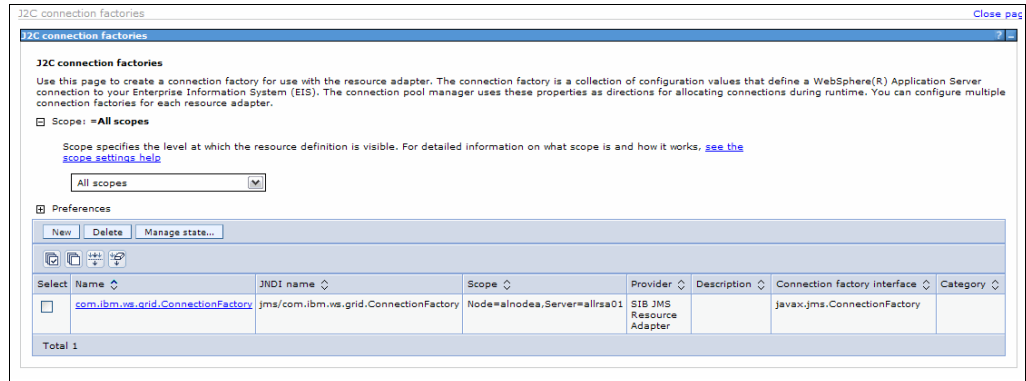


Figure 14-18 J2C connection factory

- JMS activation specification, `com.ibm.ws.grid.ActivationSpec`, as shown in Figure 14-19.

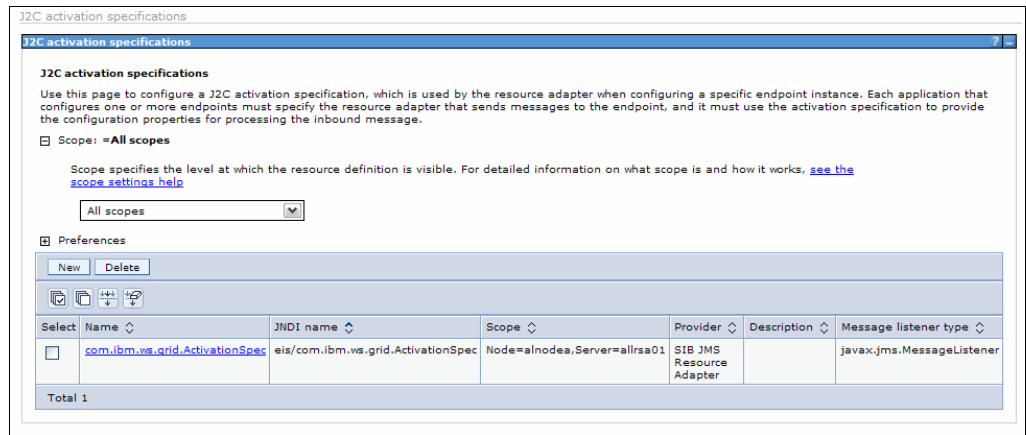


Figure 14-19 J2C activation specification

- JMS input queue name, `com.ibm.ws.grid.InputQueue`, and JMS output queue name, `com.ibm.ws.grid.OutputQueue`, as shown in Figure 14-20.

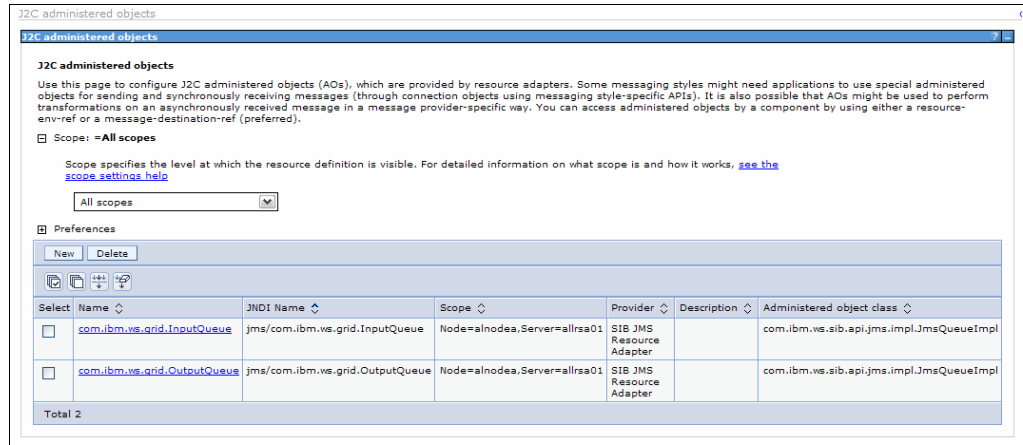


Figure 14-20 J2C administered objects

- SIB identifier, `JobSchedulerBus`, as shown in Figure 14-21.

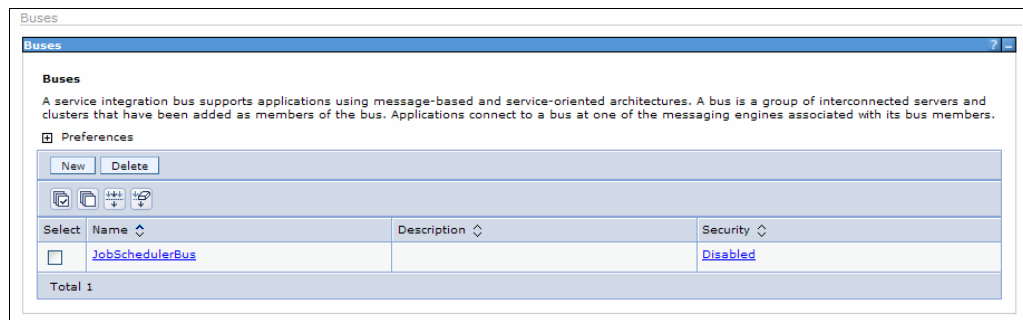


Figure 14-21 Bus

After WSGrid is configured, you can launch it using the `WSGrid.sh` shell script. The script takes in a control file as parameter which is a simple properties file that contains information about the scheduler location and the batch job to submit.

2. Run the `WSgrid.sh` script using the commands we describe here. WSGrid supports distinct ways of specifying a WebSphere job:
 - a. You can specify an xJCL file. In this case, the format of invoking the command is as follows:

```
WSGrid.sh <control properties file> <path to job xJCL>
```

The control properties file is a fully qualified path name to the file that contains the WSGrid job properties:

- Scheduler-host property is the scheduler server host machine name.
- Scheduler-port property is the `WC_defaulthost` port of the scheduler server. This information can be retrieved from the administrative console at **Application Servers** → **<name of scheduler server or cluster member>** → **Ports**.

See Example 14-9 for an example of these properties.

Example 14-9 Sample job properties file without job information - wsgrid.cntl

```
#scheduler-host=<hostname>
scheduler-host=wtsc48.itso.ibm.com
#scheduler-port=<port>
scheduler-port=9575
#properties
debug=false
```

- b. You can specify the name of an xJCL file in the job repository. In this case the format of the command is as follows:

```
WSGrid.sh <control properties file>
```

The control properties file is a fully qualified path name to the file that contains the WSGrid control properties.

In addition to the scheduler-host and the scheduler-port, the information of the xJCL in the repository is provided. The example zOSEchowsgrid.cntl file, as shown in Example 14-10, is included in the workspace in the folder cntl.

Example 14-10 Sample job zOSEchowsgrid.cntl properties file with job information

```
#scheduler-host=<hostname>
scheduler-host=wtsc48.itso.ibm.com
#scheduler-port=<port>
scheduler-port=9575
#repository-job=<jobname>
repository-job=zOSEcho
#properties
debug=false
```

The output of the command is shown in Example 14-11.

Example 14-11 Run Echo sample with ./WSGrid.sh zOSEchowsgrid.cntl

```
cd <WAS_HOME>/bin {f.e. /wasalconfig/V6R1/alce11/alnodea/AppServer/bin}
```

```
./WSGrid.sh zOSEchowsgrid.cntl
```

```
...
CWLRB5618I: Y06/25/09 13:16:01:166 GMT+00:00` Initializing step EchoStep1 batch
data stream inputStream
CWLRB5620I: Y06/25/09 13:16:01:166 GMT+00:00` Opening step EchoStep1 batch data
stream inputStream
CWLRB5622I: Y06/25/09 13:16:01:180 GMT+00:00` Loading job step bean for step Ech
oStep1 using jndi name: ejb/GenericXDBatchStep
CWLRB5594I: Y06/25/09 13:16:01:191 GMT+00:00` Step EchoStep1 setup is complete:
ended normally
CWLRB2440I: Y06/25/09 13:16:01:225 GMT+00:00` Job YEcho:00035` Step YEchoStep1`
is dispatched.
...
CWLRB5628I: Y06/25/09 13:16:04:930 GMT+00:00` Step EchoStep1: chkpt checkpoint t
aken Yiteration 44000`
CWLRB5628I: Y06/25/09 13:16:05:000 GMT+00:00` Step EchoStep1: chkpt checkpoint t
aken Yiteration 45000`
```

CWLRB5628I: Ý06/25/09 13:16:05:104 GMT+00:00` Step EchoStep1: chkpt checkpoint t
aken Ýiteration 46000`
...
CWLRB5628I: Ý06/25/09 13:16:07:058 GMT+00:00` Step EchoStep1: chkpt checkpoint t
aken Ýiteration 75000`
CWLRB5628I: Ý06/25/09 13:16:07:111 GMT+00:00` Step EchoStep1: chkpt checkpoint t
aken Ýiteration 76000`
CWLRB5630I: Ý06/25/09 13:16:07:157 GMT+00:00` Step EchoStep1 completes normally:
ended normally
CWLRB2460I: Ý06/25/09 13:16:07:157 GMT+00:00` Job ÝEcho:00035` Step ÝEchoStep1`
is in step breakdown.
CWLRB5606I: Ý06/25/09 13:16:07:183 GMT+00:00` Destroying job step: EchoStep1
System.out: Ý06/25/09 13:16:07:183 GMT+00:00` INFO->jobid: Echo:00035:GenericXDB
atchStep.destroyStep()- Total Execution Time: 5993
CWLRB5608I: Ý06/25/09 13:16:07:183 GMT+00:00` Job step EchoStep1 destroy complet
ed with rc: 0
CWLRB5610I: Ý06/25/09 13:16:07:185 GMT+00:00` Firing EchoStep1 results algorithm
com.ibm.wsspi.batch.resultsalgorithms.jobsum: ÝRC 0` ÝjobRC 0`
CWLRB5624I: Ý06/25/09 13:16:07:188 GMT+00:00` Stopping step EchoStep1 chkpt chec
kpoint. User transaction status: STATUS_ACTIVE
CWLRB5602I: Ý06/25/09 13:16:07:193 GMT+00:00` Closing EchoStep1 batch data strea
m: outputStream
CWLRB5602I: Ý06/25/09 13:16:07:212 GMT+00:00` Closing EchoStep1 batch data strea
m: inputStream
CWLRB5604I: Ý06/25/09 13:16:07:216 GMT+00:00` Freeing EchoStep1 batch data strea
m: outputStream
CWLRB5604I: Ý06/25/09 13:16:07:216 GMT+00:00` Freeing EchoStep1 batch data strea
m: inputStream
CWLRB2600I: Ý06/25/09 13:16:07:216 GMT+00:00` Job ÝEcho:00035` Step ÝEchoStep1`
completed normally rc=0.
CWLRB5594I: Ý06/25/09 13:16:07:241 GMT+00:00` Step EchoStep1 execution is comple
te: ended normally
CWLRB1890I: Ý06/25/09 13:16:07:244 GMT+00:00` Unsubscribing from job cancel or s
top subject: BizGridJobCancel_Echo:00035
CWLRB3800I: Ý06/25/09 13:16:07:248 GMT+00:00` Job ÝEcho:00035` ended normally.
CWLRB5596I: Ý06/25/09 13:16:07:281 GMT+00:00` Grid Execution Environment sequent
ial step processing complete: ended
CWLRB2250I: Ý06/25/09 13:16:07:283 GMT+00:00` Job setup manager bean is breaking
down job: Echo:00035
CWLRB5598I: Ý06/25/09 13:16:07:285 GMT+00:00` Removing job abstract resources
CWLRB5600I: Ý06/25/09 13:16:07:289 GMT+00:00` Removing job step status table ent
ries
CWLRB2270I: Ý06/25/09 13:16:07:293 GMT+00:00` Job setup manager bean completed j
ob Echo:00035 breakdown
CWLRB5764I: Ý06/25/09 13:16:07:294 GMT+00:00` Job Echo:00035 ended
CWLRB3880I: Job ÝEcho:00035` ending status: RC=0

See Figure 14-22 for the status of the job in the JMC.

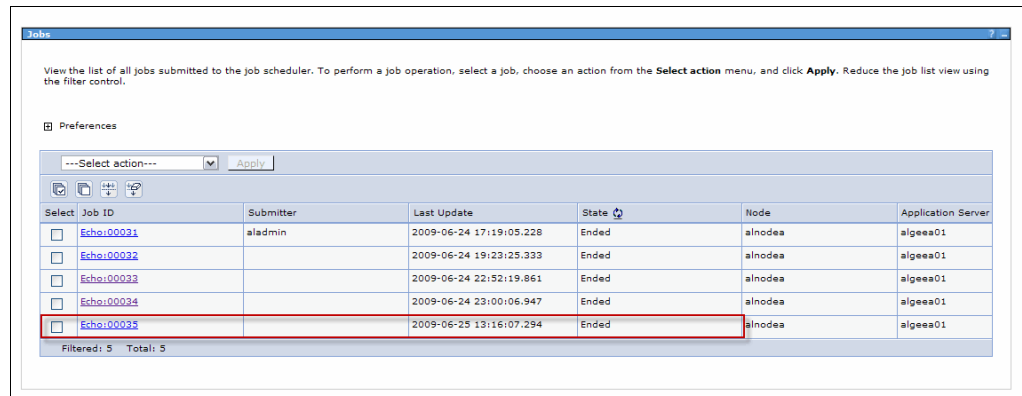


Figure 14-22 Run zOSEcho with WSGrid: View submitted job using WSGrid in JMC

14.2.4 Web services and EJB interfaces for the Job Scheduler

Schedulers enable Java EE application tasks to run at a requested time. Schedulers also enable application developers to create their own stateless session EJB components to receive event notifications during a task life cycle, allowing the plugging-in of custom logging utilities or workflow applications.

The Job Scheduler supports programmatic access to its functions over both an enterprise bean (EJB) interface for Java EE applications and a Web Services interface for both Java EE and non-Java EE applications.

Using the base scheduler in WebSphere with the Job Scheduler

The Job Scheduler EJB interface is used to programmatically submit and manipulate a WebSphere XD Compute Grid job. You can use the EJB interface in conjunction with the base scheduler in WebSphere Application Server for z/OS to perform calendar-based submission of a WebSphere XD Compute Grid job. This sample describes how to submit a WebSphere XD Compute Grid job to the Job Scheduler using the base scheduler in WebSphere Application Server.

Developing the code

First, you have to look up a configured scheduler. Each configured scheduler is available from two different programming models.

- ▶ A Java EE server application, such as a servlet or EJB component can use the Scheduler API. Schedulers are accessed by looking them up using a JNDI name or resource reference.
- ▶ Java Management Extensions (JMX) applications, such as `wsadmin` scripts, can use the Scheduler API using `WASScheduler` MBeans.

Note 1: In this sample, we use a servlet or an EJB module. All necessary information for using the JMX API is provided in the WebSphere XD Compute Grid Information Center.

You can also find the entire project that we develop in this section in Appendix C, “Additional material” on page 453.

Follow these steps:

1. First, switch to the J2EE Perspective in Rational Application Developer. Then, import the projects of the application Scheduler Wrapper Test application template in Rational Application Developer by selecting **File** → **Import** → **General** → **Existing Project into Workspace** → **Next**. Choose **Select archive file**, and browse for the SchedulerWrapperProject.zip file of the additional material of this book, and press **Finish**.

Your Rational Application Developer Project Explorer should now look similar that shown Figure 14-23.

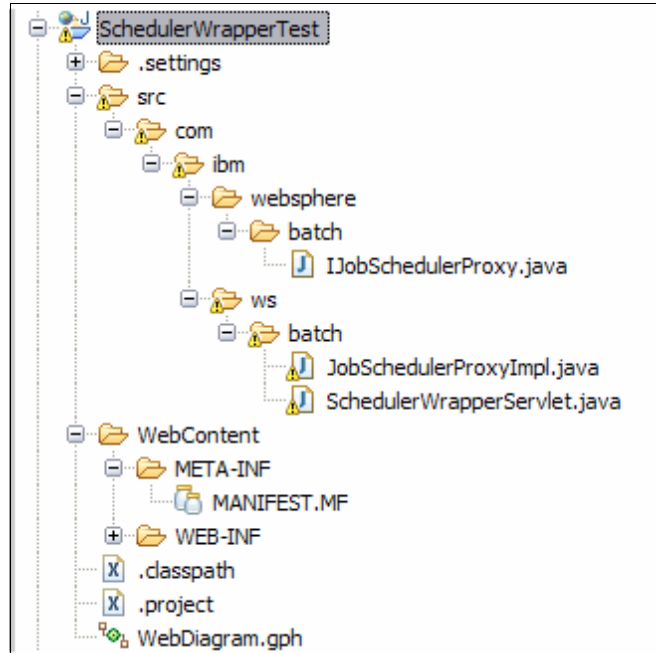


Figure 14-23 Rational Application Developer Project Explorer

- Next, include the `batchruntime.jar` file by right-clicking **SchedulerWrapperTest** project and selecting **Build Path** → **Configure Build Path**. Then, select **Add JARs**. Locate the `batchruntime.jar` on your local machine, for example `C:\WebSphere\ApplicationServer\lib`, and add this JAR to your build path. Click **OK**. See Figure 14-24.

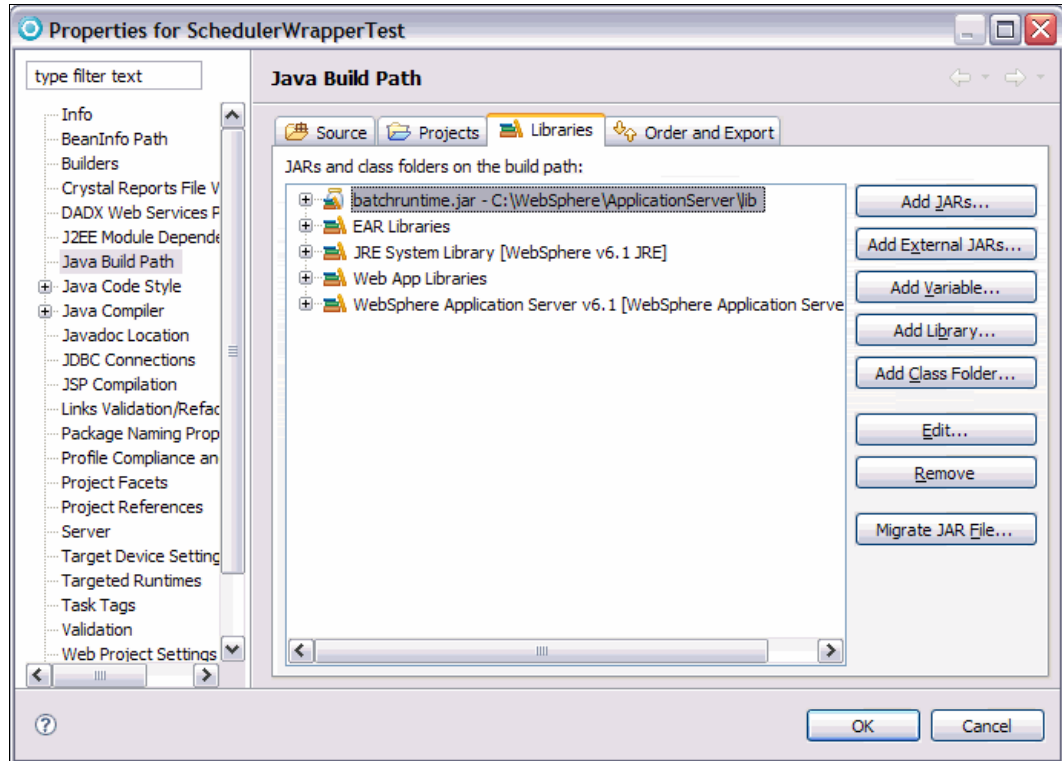


Figure 14-24 Configure Build Path for SchedulerWrapper project

A Java EE server application, such as a servlet or EJB component can use the Scheduler API. Schedulers are accessed by looking them up using a JNDI name or resource reference. Locate Schedulers using the `javax.naming.Context.lookup()` method from a J2EE server application, such as a servlet or EJB module.

- In the `JobSchedulerProxyImpl.java` file, insert the code shown in Figure 14-12.

Example 14-12 JobSchedulerProxyImpl.java

```

package com.ibm.ws.batch;

import java.rmi.RemoteException;
import java.util.Hashtable;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;

import com.ibm.websphere.batch.IJobSchedulerProxy;
import com.ibm.websphere.longrun.InvalidJobIDException;
import com.ibm.websphere.longrun.InvalidOperationException;
import com.ibm.websphere.longrun.JCLException;
import com.ibm.websphere.longrun.JobScheduler;
import com.ibm.websphere.longrun.JobSchedulerHome;
import com.ibm.websphere.longrun.JobSubmissionException;
import com.ibm.websphere.longrun.SchedulerException;

public class JobSchedulerProxyImpl implements IJobSchedulerProxy {

    private JobSchedulerHome jsHome;

    private String schedulerHostName;

```

```

private String schedulerNodeName;
private String schedulerServerName;
private String schedulerBootstrapPort;
private String schedulerClusterName;

private JobScheduler getJobScheduler() {
    JobScheduler js = null;
    JobSchedulerHome jsHome = null;
    try {
        Hashtable env = new Hashtable();
        InitialContext ctxt = null;
        String longRunningContext = null;

        env.put (
            Context.INITIAL_CONTEXT_FACTORY,
            "com.ibm.websphere.naming.WsnInitialContextFactory"
        );

        env.put(
            Context.PROVIDER_URL,
            "corbaloc:iiop:"
            + schedulerHostName
            + ":"
            + schedulerBootstrapPort
            + "/NameServiceCellRoot"
        );

        ctxt = new InitialContext(env);

        if(schedulerNodeName != null && schedulerServerName != null) {
            longRunningContext = "nodes/" + schedulerNodeName + "/servers/" + schedulerServerName ;
        }
        else
            if(schedulerClusterName != null) {
                longRunningContext = "clusters/" + schedulerClusterName;
            }

        String longRunningScheduler = "/ejb/com/ibm/websphere/longrun/JobSchedulerHome";

        Object o = ctxt.lookup( longRunningContext + longRunningScheduler );

        jsHome = (JobSchedulerHome) PortableRemoteObject.narrow( o, JobSchedulerHome.class );
        js = jsHome.create();

    } catch (Exception e) { throw new RuntimeException( "unable to obtain JobSchedulerHome: " + e ); }

    return js;
}

public String submitJob(String xJCL) throws
    RemoteException, SchedulerException, JCLEException, JobSubmissionException {

    JobScheduler js = getJobScheduler();
    String jobId = js.submitJob(xJCL);
    return jobId;
}

public void cancelJob(String jobId) throws
    RemoteException, InvalidOperationException, InvalidJobIDException, SchedulerException {

    JobScheduler js = getJobScheduler();
    js.cancelJob(jobid);
}

public void restartJob(String jobId) throws
    RemoteException, InvalidJobIDException, InvalidOperationException,
    SchedulerException, JCLEException, JobSubmissionException {

    JobScheduler js = getJobScheduler();
    js.restartJob(jobid);
}

public int getJobRC(String jobId) throws
    RemoteException, InvalidOperationException, InvalidJobIDException, SchedulerException {

    JobScheduler js = getJobScheduler();
    int rc = js.getBatchJobRC(jobid);
    return rc;
}

public int getJobStatus(String jobId) throws RemoteException, InvalidJobIDException, SchedulerException {
    JobScheduler js = getJobScheduler();
    int rc = js.getJobStatus(jobid);
}

```



```

        return rc;
    }

    public JobSchedulerProxyImpl(String schedulerHostName,String schedulerNodeName,
        String schedulerServerName, String schedulerBootstrapPort) {
        this.schedulerHostName = schedulerHostName;
        this.schedulerNodeName = schedulerNodeName;
        this.schedulerServerName = schedulerServerName;
        this.schedulerBootstrapPort = schedulerBootstrapPort;
    }

    public JobSchedulerProxyImpl(String schedulerClusterName,
        String schedulerBootstrapPort, String schedulerHostName ) {
        this.schedulerHostName = schedulerHostName;
        this.schedulerBootstrapPort = schedulerBootstrapPort;
        this.schedulerClusterName = schedulerClusterName;
    }
}

```

The Scheduler is now available to use from a J2EE server application. The source is included in the `com.ibm.ws.batch` package.

In the servlet `SchedulerWrapperServlet.java`, you need to hard code the following variables:

- Host name
- Scheduler name
- Node name
- Bootstrap port
- Cluster name

The host name and the bootstrap port is always required. Depending on your WebSphere XD Compute Grid environment you have to set the cluster name in a clustered environment or the server and node name in a non-clustered environment.

Also, ensure that the xJCL is placed in a string, for example, by reading an xJCL file into a string. In this example, the `simpleEchoJob` xJCLs implements as a string, the same xJCL we used in the other WebSphere XD Compute Grid samples. Depending if you use the local UTE or the z/OS environment you have to change the file names for the input and output file in the xJCL, for example:

- local UTE
 - `<prop name=\"FILENAME\" value=\"C:\\output.txt\"></prop>`
 - `<prop name=\"FILENAME\" value=\"C:\\input.txt\"></prop>`
- z/OS environment
 - `<prop name=\"FILENAME\" value=\"/u/sthomas/data/output.txt\"></prop>`
 - `<prop name=\"FILENAME\" value=\"/u/sthomas/data/input.txt\"></prop>`

The source of the `SchedulerWrapperServlet` is shown in Example 14-13.

Example 14-13 SchedulerWrapperServlet source

```

package com.ibm.ws.batch;

import java.io.IOException;
import java.io.PrintWriter;
import java.rmi.RemoteException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.ibm.websphere.batch.IJobSchedulerProxy;
import com.ibm.websphere.longrun.InvalidJobIDException;
import com.ibm.websphere.longrun.InvalidOperationException;

```

```

import com.ibm.websphere.longrun.JCLEException;
import com.ibm.websphere.longrun.JobSubmissionException;
import com.ibm.websphere.longrun.SchedulerException;

/**
 * Servlet implementation class for Servlet: SchedulerWrapperServlet
 *
 */
public class SchedulerWrapperServlet extends javax.servlet.http.HttpServlet implements
javax.servlet.Servlet {
    /* WebSphere zOS
    private String schedulerHostName = "wtsc48.itso.ibm.com";
    private String schedulerNodeName = "alnodea";
    private String schedulerServerName = "allrsa01";
    private String schedulerBootstrapPort = "9560";
    private String schedulerClusterName = null;
    */

    // local WebSphere UTE
    private String schedulerHostName = "localhost";
    private String schedulerNodeName = "suthomasNode01";
    private String schedulerServerName = "server1";
    private String schedulerBootstrapPort = "2812";
    private String schedulerClusterName = null;

    IJobSchedulerProxy schedProxy = null;
    /*

    private static final String simpleEchoJob=

        "<?xml version=\"1.0\" encoding=\"UTF-8\"?>          "+
        "<job default-application-name=\"Echo\" name=\"Echo\"
xml:ns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\"> +
"<jndi-name>ejb/com/ibm/ws/batch/EchoBatchController</jndi-name>      "+
        "<step-scheduling-criteria>          "+
        "<scheduling-mode>sequential</scheduling-mode>          "+
        "</step-scheduling-criteria>          "+
        "<checkpoint-algorithm name=\"chkpt\">          "+
        "<classname>com.ibm.wsspi.batch.checkpointalgorithms.RecordbasedBase</classname>"+
        "<props>          "+
        "<prop name=\"recordcount\" value=\"1000\"></prop>          "+
        "</props>          "+
        "</checkpoint-algorithm>          "+
        "<results-algorithms>          "+
        "<results-algorithm name=\"jobsum\">          "+
        "<classname>com.ibm.wsspi.batch.resultsalgorithms.jobsum</classname>"+
        "</results-algorithm>          "+
        "</results-algorithms>          "+
        "<job-step name=\"Step1\">          "+
        "<jndi-name>ejb/GenericXDBatchStep</jndi-name>          "+
        "<checkpoint-algorithm-ref name=\"chkpt\"></checkpoint-algorithm-ref>"+
        "<results-ref name=\"jobsum\"></results-ref>          "+
        "<batch-data-streams>          "+
        "<bds>          "+
        "<logical-name>outputStream</logical-name>          "+
        "<props>          "+
        "<prop name=\"FILENAME\" value=\"C:\\output.txt\"></prop>          "+
        "<prop name=\"EnablePerformanceMeasurement\" value=\"false\"></prop>"+
        "<prop name=\"EnableDetailedPerformanceMeasurement\" value=\"false\"></prop>"+

```

```

"<prop name=\"AppendJobIdToFileName\" value=\"false\"></prop>          "+
"<prop name=\"debug\" value=\"false\"></prop>                          "+
"<prop name=\"PATTERN_IMPL_CLASS\" value=\"com.batch.streams.outputstreams.EchoWriter\"></prop>"+
"</props>                                                                "+
"<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteWriter</impl-class>
"+
"</bds>                                                                    "+
"<bds>                                                                    "+
"<logical-name>inputStream</logical-name>                                "+
"<props>                                                                    "+
"<prop name=\"FILENAME\" value=\"C:\\input.txt\"></prop>                "+
"<prop name=\"EnablePerformanceMeasurement\" value=\"false\"></prop>"+
"<prop name=\"EnableDetailedPerformanceMeasurement\" value=\"false\"></prop>"+
"<prop name=\"debug\" value=\"false\"></prop>                            "+
"<prop name=\"PATTERN_IMPL_CLASS\" value=\"com.batch.streams.inputstreams.EchoReader\"></prop>"+
"</props>                                                                    "+
"<impl-class>com.ibm.websphere.batch.devframework.datastreams.patterns.FileByteReader</impl-class>
"+
"</bds>                                                                    "+
"</batch-data-streams>                                                    "+
"<props>                                                                    "+
"<prop name=\"EnablePerformanceMeasurement\" value=\"false\"></prop>"+
"<prop name=\"EnableDetailedPerformanceMeasurement\" value=\"false\"></prop>"+
"<prop name=\"debug\" value=\"false\"></prop>                            "+
"<prop name=\"BATCHRECORDPROCESSOR\" value=\"com.batch.steps.Echo\"></prop>"+
"</props>                                                                    "+
"</job-step>                                                                "+
"</job>";

/* (non-Java-doc)
 * @see javax.servlet.http.HttpServlet#HttpServlet()
 */
public SchedulerWrapperServlet() {
    super();
}

/* (non-Javadoc)
 * @see javax.servlet.GenericServlet#init()
 */
public void init() throws ServletException {
    // TODO Auto-generated method stub
    super.init();

    if (schedulerClusterName==null){
        schedProxy = new JobSchedulerProxyImpl(schedulerHostName, schedulerNodeName, schedulerServerName,
schedulerBootstrapPort);
    }
    else {
        schedProxy = new JobSchedulerProxyImpl(schedulerClusterName, schedulerBootstrapPort,
schedulerHostName);
    }
}

/* (non-Java-doc)
 * @see javax.servlet.http.HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
    String cmd = request.getParameter("command");

```

```

if(cmd != null && !cmd.equals("")) {

    if(cmd.equalsIgnoreCase("submit")) {
        _submitJob(request, response);
    } else if(cmd.equalsIgnoreCase("cancel")) {
        _cancelJob(request, response);
    } else if(cmd.equalsIgnoreCase("getJobStatus")) {
        _getJobStatus(request, response);
    } else if(cmd.equalsIgnoreCase("getJobRC")) {
        _getJobRC(request, response);
    }
}
}

private void _getJobRC(HttpServletRequest request, HttpServletResponse response) {
    try {
        String jobId = request.getParameter("jobid");
        if(jobId != null && !jobId.equals("")) {
            int jobRC = schedProxy.getJobRC(jobId);
            PrintWriter pw = response.getWriter();
            pw.print("RC of Job: " + jobId + " is: " + jobRC);
        }
    } catch (RemoteException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (SchedulerException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (InvalidJobIDException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (InvalidOperationException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

private void _getJobStatus(HttpServletRequest request, HttpServletResponse response) {
    try {
        String jobId = request.getParameter("jobid");
        if(jobId != null && !jobId.equals("")) {
            int status = schedProxy.getJobStatus(jobId);
            PrintWriter pw = response.getWriter();
            pw.print("Status of Job: " + jobId + " is: " + status);
        }
    } catch (RemoteException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (SchedulerException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (InvalidJobIDException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

```

```

    }
}

private void _cancelJob(HttpServletRequest request, HttpServletResponse response) {
    try {
        String jobId = request.getParameter("jobid");
        if(jobId != null && !jobId.equals("")) {
            schedProxy.cancelJob(jobId);
            PrintWriter pw = response.getWriter();
            pw.print("Cancelled Job: " + jobId);
        }
    } catch (RemoteException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (SchedulerException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (InvalidOperationException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (InvalidJobIDException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

private void _submitJob(HttpServletRequest request, HttpServletResponse response) {
    try {
        //String jobId = schedProxy.submitJob(simpleCIJob);
        String jobId = schedProxy.submitJob(simpleEchoJob);
        PrintWriter pw = response.getWriter();
        pw.print("Submitted Job: " + jobId);
    } catch (RemoteException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (SchedulerException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (JCLEException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (JobSubmissionException e){
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
/* (non-Java-doc)
 * @see javax.servlet.http.HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
    // TODO Auto-generated method stub
}
}

```

4. Now, export the SchedulerWrapperTest application as a *.war file. Right-click **SchedulerWrapperTest** in the Eclipse environment, and choose **Export** as shown in Figure 14-25.

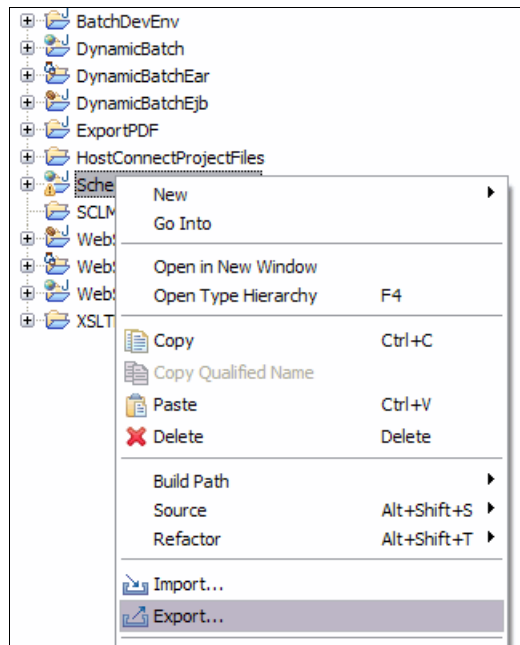


Figure 14-25 Export SchedulerWrapperTest from Eclipse environment

5. Select **WAR file** as type for export, and click **Next**, as shown in Figure 14-26.

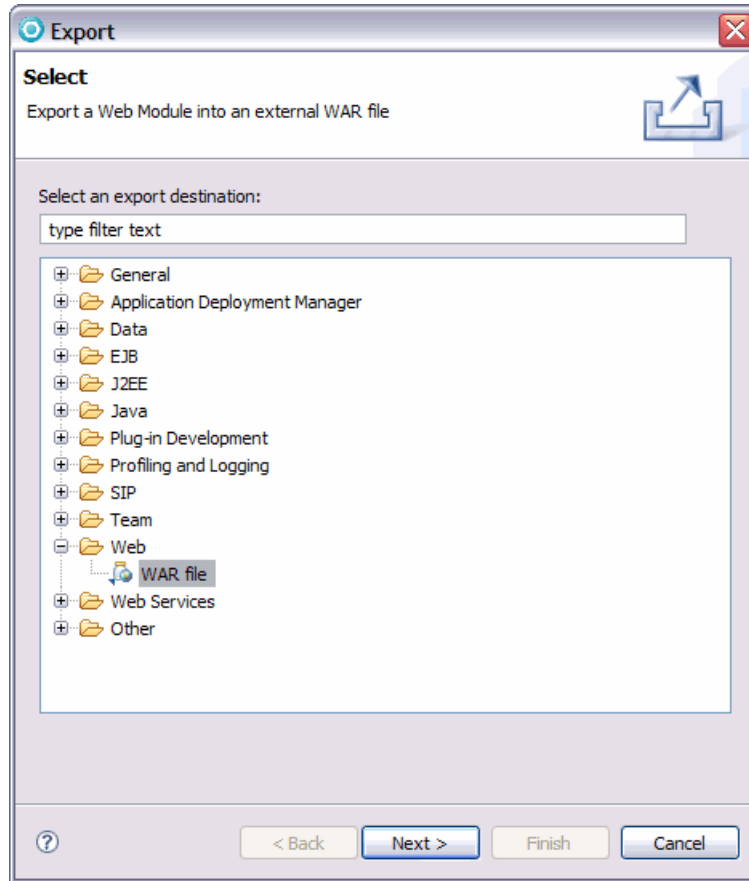


Figure 14-26 Export SchedulerWrapperTest as Web module

6. Select the destination for the exported SchedulerWrapperTest WAR file, and click **Finish**, as shown in Figure 14-27.

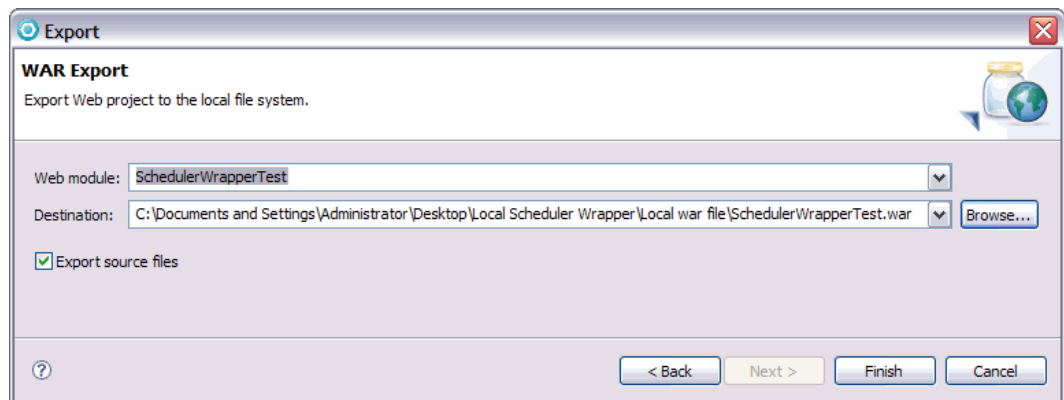


Figure 14-27 Export Web project to the local file system

The SchedulerWrapperTest.war file will be created in the directory and can now be installed in the WebSphere environment for testing the application.

You have to log in to the administrative console on the local UTE or on the z/OS environment depending where you want to install the application. The installation process is the same in both environments. Select **Applications** → **Install New Application**, as shown in Figure 14-28.

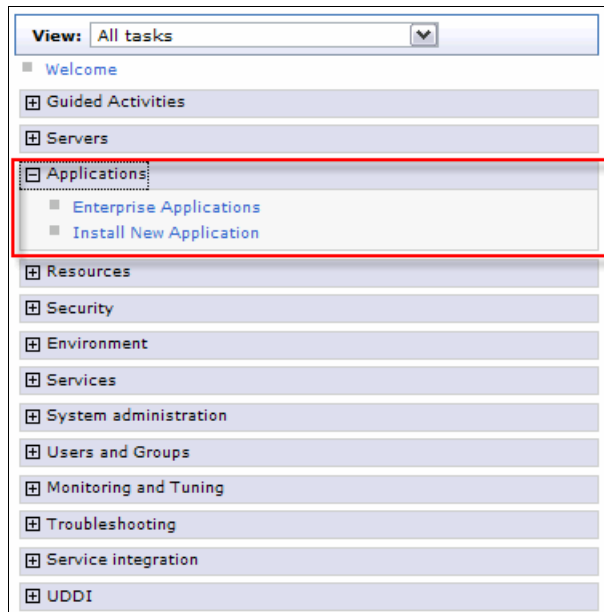


Figure 14-28 SchedulerWrapperTest.war - Install application

7. Browse to the SchedulerWrapperTest.war file on your local file system, type /SchedulerWrapperTest as context root, and click **Next**. See Figure 14-29.

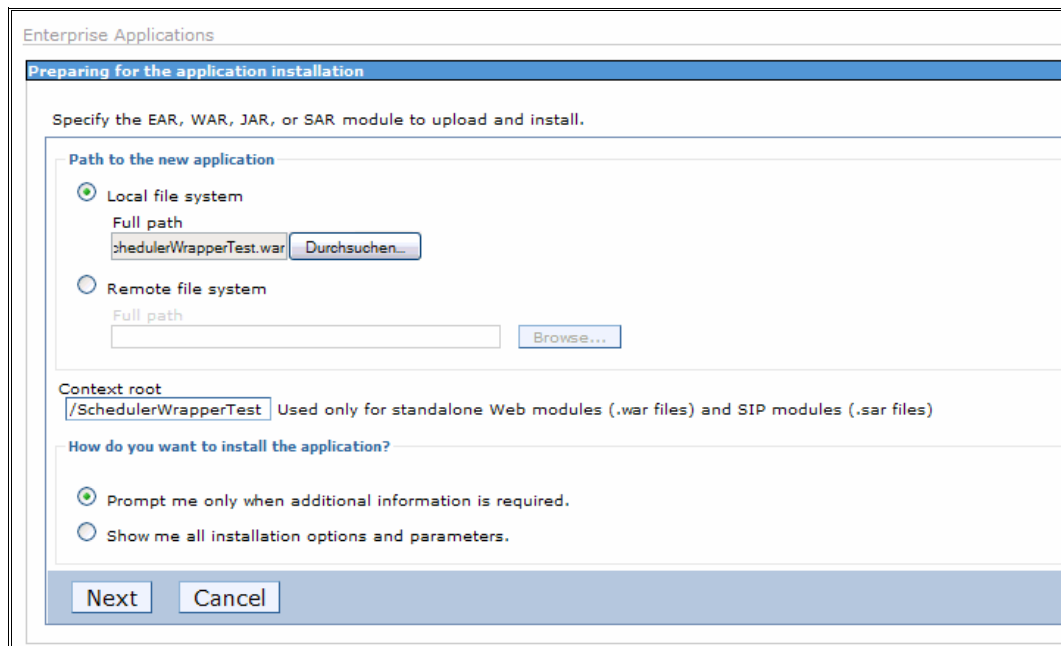


Figure 14-29 SchedulerWrapperTest.war - Path to application and context root

If you install the application in your local UTE, there is only one server where you can install the WAR file. If you want to install the application on a different WebSphere

environment configuration, select a server that hosts a Scheduler and not a Grid Execution Endpoint. For all the other installation options, select the default.

8. After you installed the application successfully, start the application. Now, you are ready to use the application where you can submit, cancel, and restart Echo jobs, or you can also get the job status or the job return code from submitted jobs.
9. Next, call the application by entering the following address in the browser. The servlet URL is:

`http://<hostname>:<port>/SchedulerWrapperTest/SchedulerWrapperServlet`

The servlet usage is simple. Use the servlet URL, and add the command that you want to set up as follows:

- `<servleturl>?command=submit` to submit an Echo job
- `<servleturl>?command=cancel&jobid=xxx` to cancel a job with a special job id
- `<servleturl>?command=getJobStatus&jobid=xxx` to get the job status
- `<servleturl>?command=getJobRC&jobid=xxx` to get the job return code

The Job Scheduler Web Services interface

The Job Scheduler for Web Services provides the following interfaces for driving a WebSphere XD Compute Grid job from a Web Services client program. Through this interface you can perform the following actions like provided in the JMC:

- ▶ `cancelJob(jobid)`
- ▶ `cancelRecurringRequest(request)`
- ▶ `getBatchJobRC(jobid)`
- ▶ `getJobDetails(jobid)`
- ▶ `getJobOutput(jobid)`
- ▶ `getJobStatus(jobid)`
- ▶ `modifyRecurringRequest(request)`
- ▶ `purgeJob(jobid)`
- ▶ `removeJobFromRepository(job)`
- ▶ `restartJob(jobid)`
- ▶ `resumeJob(jobid)`
- ▶ `saveJobToRepositoryAndSubmit(xJCL,job,replace)`
- ▶ `saveJobToRepository(xJCL,job,replace)`
- ▶ `showAllJobs()`
- ▶ `showAllRecurringRequests()`
- ▶ `showJobFromRepository(job)`
- ▶ `showRecurringJobs(request)`
- ▶ `submitJobFromRepository(job)`
- ▶ `submitJob(xJCL)`
- ▶ `submitRecurringRequest(job)`
- ▶ `suspendJob(jobid,seconds)`

Refer to the Job Scheduler Web Services Web Service Description Language (WSDL) file, as shown in Figure 14-30, for detailed descriptions of the Web services interface.

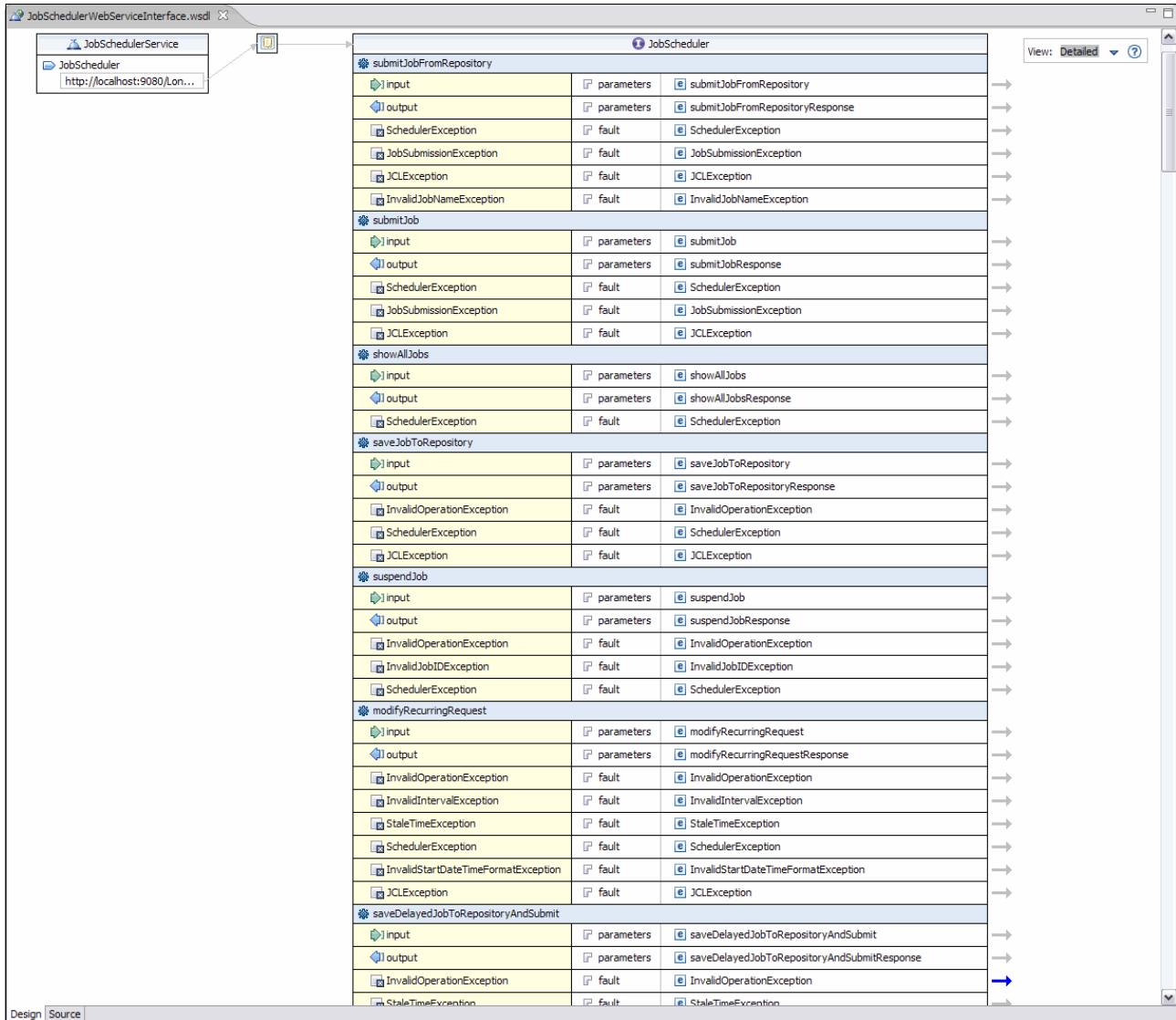


Figure 14-30 Job Scheduler Web Service interface

Tools for Web Services development

Tools are provided to assist with the following aspects of Web Services development:

- Discover Browse the UDDI Business Registries or WSIL documents to locate existing Web Services for integration.
- Create or Transform Create bottom-up Web Services from existing artifacts, such as Java beans and enterprise beans. Create top-down Web Services from WSDL discovered from others or created using the WSDL Editor.
- Build Wrap existing artifacts as SOAP accessible services and describe them in WSDL. The Web Services wizards assist you in generating a Java client proxy to Web Services described in WSDL and in generating Java bean skeletons from WSDL.
- Deploy Deploy Web Services into a variety of test environments.

Test	Test Web Services running locally or remotely in order to get instant feedback.
Develop	Generate sample applications to assist you in creating your own Web Services client application.
Publish	Publish Web Services to a UDDI v2 or v3 Business Registry, advertising your Web Services so that other businesses and clients can access them.

Note: There is no sample available at the moment, but to develop the code should work as using the EJB interface.

14.3 Exploiting enhanced features of Tivoli Workload Scheduler for z/OS

Tivoli Workload Scheduler for z/OS as part of the IBM Tivoli Dynamic Workload Automation portfolio provides mainframe batch and real-time workload scheduling. Tivoli Workload Scheduler on z/OS is a calendar- and event-based enterprise batch job scheduler for mainframe environments.

14.3.1 Strengths of Tivoli Workload Scheduler for z/OS

In the following sections, we provide an overview of the strengths of Tivoli Workload Scheduler:

- ▶ Calendar based and event based
- ▶ End-to-end solution
- ▶ Single point of control
- ▶ Scalability
- ▶ High availability
- ▶ Fault tolerance
- ▶ Dynamic scheduling: Integration with Workload Manager
- ▶ Triggers on System Console messages
- ▶ Automated job rerun
- ▶ Critical Path
- ▶ Exposing services into WEB API, Java EE REXX calls, and REXX-based interface
- ▶ Reporting
- ▶ Integrated file transfer

Calendar based and event based

Tivoli Workload Scheduler for z/OS is a leading scheduling solution in combining extensive calendar and event triggering services.

- ▶ Calendar services

Tivoli Workload Scheduler for z/OS builds the plans from your description of the production workload. It builds both high-level (long-term) plan and detailed (current) plans. You can produce trial plans to forecast future workloads (for example, to simulate the effects of changes to your production workload, calendar, and installation).

► **Event Triggering services**

Tivoli Workload Scheduler for z/OS enables unexpected work to be added automatically to the plan. On demand scheduling can be driven by jobs that arrive to JES, resources that change status to available, data sets that are revealed on the system.

End-to-end solution

Tivoli Dynamic Workload Automation components allow IT organizations to establish a virtual control point to build and automate a repeatable, scalable service execution process across the enterprise. It provides the ability to consolidate enterprise-wide batch and event-triggered workloads spanning multiple applications and systems, giving IT organizations efficient control and management of cross-enterprise workloads.

The *Tivoli Workload Automation* family components can be flexibly deployed into any solution-based environment. Customers can choose between mainframe-centric, distributed-centric or peer-to-peer parallel end-to-end enterprise scheduling solutions. Extensive awareness and interfacing is provided to SAP, PeopleSoft and Oracle business applications. Additionally, the end to end solution can be extended to include *Tivoli Dynamic Workload Broker agent*, which routes workloads to the best available resources on demand, based on user policies and IT resource availability. It also automatically identifies newly provisioned IT resources and incorporates them into the workload matching pool.

Single point of control

Tivoli Workload Scheduler for z/OS integrates with Tivoli Enterprise Portal. This integration enables you to monitor job status changes, alert conditions and critical paths, and also to combine Tivoli Workload Scheduler events with other application and system events. You will have a unified integrated interface, bringing up in a single window different view of resources in the enterprise. The integration is easily customizable and gives you the possibility to be alerted at any situation which is relevant in your environment, making it faster to proactively react to anomalous conditions.

Scalability

Tivoli Workload Scheduler for z/OS has a proven track record of scaling beyond 250,000 jobs a day in real customer production environments.

High availability

High availability for any Tivoli Dynamic Workload Automation solution is guaranteed by fail over mechanisms. Stand-by Controller, alternate workstation, backup and flip-flop plans are some of the features used to minimize the risk of outages in the z/OS production workload. Backup master, full status fault tolerant agents implement the fail over mechanism in end-to-end environment.

Fault tolerance

The end-to-end solution gives built-in fault tolerance, using the capability of the master domain manager to generate the plan and distribute it to the agents in the network. The plan contains a complete set of scheduling instructions, so that the agent can manage workload and resolve dependencies even in the event of a loss of communication with the master.

Dynamic scheduling: Integration with Workload Manager

Tivoli Workload Scheduler for z/OS integrates with IBM Workload Manager, a z/OS component, to provide the ability to dynamically route workloads to best available System z resources, based on resource availability and capacity. Integration with WLM is provided in two different ways:

- ▶ Using WLM Scheduling Environment (SE)

WLM SE is an attribute of Tivoli Workload Scheduler z/OS operations and WLM SE availability status is checked before jobs submission to avoid queuing jobs that cannot be executed. Jobs are automatically re-submitted at SE availability status change.

- ▶ Using WLM Service Class.

A Tivoli Workload Scheduler for z/OS operation which is on a Critical Path (see Critical Path section) and is running late can be automatically promoted to a better performance WLM Service Class. You can associate different WLM Service Classes to different operations, to flexibly accelerate or slow down your workload.

Triggers on System Console messages

Tivoli Workload Scheduler for z/OS integrates with Tivoli System Automation for z/OS and AF Operator to bilaterally facilitate automatic scheduling of workflows.

Integration with Tivoli System Automation for z/OS provides scheduling teams with complete visibility and control over automated System z resources from within the scheduling infrastructure, while, on the other side, allows to automatically trigger processes based on messages logged at System Console.

The same capability—automatic triggering of processes based on messages logged on System Console—is provided by Tivoli Workload Scheduler z/OS integration with AF Operator.

Automated job rerun

Tivoli Workload Scheduler for z/OS provides a Restart and Cleanup function to automatically restart any job in the plan (both failing and successfully) from any step, including:

- ▶ Capability to recover the data set or catalog configuration to the state previous to job running.
- ▶ Capability to restart the job at any step with optional adjustment of GDG generation numbers to match the ones generated in one of the previous job runs.
- ▶ Automatic determination of “restartable” steps, based on referenced data sets, and “best” step to restart the job from.
- ▶ Data set cleanup and data set protection.

Critical Path

Tivoli Workload Scheduler for z/OS provides a Critical Path feature to automatically manage SLAs for milestone jobs. Milestone jobs are jobs that are critical to the business and must not miss their deadlines. The user will flag Critical Jobs and Tivoli Workload Scheduler for z/OS will automatically calculate the Critical Path to those jobs and promote jobs on the Critical Paths which are late and might affect the Critical Jobs deadline. The promotion algorithm uses the WLM integration.

Exposing services into WEB API, Java EE REXX calls, and REXX-based interface

Open Java EE and Web Services interfacing allow IT organizations to consolidate custom applications and services into the service execution process. REXX-based programming interfaces are also available to flexibly communicate with Tivoli Workload Scheduler for z/OS.

Reporting

Advanced reporting capabilities are going to be provided this year as part of the Tivoli Dynamic Workload Console. Standard and user defined reports will enable to display product plan details, track and tune workload capacity during time, control the timely workload execution according to goals and SLA, and so forth.

Integrated file transfer

With WebSphere MQFDI and Tivoli Workload Scheduler for z/OS or distributed IBM provides an integrated file transfer capability.

14.3.2 Integrate Tivoli Workload Scheduler for z/OS with WebSphere XD Compute Grid

WebSphere XD Compute Grid is an execution run time for enterprise grid and java batch applications. It is not an enterprise scheduler. Enterprise schedulers serve as an integration point where the entire batch infrastructure is managed centrally. Artifacts such as enterprise-wide batch schedulers, dependencies among jobs and external resources, the location for where the job should execute, and so on are defined and managed at this point. WebSphere XD Compute Grid works alongside enterprise schedulers and is essentially one destination where enterprise schedulers can dispatch to. WebSphere XD Compute Grid's primary objectives are to execute batch jobs with high performance, recoverability, and availability.

On the z/OS platform, WebSphere XD Compute Grid integrates with JES and allows jobs to be submitted using JCL. Because enterprise schedulers are familiar with how to manage JES batch jobs, they by proxy are able to manage WebSphere XD Compute Grid jobs. Note that on z/OS, a native MQ client is used for job submission and monitoring, therefore the job submissions do not require the initialization and termination of a Java Virtual Machine, ensuring high performance. On distributed platforms, a Java-based adapter client bridges the gap, again by proxy, between the enterprise scheduler and Compute Grid.

Enterprise schedulers centrally manage operational plans and other such scheduler-specific artifacts. These schedulers then dispatch batch jobs to WebSphere XD Compute Grid using JES, on distributed platforms, the details differ but the concept remains the same. WebSphere XD Compute Grid then executes the job, assuring that the defined qualities of service are met, and notifies the enterprise scheduler of job state information and other such execution data.

WebSphere XD Compute Grid complements enterprise schedulers such as Tivoli Workload Scheduler, Control-M, Zeke, and so on, but it does not replace them.

WebSphere XD Compute Grid and enterprise schedulers (such as Tivoli Workload Scheduler) have the following differences:

- ▶ WebSphere XD Compute Grid provides:
 - Batch programming model
 - Batch development tools
 - Batch execution environment (called *batch container*)
 - Checkpoint or restart and failover capabilities
 - Job operations such as submit, cancel, stop, pause, and resume
 - Classification or WLM for batch jobs (requires object-oriented on distributed)
 - Job management console (plus command-line console and APIs)
 - Job logs (viewable or retrievable through JMC, command-line console, or APIs)
 - Job classes (rules for how much CPU, threads, and so forth that batch jobs can consume)
 - High availability or scalability for job scheduler and batch container
 - Support for both WebSphere Application Server-hosted batch applications and native applications
 - Integration with enterprise schedulers, so that the enterprise scheduler can be the superior job scheduler that dispatches jobs to the WebSphere XD Compute Grid job scheduler (called *meta-scheduling*), which enables WebSphere XD Compute Grid jobs to be part of a larger workload that is scheduled by the enterprise scheduler

Note: Enterprise schedulers have more powerful scheduling rules, such as time/date/resource (file) and does job choreography. WebSphere XD Compute Grid has only time/date scheduling and does not do job choreography. WebSphere XD Compute Grid does do multi-step jobs with conditional execution among the job steps based on return codes from the batch applications.

- ▶ Enterprise schedulers provide:
 - Ability to define workload schedule with dependencies
 - Ability to execute (JES jobs on z/OS or UNIX and bat commands on UNIX or Windows)
 - Ability to execute based on time or date (once or recurring) plus a trigger based on resource (that is file)
 - Ability to conditionally execute next item in the schedule based on result of previous thing in schedule
 - On z/OS, basic operations console to visualize the workload

Because an external scheduler does not know how to directly manage WebSphere XD Compute Grid jobs, a proxy model is used. The proxy model uses a regular JCL job to submit or monitor the WebSphere XD Compute Grid batch job. The JCL job step invokes a special program provided by the product, named WSGrid, an application which submits and monitors a batch job. WSGrid writes intermediary results of the job into the JCL job's joblog and does not return until the underlying job is complete, thus providing a synchronous execution model. Because the external scheduler can manage JCL jobs, it can manage a JCL job that invokes WSGrid. Using this pattern, the external scheduler can indirectly manage a job. An optional plug-in interface in the Job Scheduler enables a user to add code that updates the external scheduler operation plan to reflect the unique state of the underlying job, such as job started, step started, step ended, job ended. The WSGrid program is written with special recovery

processing so that if the JCL job is cancelled, the underlying job is cancelled also, thus ensuring synchronized life cycle of the two jobs.

The external scheduler interface uses the Java Message Service (JMS) as a bidirectional communication mechanism between an external client and the Job Scheduler. The interface uses platform messaging as the JMS provider. Integrating the Job Scheduler with an external workload scheduler is achieved by configuring and securing the Job Scheduler, enabling the interface by configuring the Job Scheduler message-driven interface, the service integration bus and the JMS queues, then using the WSGrid utility to run grid jobs.

WebSphere XD Compute Grid jobs can optionally be controlled through an external workload scheduler, such as Tivoli Workload Scheduler. Prepare for this integration by following these required steps in the WebSphere XD Compute Grid environment:

1. Configure the WebSphere Service Integration Bus (SIBus) and required JMS artifacts.
2. Install the JobSchedulerMDI application in the same server or cluster that is hosting the Job Scheduler.
3. Configure bus and JMS security if security is enabled in your environment.
4. Optionally implement and install the WSGridNotification SPI.
5. Use WSGrid utility as the executable launched by the external workload scheduler.

Figure 14-31 shows the job control by an external workload scheduler for the z/OS platform environment. In this diagram, Tivoli Workload Scheduler is shown as an example workload scheduler and shows the flow of scheduled jobs from Tivoli Workload Scheduler to WSGrid, through Job Scheduler to the Grid Execution Endpoint and from the Grid Execution Endpoint back to Tivoli Workload Scheduler z/OS using zConnector.

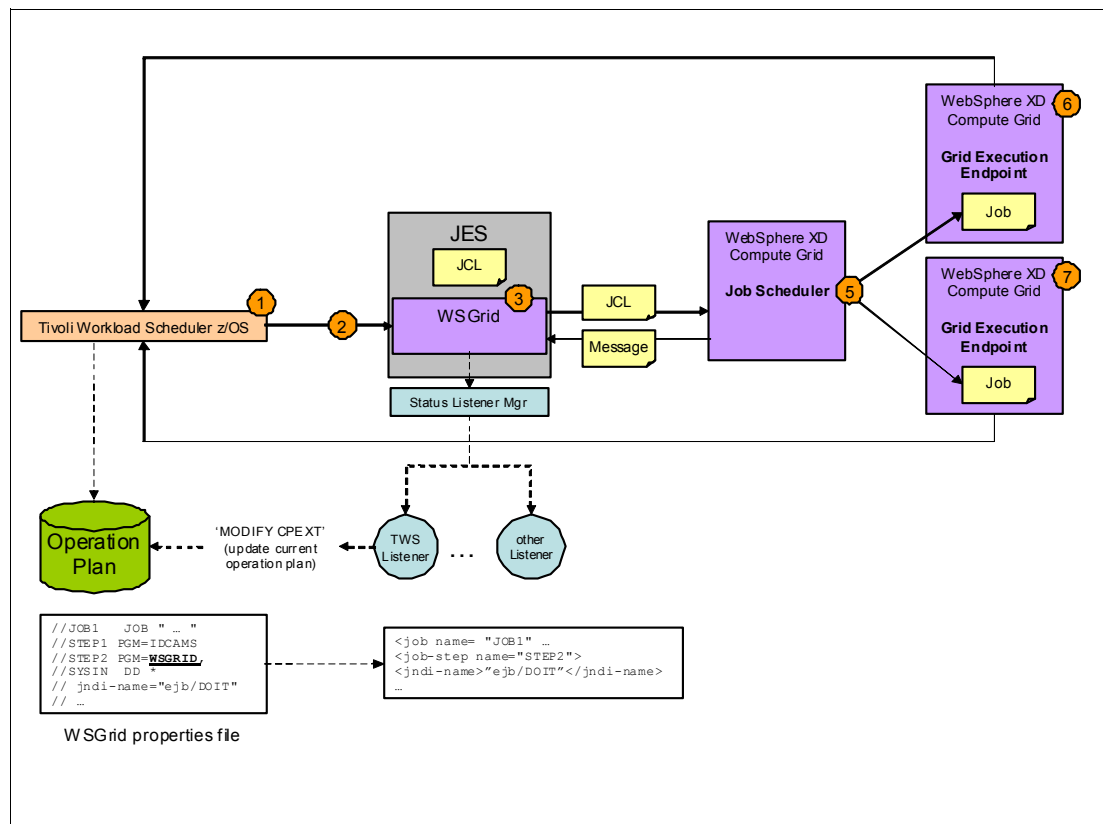


Figure 14-31 Flow of dynamic job scheduling with Tivoli Workload Scheduler for z/OS

The flow shown in the figure is as follows:

1. Tivoli Workload Scheduler determines which job to execute, for example based on operational plans.
2. Tivoli Workload Scheduler submits the corresponding JCL to JES.
3. The WSGrid connector is initialized (the submitted JCL contains a traditional `pgm=WSGRID` statement) and passes the xJCL as a parameter.
4. WSGrid submits the xJCL to the WebSphere XD Compute Grid Job Scheduler (JS).
5. The WebSphere XD Compute Grid Job Scheduler, using the z/OS workload manager (WLM), selects the best Grid Execution Endpoint (GEE) to which to dispatch the batch job.
6. Job logs, job execution status data, and the job and step return codes are transmitted to the WSGrid connector. The job executes, with its output directed to SYSOUT, like a standard MVS batch job.
7. Batch jobs executing within the Grid Execution Endpoint can dynamically submit new jobs through Tivoli Workload Scheduler z/OS.

Tivoli Workload Scheduler is already very often used to manage batch workloads on the z/OS platform. While Java batch executed inside a WebSphere environment is attractive, a way to control the WebSphere XD Compute Grid jobs through Tivoli Workload Scheduler is needed.

Figure 14-32 puts into perspective a WebSphere XD Compute Grid batch infrastructure, traditional batch infrastructure, and potential OLTP infrastructure where the workload is scheduled by Tivoli Workload Scheduler for z/OS.

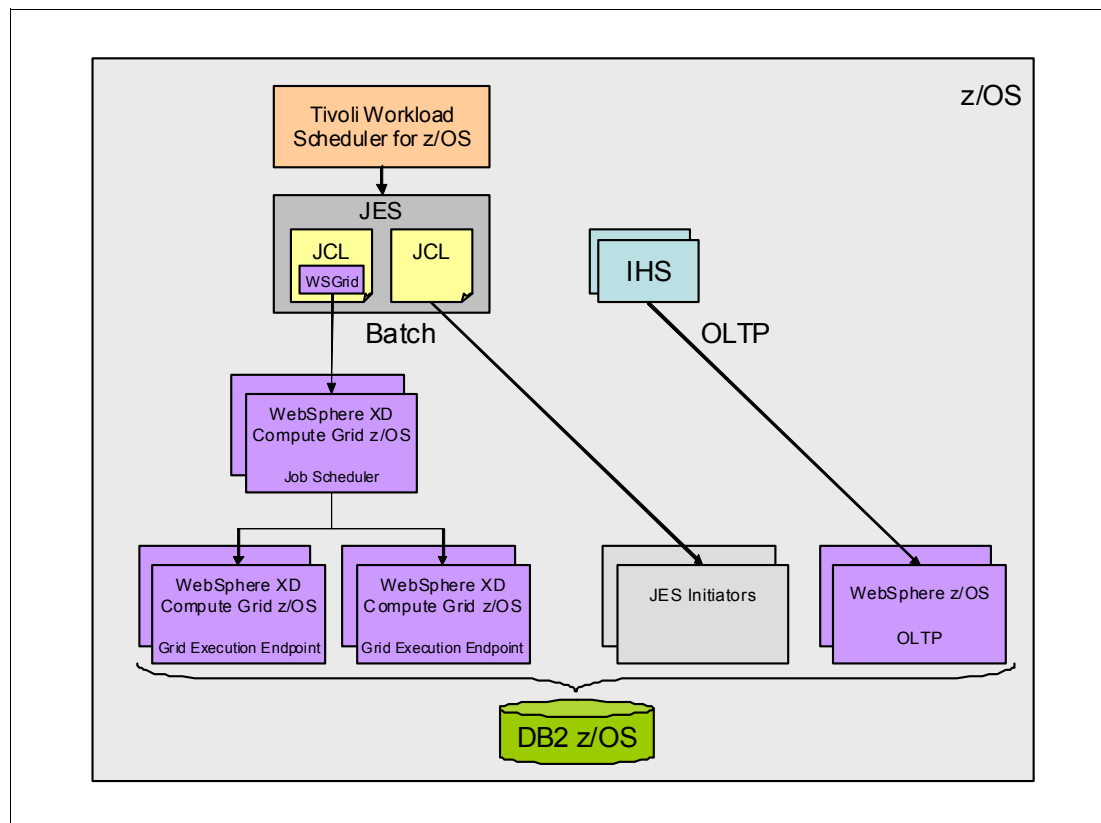


Figure 14-32 Scheduling traditional batch, WebSphere XD Compute Grid batch and OLTP workload with Tivoli Workload Scheduler z/OS

More information about the integration of WebSphere XD Compute Grid with external schedulers is provided in the specific Information Center:

- ▶ For Job Scheduler configuration, see:
<http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r1/index.jsp?topic=/com.ibm.websphere.gridmgr.doc/info/scheduler/tcgconf.html>
- ▶ For security information, see:
<http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r1/topic/com.ibm.websphere.gridmgr.doc/info/scheduler/tcgexsched.html>
- ▶ For information about how to run WebSphere XD Compute Grid jobs with the WSGrid utility, see:
<http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r1/index.jsp?topic=/com.ibm.websphere.gridmgr.doc/info/scheduler/ccgwsgrid.html>

14.4 Triggering a DB2 stored procedure

You can trigger DB2 stored procedures using any of the following techniques.

- ▶ SQL “CALL” statement
Every program (written in any language) that can connect to DB2 and supports calling stored procedures is suitable to trigger DB2 stored procedures using the SQL CALL statement.
- ▶ MQ message
In this case a MQ message would be written to a dedicated queue. If the stored procedure needs input parameters, they must be included in the input message. If DB2 MQListener is started, the input message is read and passed to the stored procedure. The stored procedure output message would then be stored by MQListener in the reply queue.
- ▶ Web Services
DB2 allows you to access your DB2 data and applications as Web Services. Also, by using DB2 User Defined Functions (UDF) you can call other Web Services (for example within a stored procedure or simply by using SQL statements).

In the following sections, we describe how to trigger stored procedures by sending an MQ message or calling a Web Service.

14.4.1 Using WebSphere MQ to trigger a DB2 stored procedure

DB2 for z/OS provides an asynchronous listener, *MQListener*. MQListener is a framework for tasks that read from WebSphere MQ queues and call DB2 stored procedures with messages as those messages arrive.

MQListener combines messaging with database operations. You can configure the MQListener daemon to listen to the WebSphere MQ message queues that you specify in a configuration database. MQListener reads the messages that arrive from the queue and calls DB2 stored procedures using the messages as input parameters. If the message requires a reply, MQListener creates a reply from the output that is generated by the stored procedure. The message retrieval order is fixed at the highest priority first, and then within each priority the first message received is the first message served.

MQListener runs as a single multi-threaded process under z/OS UNIX System Services. Each thread or task establishes a connection to its configured message queue for input. Each task also connects to a DB2 database on which to run the stored procedure. The information about the queue and the stored procedure is stored in a table in the configuration database. The combination of the queue and the stored procedure is a task.

MQListener tasks are grouped together into named configurations. By default, the configuration name is empty. If you do not specify the name of a configuration for a task, MQListener uses the configuration with an empty name.

There is support for both one-phase and two-phase commit environments. A one-phase commit environment is where DB and MQ interactions are independent. A two-phase commit environment is where DB and MQ interactions are combined in a single unit of work.

The MQListener UNIX System Services command `db2mq1n1` is the name of the executable for one phase and `db2mq1n2` is the name of the executable for two phase.

Configuring MQListener system environment

To trigger a stored procedure from MQ, you first have to configure your WebSphere MQ and DB2 environment to work together. Therefore, you must go through some installation tasks (such as creating queues, creating the DB2 table `SYSMQL.LISTENERS`, and so forth). The systems programming activities to configure the environment are beyond the scope of this book.

After the environment is configured, some new message queues are created in the system. These queues are used internally by MQListener. In our test system, we created the following queues:

ADMIN_Q	Admin queue
BACKOUT_Q	Backout queue
IN_Q	Input queue having a backout queue with threshold=3
REPLY_Q	Output queue or reply queue
DEADLETTER_Q	Dead letter queue

Creating the DB2 stored procedure to process MQ message

To test the WebSphere MQ trigger mechanism we use the same DB2 Java stored procedure `BATCH.JAVASTP` that we developed in 6.3, “Java in DB2 for z/OS” on page 64. This stored procedure generates PDF files based on XML input data and stores these files in DB2, a UNIX System Services file system or both. Therefore, if the `BATCH.JAVASTP` procedure is missing in your environment, first create this procedure as described in 6.3, “Java in DB2 for z/OS” on page 64.

In this section, we describe only the necessary changes to start the batch stored procedure additionally from WebSphere MQ.

Note: You can download all the necessary material to create this example (such as the Eclipse workspace, DDL, and so forth) as described in Appendix C, “Additional material” on page 453.

Java stored procedure

DB2 MQListener reads messages from WebSphere MQ queues and calls DB2 stored procedures. When the stored procedure completes its work, MQListener reads the output message from the stored procedure and writes this message to the configured reply queue.

Therefore, the stored procedure that MQListener calls must have the following two parameters:

- ▶ Input parameter for incoming message in character format
- ▶ Output parameter for outgoing message in character format

The Java class to generate PDF files needs two different parameters, but the incoming message is only a single character string. Further, we want to characterize the current running process with a *jobID*. Adding a jobID is necessary if you want to process many incoming messages. The output message should contain this *jobID*, so you can see which jobs have finished.

Therefore, we need create an incoming message with these parameters separated by the colon (:) in one single character string as shown in Example 14-14.

Example 14-14 Sample incoming message

```
ReportJob01:store_fs_db2:/u/wagnera/javastp/pdfs
```

For the Java stored procedure, you need to create a method with the following features and capabilities:

- ▶ Contain one input and one output parameter
- ▶ Split the incoming message into three different parameters
- ▶ Call the Java class to generate PDF files
- ▶ Define the output message

Example 14-15 shows the content of method `triggeredByMQ`, which is called by DB2 when the stored procedure is called by MQListener. You can find this source code in “Triggering Java Stored procedure to generate PDF files” on page 450.

Example 14-15 Method triggeredByMQ from Java Class GenPdf

```
public static void triggeredByMQ(String inMsg, String[] outMsg)
    throws SQLException {

    String temp = inMsg;
    String jobID = temp.substring(0, temp.indexOf(":"));
    temp = temp.substring(temp.indexOf(":") + 1);
    String action = temp.substring(0, temp.indexOf(":"));
    String pdfDir = temp.substring(temp.indexOf(":") + 1);

    PdfCreator pdfc = new PdfCreator();
    pdfc.generatePDF(action, pdfDir);

    outMsg[0] = new String("PDF(s) fuer JobID '"
        + jobID + "' successfully created.");
}

```

Finally, build and copy the JAR file with the Ant script by right-clicking **deploy.xml** and selecting **Run As** → **Ant Build** as described in “Deploying the Java files to the UNIX System Services file system” on page 72.

Defining the stored procedure in DB2

Next, you need to redefine the new Java stored procedure that MQListener calls, because in this case, the `triggeredByMQ` method must be issued (instead of `runGenerate`) and this stored procedure has different input and output parameter as `BATCH.JAVASTP`. Therefore, you need

to create a new procedure called BATCH.MQ_TRIGGERED_STP in DB2 as shown in Example 14-16.

Example 14-16 Define Java stored procedure for MQListener

```
CREATE PROCEDURE BATCH.MQ_TRIGGERED_STP
  (IN IN_MSG VARCHAR(115)
  ,OUT OUT_MSG VARCHAR(100))
EXTERNAL NAME 'com.ibm.itso.sample.GenPdf.triggeredByMQ'
LANGUAGE JAVA
PARAMETER STYLE JAVA
NOT DETERMINISTIC
FENCED
CALLED ON NULL INPUT
MODIFIES SQL DATA
NO DBINFO
NO COLLID
WLM ENVIRONMENT D9GGWLMJ
ASUTIME NO LIMIT
STAY RESIDENT YES
PROGRAM TYPE SUB
SECURITY DB2
INHERIT SPECIAL REGISTERS
STOP AFTER SYSTEM DEFAULT FAILURES
COMMIT ON RETURN NO ;

GRANT EXECUTE ON PROCEDURE BATCH.MQ_TRIGGERED_STP TO PUBLIC;
```

After changing the Java code, do *not* to refresh the current WLM application environment by running the RefreshWLM program, which is in the Eclipse UtilSTP project, as described in “Refreshing the WLM Application Environment” on page 74.

Testing the stored procedure locally

Before using MQListener to start the Java stored procedure, test this stored procedure. Change the TestSTP Java program in the UtilSTP program as shown in Example 14-17. The first parameter for the stored procedure contains one character string with three internal parameters, separated by a colon (:). You can find this source code in “Triggering Java Stored procedure to generate PDF files” on page 450.

Example 14-17 Testing stored procedure “MQ_TRIGGERED_STP” locally

```
String inMsg = "TestLocal:store_fs_db2:/u/wagnera/javastp/pdfs";

System.out.println("Call STP.....");
CallableStatement cs = con.prepareCall("CALL BATCH.MQ_TRIGGERED_STP(?, ?)");
cs.setString(1, inMsg);
cs.registerOutParameter(2, java.sql.Types.VARCHAR);
cs.execute();
System.out.println("Call STP ended: " + cs.getString(2));

cs.close();
```

Configuring MQListener tasks

As part of configuring MQListener in DB2 for z/OS, you must configure at least one MQListener task. These tasks are UNIX System Services processes that listen on assigned

queues and start the configured stored procedure for this queue to process the incoming message. Normally the system programmer is responsible for configuration and starting MQListener tasks. Therefore, in this book we describe only the necessary steps to configure and start a MQListener.

Use the MQListener command, **db2mq1n1** or **db2mq1n2**, to configure and start MQListener tasks. Issue the command from the z/OS UNIX System Services command line in any directory. Alternatively, you can put the command in a file, grant execute permission, and use the BPXBATCH utility to invoke the script from JCL.

To configure and start DB2 MQListener tasks, set up the UNIX System Services environment for DB2 and MQ connectivity. Edit the `.profile` file in your UNIX System Services home directory and append the lines as shown in Example 14-18). The values for LIBPATH, PATH, and STEPLIB are system specific. You need to change them to fit your environment.

Example 14-18 Append data to UNIX System Services user profile .profile

```
#MQLISTENER
export MQLNHOME=/usr/lpp/db2/d9gg/db2910_mq1/listener
export PATH=$PATH:$MQLNHOME/bin
export LIBPATH=$LIBPATH:$MQLNHOME/lib
export STEPLIB=$STEPLIB:DB9G9.SDSNEXIT:DB9G9.SDSNLOAD:DB9G9.SDSNLOD2
export STEPLIB=$STEPLIB:MQ700.SCSQLOAD:MQ700.SCSQAUTH
```

Example 14-19 shows the command to add a new MQListener configuration. The add parameter with the **db2mq1n1** or **db2mq1n2** command updates a row in the DB2 table, SYSMQL.LISTENERS. To get help with the command and the valid parameters, issue **db2mq1n1/db2mq1n2 help <command>**.

Example 14-19 Add new MQListener

```
db2mq1n1/db2mq1n2 add
  -ssID <subsystem name>
  -config <configuration name>
  -queueManager <queuemanager name>
  -inputQueue <inputqueue name>
  -procName <stored-procedure name>
  -procSchema <stored-procedure schema name>
  -numInstances <number of instances>
```

To start the MQListener task, issue the command shown in Example 14-20.

Example 14-20 Start MQListener

```
db2mq1n1/db2mq1n2 run
  -ssID <subsystem name>
  -config <configuration name>
  -adminQueue <adminqueue name>
  -adminQMgr <adminqueuemanager name>
```

To stop or restart a MQListener task, issue the command shown in Example 14-21.

Example 14-21 Stop/Restart MQListener

```
db2mq1n1/db2mq1n2 admin
  -adminQueue <adminqueue name>
  -adminQMgr <adminqueuemanager name>
```

-adminCommand shutdown/restart

To remove messaging tasks from your environment, issue the command shown in Example 14-22.

Example 14-22 Remove task from environment

```
db2mq1n1/db2mq1n2 remove
  -ssID <subsystem name>
  -config <configuration name>
  -queueManager <queuemanager name>
  -inputQueue <inputqueue name>
```

To display information about the configuration, issue the command shown in Example 14-23.

Example 14-23 Display information about configured tasks

```
db2mq1n1/db2mq1n2 show
  -ssID <subsystem name>
  -config <configuration name> | all
```

Now, you can start the MQListener tasks. In our example, we use the following configuration:

DB2 system	D9G1
Queue Manager	MQG1
Queue for input messages	WAGNERA.DB2.REQUEST
Queue for output messages	WAGNERA.DB2.REPLY
Queue for failing messages	WAGNERA.DB2.BACKOUT
DB2 stored procedure	BATCH.MQ_TRIGGERED_STP

We define our MQListener task with the UNIX System Services command shown in Example 14-24.

Example 14-24 Define MQListener task

```
db2mq1n1 add -ssid D9G1 -inputQueue WAGNERA.DB2.REQUEST -queueManager MQG1
-procSchema BATCH -procName MQ_TRIGGERED_STP
```

As mentioned previously, when configuring the MQListener system environment, we created an admin queue called ADMIN_Q. You need this information to start, stop, and restart the MQListener task. In this case, we start the MQListener task with the UNIX System Services command shown in Example 14-25.

Example 14-25 Start MQListener task

```
db2mq1n1 run -ssID D9G1 -adminQueue ADMIN_Q -adminQMGr MQG1
```

For every message that is stored in the input queue, this MQListener task performs the following tasks:

1. Reads the message.
2. Calls the configured DB2 stored procedure BATCH.MQ_TRIGGERED_STP by sending this message as the first (IN) parameter of our stored procedure.
3. Store the output message from the stored procedure, the second (OUT) parameter of the stored procedure, in the replyQueue.

You can stop the MQListener task with the UNIX System Services command shown in Example 14-26.

Example 14-26 Stop MQListener task

```
db2mq1n1 admin -adminQueue ADMIN_Q -adminQMGr MQG1 -adminCommand shutdown
```

Binding the JCC driver packages

Before you trigger a Java stored procedure with WebSphere MQ, you must bind the necessary JDBC driver packages.

Because our stored procedure is written in Java, we use DB2 JCC driver to communicate with the database. This driver needs packages that must be bound to the system. Because the MQListener program uses a special DB2 collection, you must bind JCC packages into this collection. In our test setup, MQListener is using collection *LSNR*.

To bind the JCC packages, you can use class `com.ibm.db2.jcc.DB2Binder` that is included in the DB2 driver file `db2jcc.jar`. One method to bind the JCC packages is to create a shell script in the UNIX System Services home directory containing the necessary commands and start the script. Example 14-27 shows the content of this shell script. The values for `JDBC_Path`, `Url`, `Uid`, `Pw`, and `Col` are system specific. You must change them to fit your environment.

Example 14-27 Binding DB2 JCC packages

```
JDBC_Path=/usr/lpp/db2/d9gg/db2910_jdbc
Url=jdbc:db2://<server>:<port>/<DB2Location>
Uid=<User>
Pw=<Pw>
Col=LSNR

export CLASSPATH=$CLASSPATH:$JDBC_Path/classes/db2jcc.jar
export CLASSPATH=$CLASSPATH:$JDBC_Path/classes/db2jcc_license_cisuz.jar

java com.ibm.db2.jcc.DB2Binder -url $Url -user $Uid -password $Pw -collection $Col
```

Testing the application to send MQ message

To test DB2 MQListener, you need to send a message to the input queue that was used when you defined the MQListener task, (In our test system, this is *WAGNERA.DB2.REQUEST*.)

To test MQListener easily, we created a sample J2EE application, as shown in Figure 14-33. A simple servlet acts as a Web interface to the user. We used this same generic application for the sample in 14.2, “Using WebSphere XD Compute Grid trigger mechanisms” on page 234.

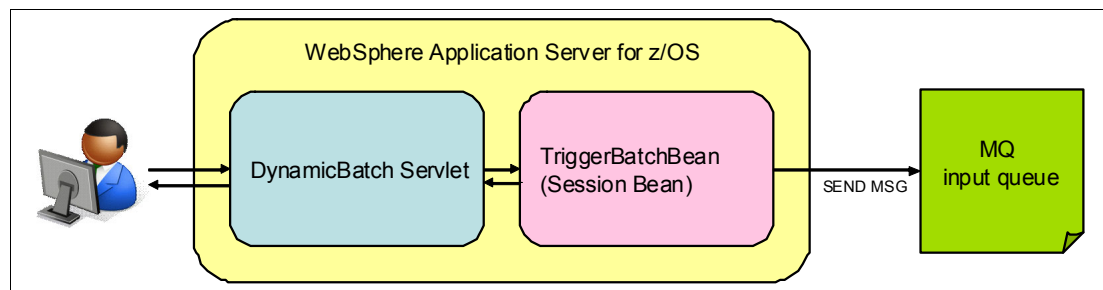
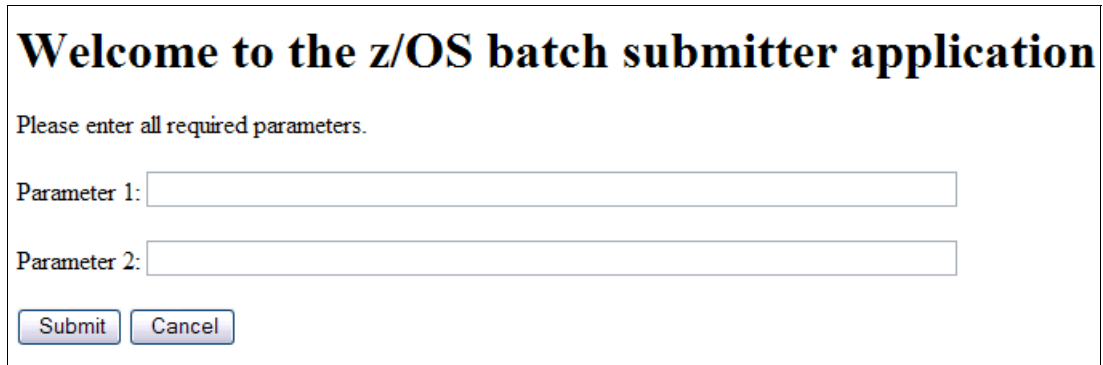


Figure 14-33 Dynamic batch sample application for use with MQListener

Figure 14-34 shows this servlet looks from a user perspective. It consists of a form with two parameter input fields as well as a Submit and Cancel button. By clicking **Submit**, the servlet calls a Session Bean called `TriggerBatchBean`. This bean contains the logic to check input parameter and send a message to queue `WAGNERA.DB2.REQUEST`.



Welcome to the z/OS batch submitter application

Please enter all required parameters.

Parameter 1:

Parameter 2:

Figure 14-34 Sample application Welcome page

Note 1: In the template version of the application, the session bean returns only the value of the two parameters.

Note 2: In our scenario, we use WebSphere Application Server for z/OS Version 6.1 to ensure compatibility with the J2EE 1.4 specification. Although we could have used Java EE 5.0, we wanted to ensure that a broad range of users can use this sample.

The servlet and session bean source code can be found under “Dynamic batch Web application” on page 445.

We also provide this application as a Rational Application Developer project. See Appendix C, “Additional material” on page 453 for more details. However, if you do not have Rational Application Developer, you could also use some other kind of J2EE development tool.

Changing the template session bean

Because the template session bean returns only the value of the input parameter, you need to change the code. You can download the modified EJB project as described in Appendix C, “Additional material” on page 453. To change the code:

1. First, verify that only one parameter is allowed, as shown in Example 14-28.

Example 14-28 Only one parameter allowed

```
// only one parameter allowed
try {
    if (parameters[0].trim() == "") {
        message = "Error: parameter 1 must be set";
        return message;
    }
    if (parameters[1].trim() != "") {
        message = "Error: parameter 2 not allowed";
        return message;
    }
} catch (Exception e) {
    message = "Error: wrong input parameter!";
}
```

```

    return message;
}

```

- Next, verify the value of the input parameter. The Java stored procedure needs the following three input parameters, separated by a colon (:):

JobID	To classify our current job
action	Possible values are
store_fs	Save PDF files only in UNIX System Services file system
store_db2	Save PDF files only in DB2 BLOB column
store_fs_db2	PDF files in UNIX System Services file system and in DB2 BLOB column
pdfDir	UNIX System Services directory to store generated PDF files

Example 14-29 shows the relevant code.

Example 14-29 Check input parameter value

```

// Check input parameter
String temp = parameters[0];
String jobID = "";
String action = "";
String pdfDir = "";
try {
    jobID = temp.substring(0, temp.indexOf(":"));
    temp = temp.substring(temp.indexOf(":") + 1);
    action = temp.substring(0, temp.indexOf(":"));
    pdfDir = temp.substring(temp.indexOf(":") + 1);
} catch (Exception e) {
    message = "Error: wrong input parameter '"
        + parameters[0]
        + "', must be
'DB2StoredProcPara1:DB2StoredProcPara2:DB2StoredProcPara3'!";
    return message;
}

// check value of parameter "action"
if (!action.equalsIgnoreCase("store_fs")
    && !action.equalsIgnoreCase("store_fs_db2")
    && !action.equalsIgnoreCase("store_db2")) {
    message = "Error: wrong input parameter for 'DB2StoredProcPara2',"
        + " must be one of 'store_fs, store_db2, store_fs_db2'!";
    return message;
}

```

- Send the value of input parameter as a MQ message to the input queue, as shown in Example 14-30.

Example 14-30 Send MQ message

```

QueueConnectionFactory conFactory = null;
QueueConnection conn = null;
QueueSession session = null;
QueueSender sender = null;
InitialContext jndi;
try {
    jndi = new InitialContext();

```

```

conFactory = (QueueConnectionFactory) jndi.lookup("jms/ConFactory");
conn = (QueueConnection) conFactory.createConnection();
session = conn.createQueueSession(false,
    QueueSession.AUTO_ACKNOWLEDGE);

Queue queue = (Queue) jndi.lookup("jms/DB2Request");
sender = session.createSender(queue);

TextMessage msg = session.createTextMessage(parameters[0]);
sender.send(msg);
} catch (NamingException e) {
    message = "Error: " + e.getMessage();
    e.printStackTrace();
} catch (JMSException e) {
    message = "Error: " + e.getMessage();
    e.printStackTrace();
}
}

```

4. Finally, inform the user that the batch job was submitted successfully, as shown in Example 14-31.

Example 14-31 Return info message to Web application

```

message = "Job '" + jobID + "' successfully submitted to create "
    + "PDF files in USS directory '" + pdfDir + "'.";
return message;

```

Because we send only a message to the queue, we also must inform the user when the job completes. This example is a small example to show how to trigger a stored procedure by sending a message to a queue. In a production environment, you also must implement a mechanism to inform the user when the job completes.

14.4.2 DB2 as a Web service provider

For each type of architecture, DB2 for z/OS offers a robust solution for Web applications.

Note: You can find more Information about DB2 as a Web service provider in *Enabling z/OS Applications for SOA*, SG24-7669.

Specifically, using DB2 for z/OS as a database server for a Web application provides the following advantages:

- ▶ Exceptional scalability

The volume of transactions on any Web application varies. Transaction loads can increase, or spike, at different times of the day, on different days of the month, or at different times of the year. Transaction loads also tend to increase over time. In a Parallel Sysplex environment, DB2 for z/OS can handle the full range of transaction loads with little or no impact on performance. Any individual user is generally unaware of how busy the system is at a given point in time.

- ▶ High degree of availability

When DB2 for z/OS runs in a Parallel Sysplex environment, the availability of data and applications is very high. If one DB2 subsystem is unavailable, for example, because of maintenance, other DB2 subsystems in the Sysplex take over the workload. Users are

unaware that part of the system is unavailable because they have access to the data and applications that they need.

- ▶ Ability to manage a mixed workload
DB2 for z/OS effectively and efficiently manages priorities of a mixed workload as a result of its tight integration with z/OS Workload Manager.
- ▶ Protection of data integrity
Users of DB2 for z/OS can benefit from the product's well-known strength in the areas of security and reliability.

To run DB2 as a Web service provider you need a WebSphere Application Server. The WebSphere Application Server can be on z/OS or a distributed platform.

While DB2 can act as a Web service requester, DB2 made a conscious decision to reuse the Web Services infrastructure that works well that is provided with products, such as WebSphere Application Server, WebSphere Message Broker, WebSphere DataPower®, and so on, and focused on the Web service to DB2 mapping instead of reimplementing everything from scratch.

DB2 development made a conscious decision to not start over but to use the existing Web services infrastructure provided by application servers.

Using Data Web Services

The easiest way to expose DB2 data and business logic as a Web service is using the Data Web Services (DWS) feature of the IBM Data Studio tool.

In this section, we use the following tools:

- ▶ IBM Data Studio, Version 2.2 for development
- ▶ WebSphere Application Server for z/OS as the Web service hosting environment

Note: For more information about Data Studio, refer to:

<http://www.ibm.com/software/data/studio/>

Data Web Services (DWS) is a new solution to significantly ease the development, deployment, and management of Web Services-based access to DB2. DWS lets you take Data Manipulation Language (DML) statements (such as SELECT, INSERT, UPDATE, DELETE, XQuery, and, last but not least, stored procedure calls) and generate Web Services from these statements without writing a single line of code. The generated Web Services are packaged as a ready-to-deploy Web application (as a WAR file), which can easily be deployed to supported application servers.

Highlights of DWS include:

- ▶ Creating Web Services using DWS requires no programming.
DWS lets you create Web Services using a drag-and-drop interface.
- ▶ DWS supports SOAP over HTTP and Web Services Description Language (WSDL) generation.
DWS automatically generates a WSDL file that contains a description of the Web service.
- ▶ No code generation.
DWS consists of a common metadata-driven run time, and there is no “black box” code that gets generated “under the covers,” which results in a reliable and lightweight application.

Developing a stored procedure as a Web service

In this example, we show how to create a simple service that calls a stored procedure. We will create a stored procedure which generates PDF files based on XML input data and store these files in DB2, UNIX System Services file system or both. In 6.3, “Java in DB2 for z/OS” on page 64, we describe how to create the Java stored procedure BATCH.JAVASTP using native Eclipse IDE. In this section, we use Data Studio for development. Thus, we focus on the Web Services part and describe only the necessary steps to create the stored procedure.

Note: You can download all the necessary material to create this example (such as Data Studio workspace, DDL, and so forth) as described in Appendix C, “Additional material” on page 453.

Creating the Java stored procedure

To create and test the Web Service easily, first create a simple Java stored procedure. This procedure must have the same input and output parameters as the procedure to generate PDF files. This procedure is necessary because this information is relevant for the Web Services Description Language (WSDL) definition. To create the procedure:

1. After installing and starting Data Studio, close the Welcome window. Then, create a new Development project by clicking **File** → **New** → **Data Development Project**. Our project is named *Redbook*. After clicking **Next**, you need to create a connection to the database.

Figure 14-35 shows how to create the DB2 for z/OS connection. Define the following options:

- Select a database manager: DB2 for z/OS
- Location: DB2 location
- Host: Name or IP address of your DB2 server
- Port number: Port number of DB2 server
- User name: User for database access
- Password: Password for database access

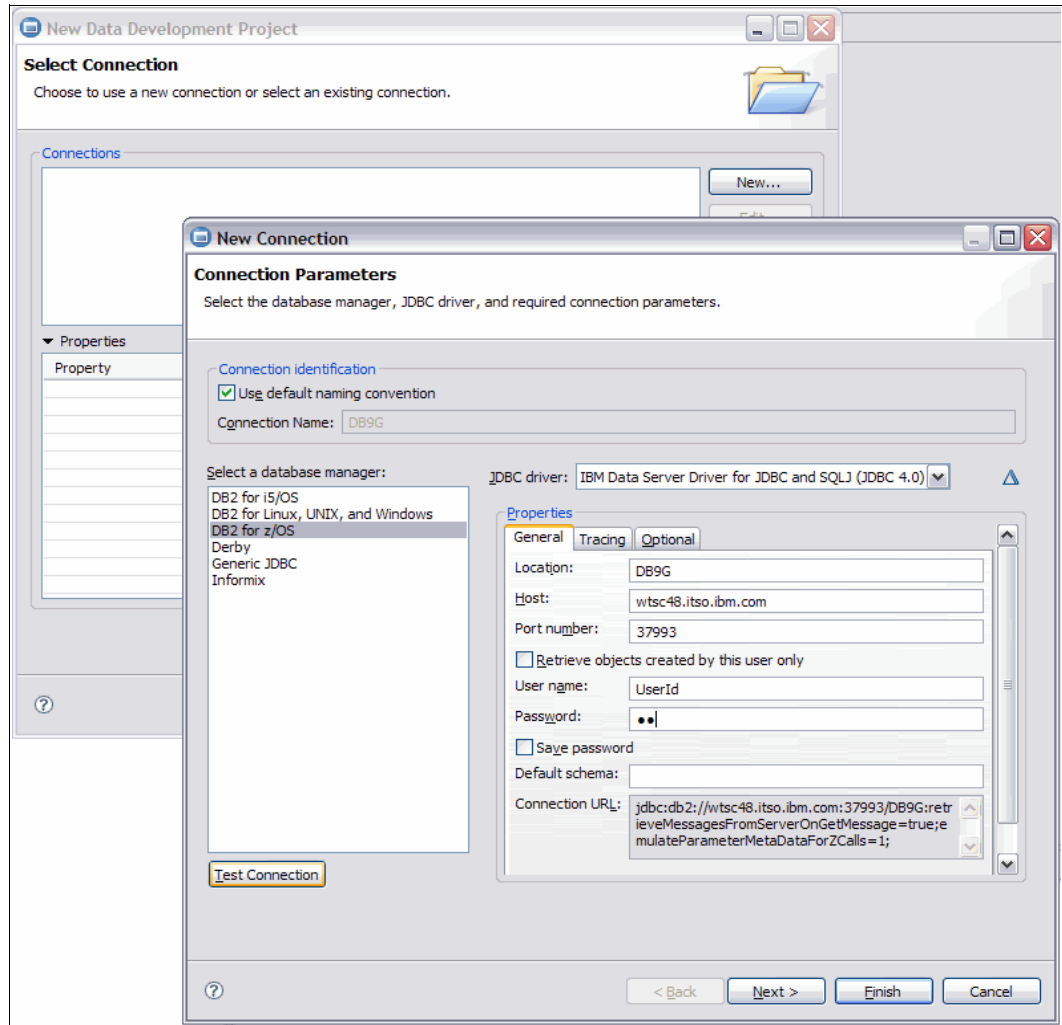


Figure 14-35 Create database connection

You can test the configuration by clicking **Test Connection**. If the connection is configured properly, you can click **Finish**.

- Next, select the configured Connection, and click **Finish** (see Figure 14-36).

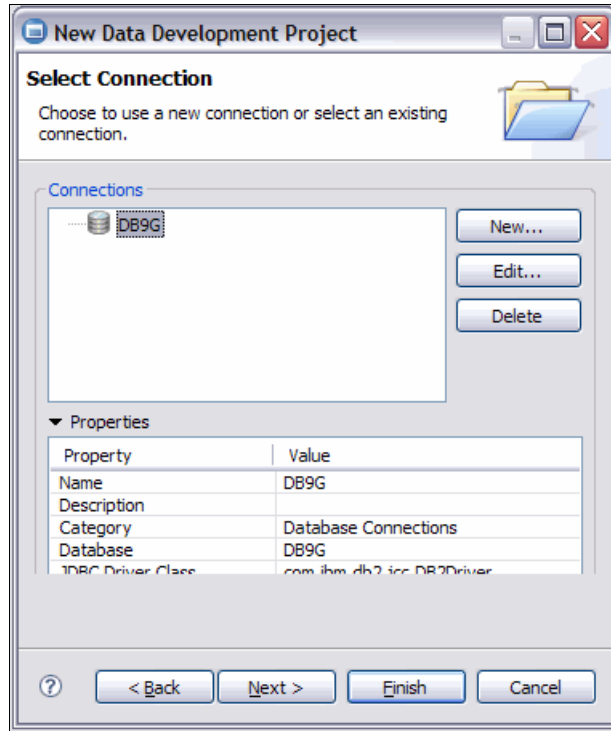


Figure 14-36 Configuring connection

- To create the stored procedure, expand the Redbook project in the Data Project Explorer window, select **Stored Procedures**, and right-click **New** → **Stored Procedure** (see Figure 14-37).

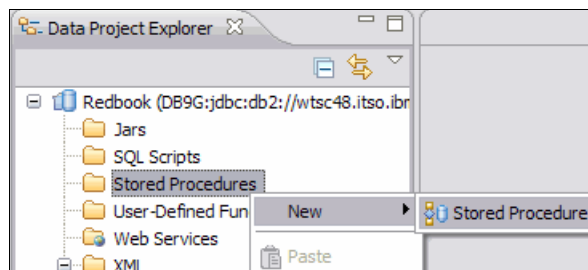


Figure 14-37 Create stored procedure

4. In the next window set the following options, as shown in Figure 14-38:

- Name: BATCH.SOAP_TRIGGERED_STP
- Language: Java
- Java package: com.ibm.itso.sample
- Dynamic SQL using JDBC

Click **Next**.

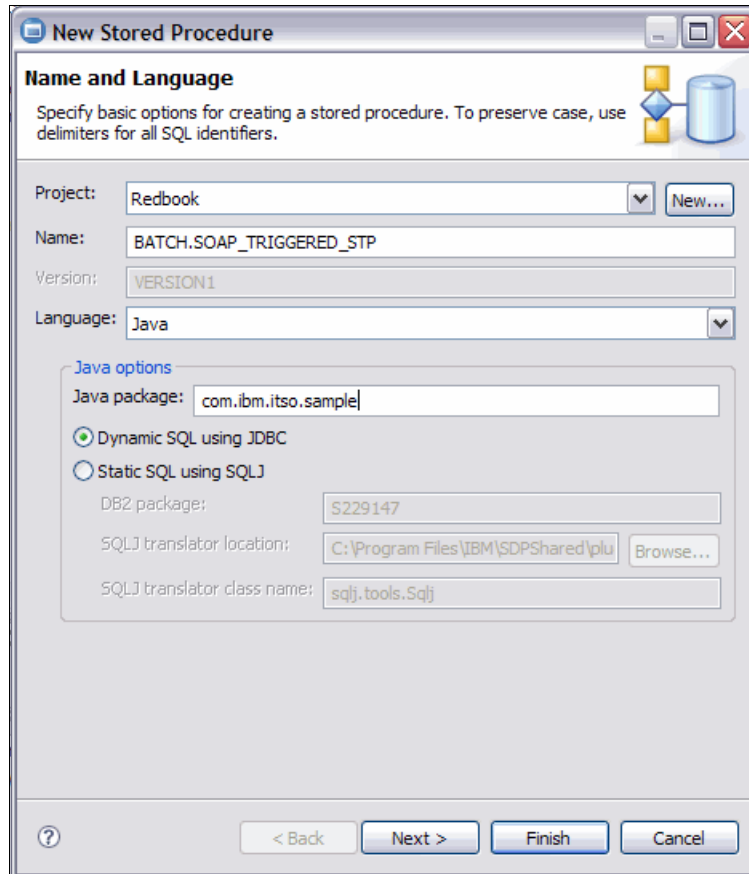


Figure 14-38 Create Java stored procedure

- Then, remove the predefined Statement, and choose **None** for the “Result set” option as shown in Figure 14-39. Click **Next**.

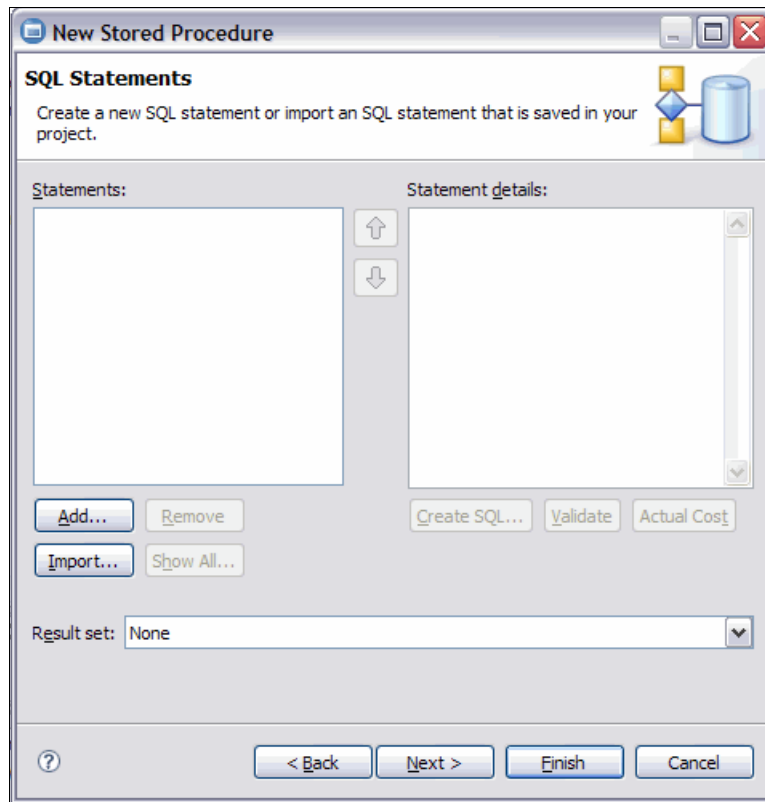


Figure 14-39 Removing pre defined SQL statement

6. Now, you need to define the input and output parameters. As mentioned previously, the test procedure must have the same input and output parameters as the procedure to generate PDF files. You need to create the following parameters:

- IN parameter ACTION - DB2 data type VARCHAR(15)
- IN parameter PDF_DIR - DB2 data type VARCHAR(100)
- OUT parameter OUT_MSG - DB2 data type VARCHAR(100)

In the current window, click **Add** to define the first parameter, as shown in Figure 14-40.

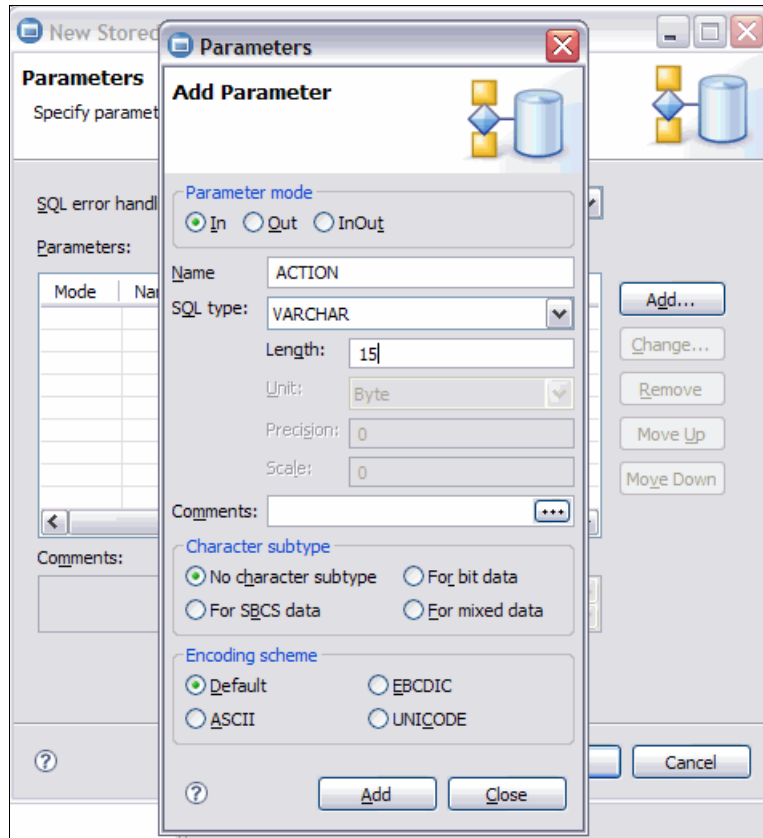


Figure 14-40 Define first input parameter

- Click **Add** to add this parameter the list of parameters.
- You can use the current window to add another parameter. Change Name from *VAR01* to *PDF_DIR* and change “Length” from *15* to *100*. Then, click **Add**.
- To add the third parameter change “Name” from *VAR01* to *OUT_MSG* and change “Parameter mode” from *IN* to *OUT*. Then, click **Add**.
- Now that all the necessary parameters are defined, click **Close** to close this window.

- Click **Next** and then click **Advanced** to define the WLM environment (see Figure 14-41).

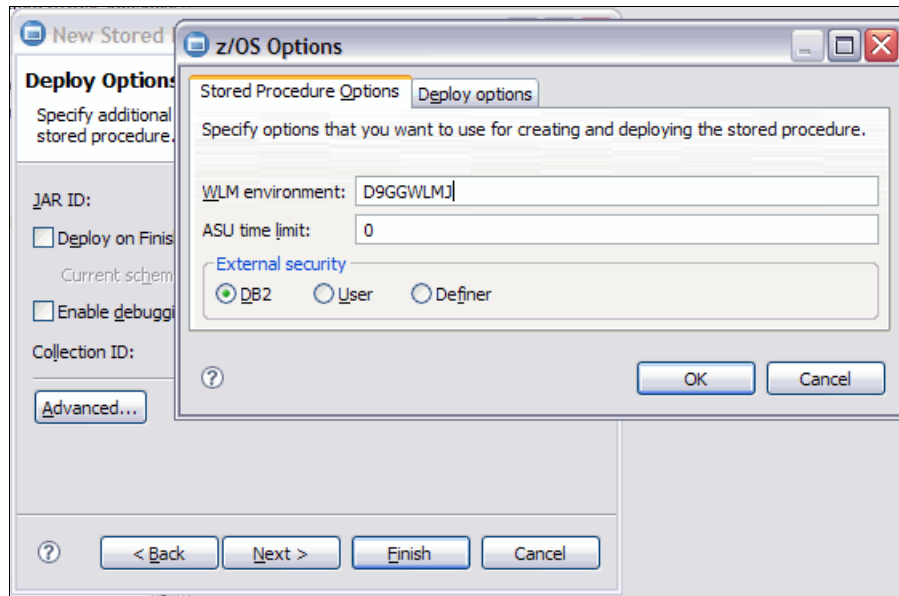


Figure 14-41 Define WLM environment

- Click **OK**. Then, to complete the stored procedure definition, click **Finish**.
- Next, you need to change the Java code. Expand the Data Project Explorer window to receive the Java source of the test procedure and double-click the Java file (see Figure 14-42).

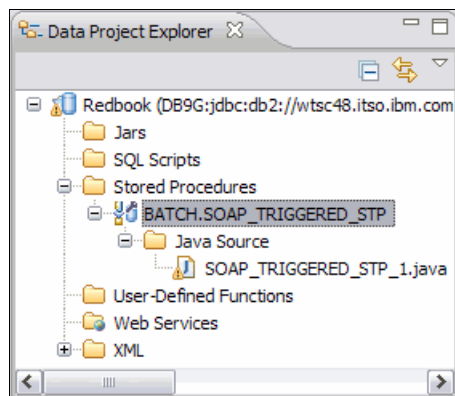


Figure 14-42 Open Java source

- Change the Java code to return a message to the Web Service caller as shown in Example 14-32.

Example 14-32 Return message to Web Service caller

```
public static void SOAP_TRIGGERED_STP_1(java.lang.String ACTION,
    java.lang.String PDF_DIR, java.lang.String[] OUT_MSG)
    throws SQLException, Exception {
    // Set return parameter
    OUT_MSG[0] = "This is our test Stored procedure";
}

```

11. Now, you can deploy this Java stored procedure by right-clicking the BATCH.SOAP_TRIGGERED_STP procedure and choosing **Deploy** (see Figure 14-43).

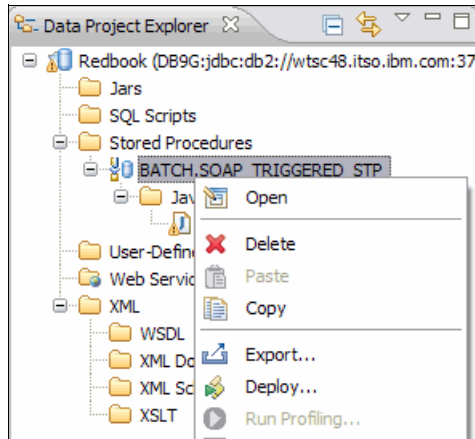


Figure 14-43 Deploy stored procedure

12. In the next window, set values for following parameters:

- Target Schema: BATCH
- Default Path: SYSIBM,SYSPROC

Then, click **Finish**. You can see the results of the deploy process in the SQL Results window (see Figure 14-44).

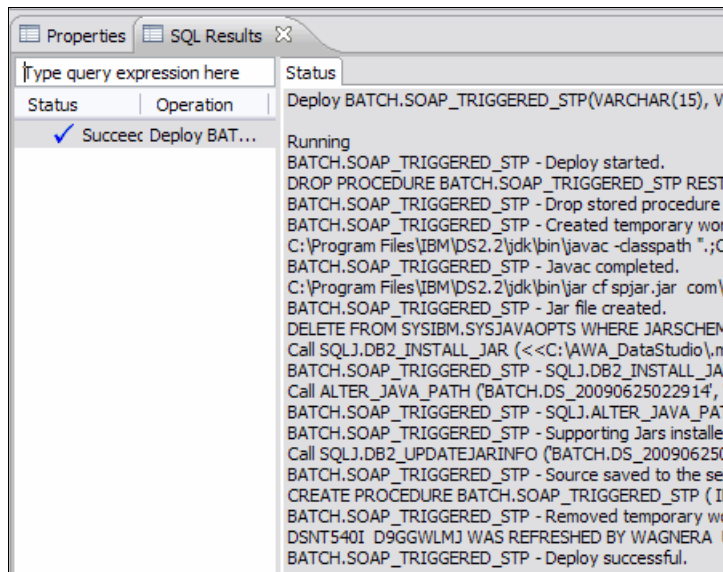


Figure 14-44 SQL Results

- To test the simple stored procedure, right-click **BATCH.SOAP_TRIGGERED_STP**, and click **Run**. In the window that opens, set up values for the two input parameters. Because this is only a test stored procedure, you can enter any value (see Figure 14-45).

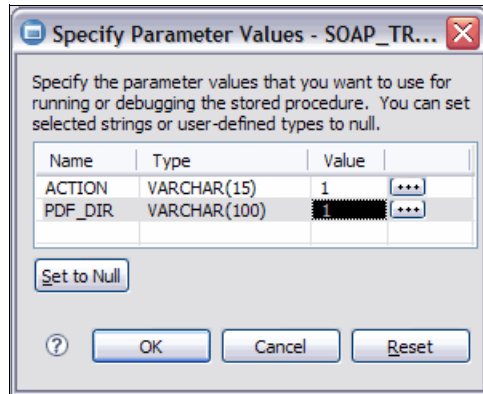


Figure 14-45 Set input parameter

- You can check the result of the test run in the Parameters tab of the SQL Results window (see Figure 14-46).

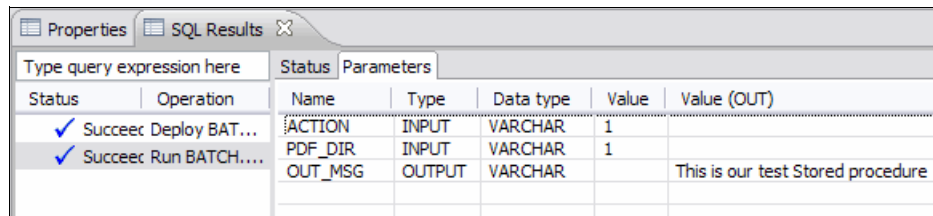


Figure 14-46 Test stored procedure

Creating a Web service that calls the stored procedure

To create a Web service that calls the sample stored procedure:

- Right-click **Web Services** (in Data Project Explorer), and choose **New Web Service**. In the window that opens, set the following values (see Figure 14-47), and click **Finish**.
 - Name: BatchStoredProcedure
 - Namespace URI: `http://redbook.itso.ibm.com/BatchStoredProcedure`

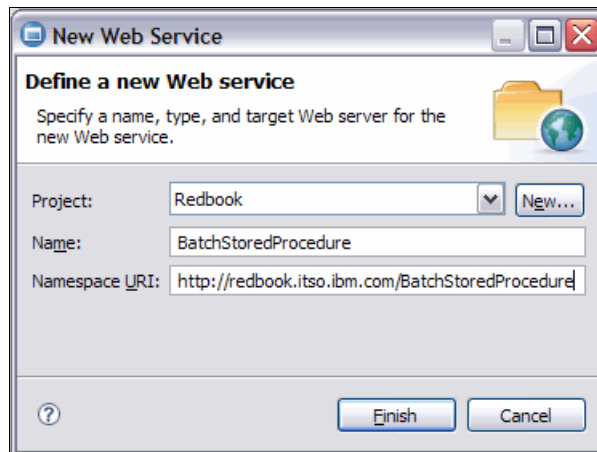


Figure 14-47 Set values for Web Service

- To add the stored procedure to this Web Service, right-click **BATCH.SOAP_TRIGGERED_STP**, and choose **Add to Web Service** (see Figure 14-48).

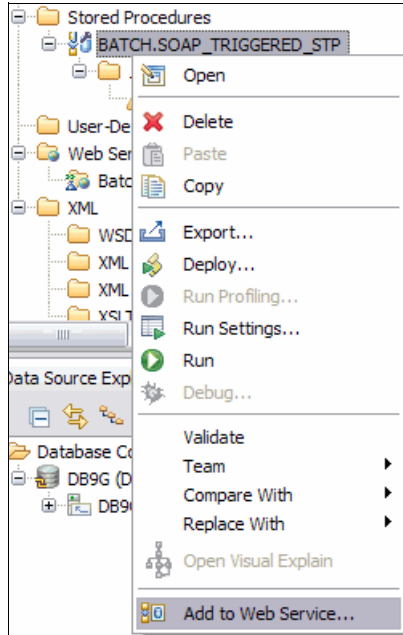


Figure 14-48 Add stored procedure to Web Service

- In the window that opens, select **BatchStoredProcedure**, and then click > to move this Web service from “Available Web services” to “Selected Web services.” Then, click **Next** and **Finish**.
- Next, build and deploy the Web service. Right-click the **BatchStoredProcedure** Web service, and choose **Build and Deploy** (see Figure 14-49).

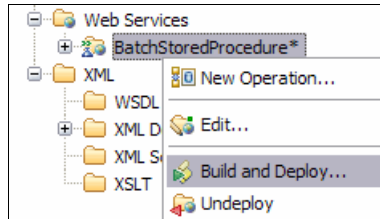


Figure 14-49 Build and Deploy Web Service

5. In the window that opens, change the following values (see Figure 14-50) and click **Finish**:
- Web server Type: WebSphere Application Server, Version 6 (all releases)
 - Message protocols: Clear the REST option (Web access)
- Clear this option because you do not need this feature for our test.

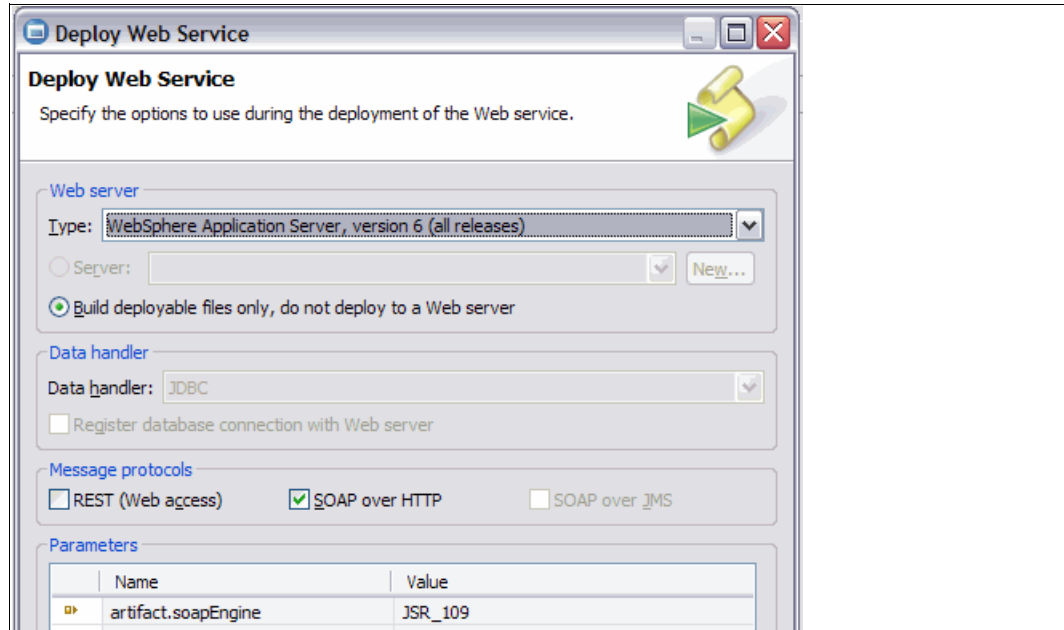


Figure 14-50 Change values for Web Service

Note: With this step, Data Studio creates a complete Java EE Web project. This project contains all the necessary data to call the stored procedure as a Web service (such as a WSDL file).

- Now, you need to change the Data Studio perspective. Select **Window** → **Open Perspective** → **Other**, and choose **Java EE** perspective as shown in Figure 14-51.

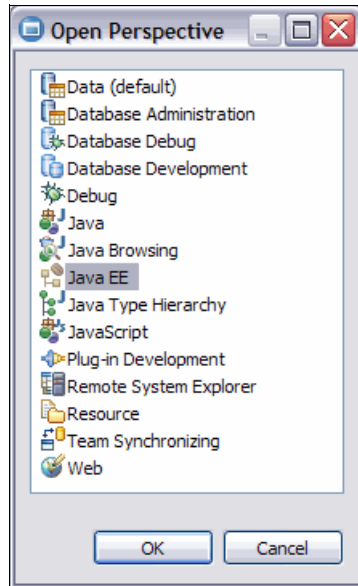


Figure 14-51 Open Java EE perspective

- Export the Web project by right-clicking **RedbookBatchStoredProcedureWeb**, and selecting **Export** → **WAR file** as shown in Figure 14-52.

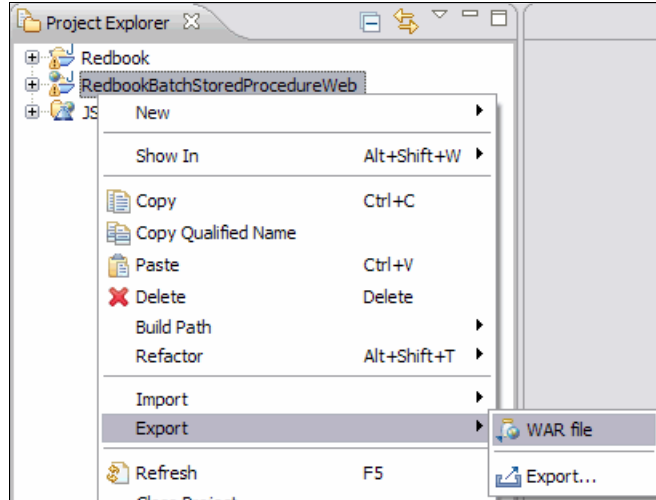


Figure 14-52 Export WAR file

- Finally, set a Destination (such as `c:\Temp\BatchSTP.war`), and then click **Finish**.

Deploying the Web service to WebSphere Application Server

You need to deploy the created WAR module to a WebSphere Application Server. Set the following options:

- ▶ Context root: BatchStoredProcedureWeb
- ▶ Data source: Resource Reference

The WAR module contains an application that uses a resource reference jdbc/BatchStoredProcedure to connect to the database. You must map this reference to an existing data source.

Testing the Web service

Data Studio contains a Web Services Test client. To use this Test client:

1. In the Java EE perspective, select **Run** → **Launch the Web Services Explorer** to start this test client. The Web Services Explorer window opens with an icon bar and three panes:

- Navigator
- Actions
- Status

2. In the icon bar, click the WSDL page icon, and select **WSDL main** in the Navigator window. In the WSDL URI field, enter the following URI:

http://<server>:<port>/BatchStoredProcedureWeb/services/BatchStoredProcedure?WSDL

In this URI, <server> is the host name of the WebSphere Application Server, and <port> is the HTTP port for the default_host virtual host.

3. As shown in Figure 14-53, select SOAP_TRIGGERED_STP in the Navigator window. In the Actions window, define values for the two input parameter. Because this is still the test Java procedure, you can enter any value. Call the new Web service by clicking **Go**.

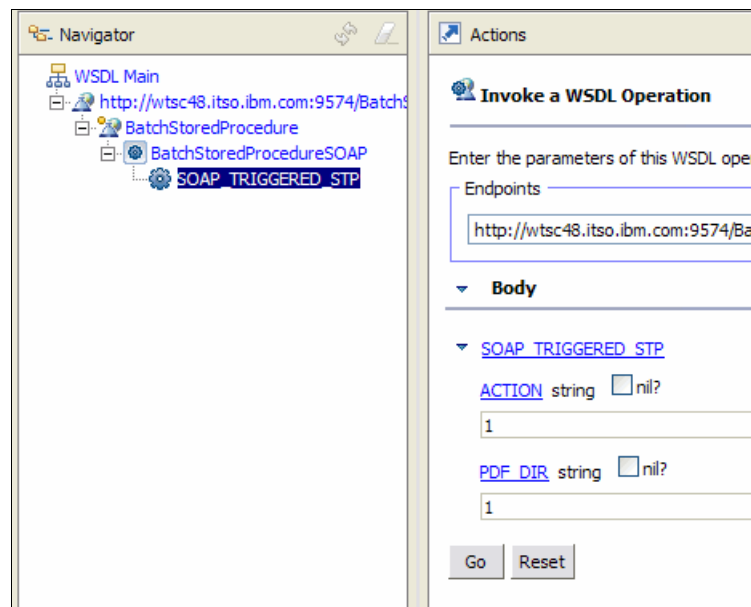


Figure 14-53 Testing a Web service using the Web Services Explorer

4. To see the result of the test, click **Source** in the Status window. The SOAP Request and Response displays as shown in Example 14-33.

Example 14-33 SOAP Request and Response

SOAP Request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:q0="http://redbook.itso.ibm.com/BatchStoredProcedure"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
- <soapenv:Body>
```

```

- <q0:SOAP_TRIGGERED_STP>
  <ACTION>1</ACTION>
  <PDF_DIR>1</PDF_DIR>
</q0:SOAP_TRIGGERED_STP>
</soapenv:Body>
</soapenv:Envelope>

```

SOAP Response:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
- <soapenv:Body>
- <SOAP_TRIGGERED_STPResponse
xmlns:ns1="http://redbook.itso.ibm.com/BatchStoredProcedure">
  <OUT_MSG>This is our test Stored procedure</OUT_MSG>
</SOAP_TRIGGERED_STPResponse>
</soapenv:Body>
</soapenv:Envelope>

```

Creating a Java stored procedure to generate PDF files

Because you have tested the test Web service successfully, you can now develop the “real” stored procedure to generate PDF files from XML data. You need to create the stored procedure in the same manner as describe in 6.3, “Java in DB2 for z/OS” on page 64 for stored procedure BATCH.JAVASTP. Follow these steps:

1. First, redefine the stored procedure in DB2 because the stored procedure needs to use a different main class. Thus, you need to DROP and CREATE our stored procedure.

The easiest way to execute SQL statements is to create an SQL Script in Data Studio. Change the Data perspective by right-clicking **SQL Scripts**, and selecting **New** → **SQL or XQuery Script**, as shown in Figure 14-54.

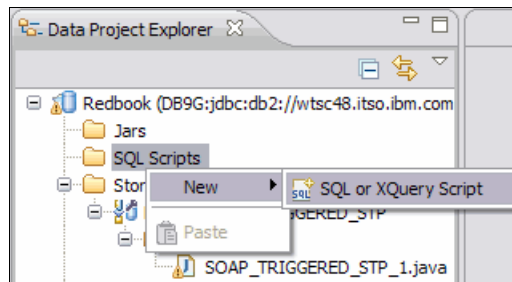


Figure 14-54 Creating a new SQL Script in Data Studio

2. Name the script ReplaceSTP.sql, and click **Finish**. In file ReplaceSTP.sql script, write the statements as shown in Example 14-34.

Example 14-34 Recreate stored procedure

```

DROP PROCEDURE BATCH.SOAP_TRIGGERED_STP RESTRICT;

CREATE PROCEDURE BATCH.SOAP_TRIGGERED_STP
  (IN ACTION    VARCHAR(15)
  ,IN PDF_DIR   VARCHAR(100)
  ,OUT OUT_MSG  VARCHAR(100))
  EXTERNAL NAME 'com.ibm.itso.sample.GenPdf.triggeredBySOAP'

```

```

LANGUAGE JAVA
PARAMETER STYLE JAVA
NOT DETERMINISTIC
FENCED
CALLED ON NULL INPUT
MODIFIES SQL DATA
NO DBINFO
NO COLLID
WLM ENVIRONMENT D9GGWLMJ
ASUTIME NO LIMIT
STAY RESIDENT YES
PROGRAM TYPE SUB
SECURITY DB2
INHERIT SPECIAL REGISTERS
STOP AFTER SYSTEM DEFAULT FAILURES
COMMIT ON RETURN NO
;
GRANT EXECUTE ON PROCEDURE BATCH.SOAP_TRIGGERED_STP TO PUBLIC;

```

- Execute this statement as shown in Figure 14-55. Right-click **ReplaceSTP.sql**, and select **Run SQL**.

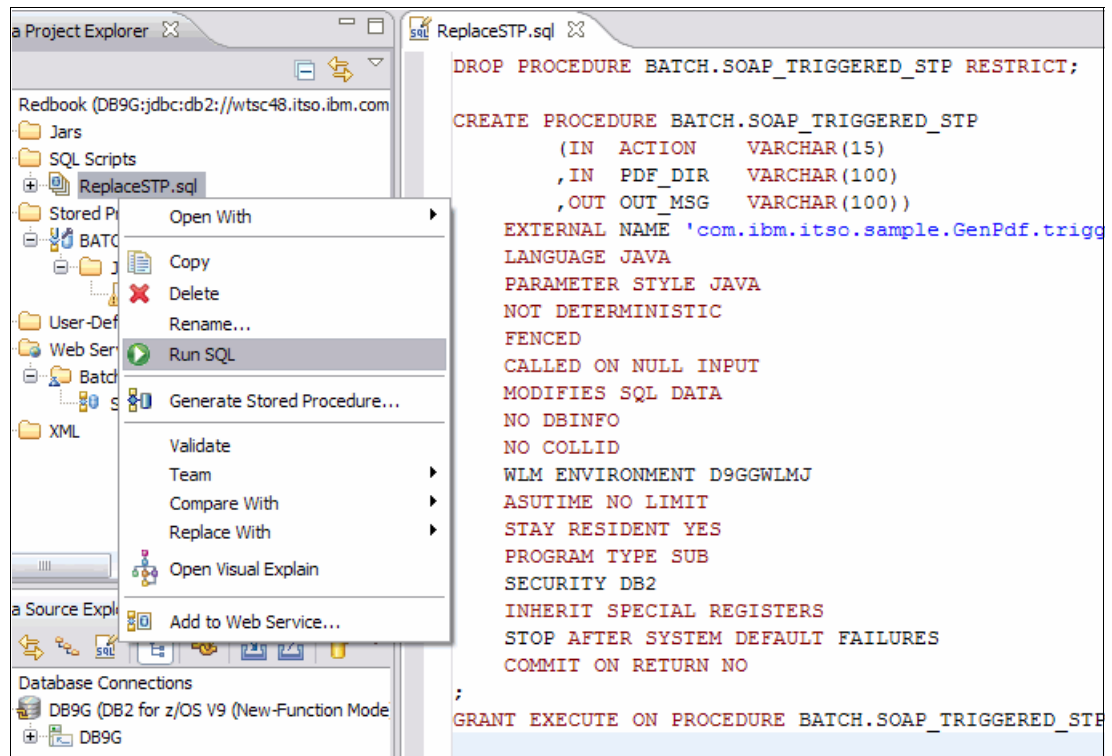


Figure 14-55 Recreating a stored procedure

You also need to create some Java classes in Data Studio. Because there are only a few differences, we briefly describe the necessary steps in Data Studio for the JavaSTP successfully. You can find more information about developing this Java class in 6.3, “Java in DB2 for z/OS” on page 64.

Important: You have to use the same Java compiler version in Data Studio Java projects as configured for the DB2 Java stored procedure runtime environment. If you do not know the configured DB2 Java runtime version ask your DB2 system programmer.

To create the Java classes in Data Studio:

1. Create a new Java project called *JavaSTP*.
2. Create a new folder called `lib`, and store the `iText-2.1.5.jar` file in this folder.
3. Add `iText-2.1.5.jar` to the Java build path.
4. Create the `deploy.xml` and `zos.properties` files in your project. For information about the content of these files, see 6.3, “Java in DB2 for z/OS” on page 64.
5. Create a Java package called `com.ibm.itso.sample`.
6. Create the `GenPDF` and `PdfCreator` Java classes. For this Java source code, see 6.3, “Java in DB2 for z/OS” on page 64.

Important: You have to add the `triggeredBySOAP` method to the `GenPDF` class as shown in Example 14-35.

Example 14-35 The `triggeredBySOAP` method and the `GenPdf` class

```
public static void triggeredBySOAP(String action, String pdfDir, String[] outMsg)
    throws SQLException {

    PdfCreator pdfc = new PdfCreator();
    pdfc.generatePDF(action, pdfDir);

    outMsg[0] = new String("PDF(s) successfully created.");
}
```

7. Append the `jakarta-oro-2.0.8.jar` and `commons-net-2.0.jar` JAR files for the Ant runtime to the Ant environment using **Window** → **Preferences** → **Ant Runtime** (as described in 6.3, “Java in DB2 for z/OS” on page 64).
8. Deploy Java sources by right-clicking **deploy.xml**, and then clicking **Run As** → **Ant Build**.

Here are the steps for the *UtilSTP* Java project:

1. Create a new Java project called *UtilSTP*.
2. Create a new folder called `lib`, and store the DB2 driver JAR files, `db2jcc.jar` and `db2jcc_license_cisuz.jar`, in this folder.
3. Add the driver JAR files to the Java build path.
4. Create a Java package called `com.ibm.itso.sample`.
5. Create the Java classes `RefreshWLM` and `TestSTP`. For Java source code, see 6.3, “Java in DB2 for z/OS” on page 64.

Important: You must change the TestSTP class as shown in Example 14-36.

Example 14-36 Change class TestSTP

```
CallableStatement cs = con.prepareCall("CALL BATCH.SOAP_TRIGGERED_STP(?, ?, ?)");
cs.setString(1, action);
cs.setString(2, pdfDir);
cs.registerOutParameter(3, java.sql.Types.VARCHAR);
cs.execute();
System.out.println("Call STP ended: " + cs.getString(3));
```

6. Refresh the WLM environment by right-clicking class **RefreshWLM**, and clicking **Run As** → **Java Application**.
7. Test the stored procedure by right-clicking class **RefreshWLM**, and clicking **Run As** → **Java Application**.

Final test to generate PDF files

Finally, you can test the Web service to generate PDF files. As mentioned previously, we use the integrated Data Studio Web Service Test client. In the Java EE perspective, select **Run** → **Launch the Web Services Explorer** to start this test client.

When starting the SOAP call, you have to set the input parameter. You must use special values because the stored procedure creates the files in UNIX System Services file system. Set the following values for the parameters:

- ▶ **ACTION:** One of the following values
 - store_fs
 - store_db2
 - store_fs_db2
- ▶ **PDF_DIR:** /u/wagnera/javastp/pdfs



Part 4

Improve batch efficiency

Traditional application designs typically split processing into the following processing windows:

- ▶ Daytime online processing, generally characterized by short-running transactions
- ▶ The overnight batch window
- ▶ Periodic batch windows, such as month-end processing

Note: In this book, we make no distinction between the overnight batch window and periodic windows because their characteristics are generally the same.

In many environments, the various batch windows are under increasing time pressure. The main sources of this pressure include the following factors:

- ▶ The need to complete batch processing in shorter time periods to allow the online day to be extended.
- ▶ The need to handle larger volumes of data.
- ▶ The need to incorporate additional functions.

These factors cause organizations to seek to speed up batch processing and to reduce the time that online applications are unavailable.

In this part of the book, we discuss methods to make batch more efficient and faster in the following chapters:

- ▶ Chapter 15, “Approaches and techniques to reduce the batch window” on page 301
- ▶ Chapter 16, “Increasing concurrency by exploiting BatchPipes” on page 309
- ▶ Chapter 17, “Batch application design and patterns in WebSphere XD Compute Grid” on page 319
- ▶ Chapter 18, “Java performance best practices” on page 337
- ▶ Chapter 19, “Increasing batch efficiency by using performance instrumentation” on page 341



Approaches and techniques to reduce the batch window

System/390 MVS Parallel Sysplex Batch Performance, SG24-2557 is the definitive work on how to tune a batch window. Though published in 1995, its methodology remains current. This chapter summarizes the approach that book advocated. Two key elements are particularly useful:

- ▶ The use of Project Management techniques, such as Gantt charts, in depicting the batch window and driving a project to reduce its run time.
- ▶ The pursuit of Seven Key Strategies to reduce the batch's run time.

This chapter also covers non-window batch, which that book did not cover.

This chapter includes the following topics:

- ▶ Project Management techniques
- ▶ Seven key strategies
- ▶ Non-window batch

15.1 Project Management techniques

Classic Project Management techniques work very well for understanding and reducing the batch window.

These techniques include

- ▶ Gantt charts to depict the batch timeline.
- ▶ Dependency (node and arc) diagrams.
- ▶ Critical Path Analysis.

However, before we discuss these techniques, consider the analogy of a batch window as a project.

A project can be viewed as a set of related activities to be completed within a given time period (with available resources). The activities are, of course, chosen to meet the objectives of the project. In addition, these activities have dependencies between them.

Now consider a typical batch workload. You can view the definition of an activity in several different ways:

- ▶ Each job is one activity.
- ▶ Each step is an activity.
- ▶ Each step consists of several activities.

These activities are inter-related, with the following dependencies:

- ▶ A data set is created in one job and read in another.
- ▶ Several jobs must complete reading a data set before another job can update it.
- ▶ A DB2 table must be processed by one job before another can use it.
- ▶ An FTP transmission must complete before other activities can commence.

Of course, as with real projects, additional work might need to be scheduled around the main batch window (and resources must be provisioned).

15.1.1 Gantt charts

A *Gantt chart* is a timeline diagram. For a regular project, the timeline is probably expressed in days, weeks, and months. For a batch window, it is expressed in hours and minutes.

Figure 15-1 depicts part of a batch window. The timeline is from 10 AM to midday. 29 job runnings are shown, with run times ranging from a few minutes to about 45 minutes.

This example has a high degree of parallelism. There are also several jobs (for example P08LO056) on which there is no benefit in working, because the run time is very small and completely overlapped with other jobs.

Other jobs (for example P01LO060) are long-running and obviously of significance in reducing the overall batch window.

While a good device for depicting a batch window, Gantt charts can become very large, spanning many pages. It is worthwhile to restrict their size, perhaps by removing short-running jobs (for example, those that run for less than 5 minutes).

It is possible to guess dependencies from Gantt charts; however, this method is not a foolproof way of deriving them, especially if you have removed short-running jobs.

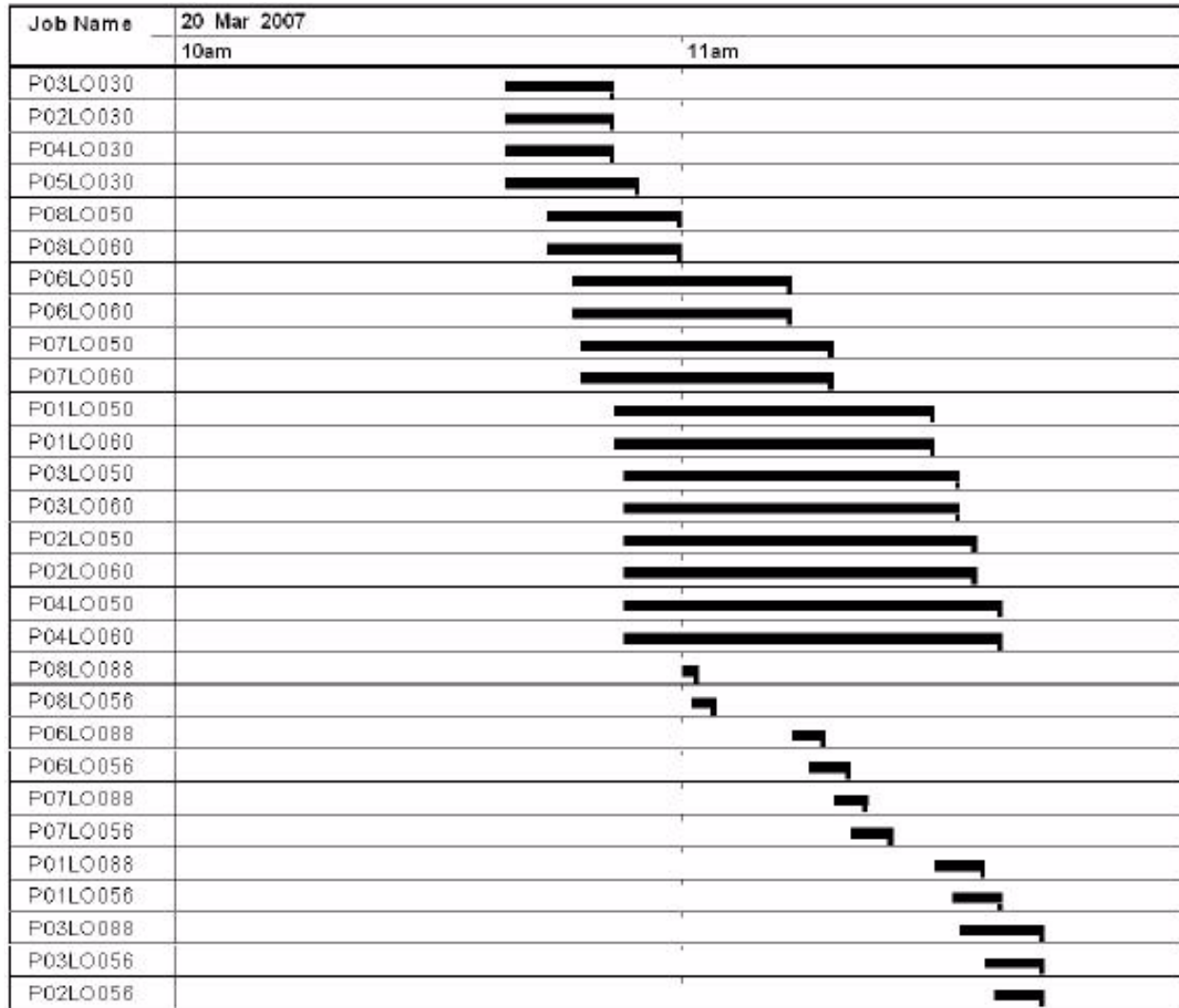


Figure 15-1 Sample Gantt chart depicting part of a batch window

15.1.2 Dependency diagrams

A useful technique for depicting the logic of a batch application is to draw a dependency diagram. Typically drawn at the job level, this comprises a series of boxes (nodes) and arrows (arcs).

Installations usually maintain such documentation manually, with each node being a job running and each arc a dependency (or, where appropriate, several dependencies such as a group of data sets).

A dependency diagram is useful for understanding why certain jobs have to wait for other jobs to complete. An installation seeking to reduce their batch run time would attempt to remove some of these dependencies, perhaps through a technique such as using BatchPipes/MVS pipe.

15.1.3 Critical Path Analysis

Critical Path Analysis establishes which jobs are on the critical path from the start of the window to the finish. Any reduction in the run time of a job on the critical path shortens the window. So establishing which jobs are on the critical path is important. Or at least that is the theory.

In practice, there are some problems with this approach, but it still has merit:

- ▶ The work in an installation's batch window varies considerably from night to night.
- ▶ Many installations' batch windows are so complex it is impractical to calculate which jobs are on the critical path.
- ▶ Real batch windows have multiple deadlines, multiple external triggers and multiple starting points.
- ▶ Jobs that are "near critical" are important as well. These are jobs that would become on the critical path if only a few jobs on the critical path were shortened.

None of these problems completely undermine the Critical Path Analysis approach: Some degree of pragmatism can be applied to the problem. For example, those jobs that are "near critical" are usually fairly obvious.

15.2 Seven key strategies

The following general approaches help reduce the batch window:

- ▶ Ensuring the system is properly configured
- ▶ Implementing data in memory (DIM)
- ▶ Optimizing I/O
- ▶ Increasing parallelism
- ▶ Reducing the impact of failures
- ▶ Increasing operational effectiveness
- ▶ Improving application efficiency

You can use these approaches (or *strategies*) in combination. You need to apply them appropriately to your specific environment and batch applications.

Some approaches can be implemented with system-wide or subsystem-wide actions, for example using the DB2 ALTER BUFFERPOOL command to increase the size of a DB2 subsystem's buffer pool. Other approaches require "engineering into" the workload, that is detailed implementation, such as implementing a BatchPipes/MVS pipeline, which we explain in detail in Chapter 16, "Increasing concurrency by exploiting BatchPipes" on page 309.

Note: Take care not to cause resource issues through the inappropriate use of the techniques or to cause batch processing failures (perhaps through poor understanding of the applications' logic and dependencies).

We now discuss these key strategies in more detail.

15.2.1 Ensuring the system is properly configured

Ensuring a properly configured system means making sure there are generally adequate system resources, so work flow is not inhibited, such as:

- ▶ Adequate CPU to avoid CPU queuing
- ▶ Memory so the system does not page
- ▶ A powerful enough I/O subsystem
- ▶ A DB2 Data Sharing environment with adequate resources

As well as resource considerations, ensure that important systems and other software are set up properly as follows:

- ▶ Workload Manager (WLM) set up to provide for good batch throughput
- ▶ DB2 subsystems optimally tuned for batch

15.2.2 Implementing data in memory

Implementing data in memory (DIM) techniques is a complex task, but one likely to yield good results. It means exploiting appropriately the various software facilities available to eliminate unnecessary I/Os. These I/Os include repeated reads of the same data, such as:

- ▶ DB2 buffer pools
- ▶ Virtual I/O (VIO) in Central Storage
- ▶ Queued Sequential Access Method (QSAM) buffering
- ▶ Batch LSR Subsystem (BLSR) exploiting VSAM Local Shared Resources buffering
- ▶ DFSORT Large Memory Object sorting

To be able to implement DIM techniques, you need spare memory, spare CPU capacity, and a batch workload that is I/O-intensive.

15.2.3 Optimizing I/O

Optimizing I/O means ensuring the I/O processing done by batch jobs is performed as quickly as possible, such as:

- ▶ The correct use of DS8000 Cache functions
- ▶ Exploiting modern Channel Programming functions such as zHPF and MIDAWs
- ▶ Using the fastest FICON channels
- ▶ Using HyperPAV to minimize IOSQ time

I/O optimization is most important when the batch workload is I/O-intensive. Where there is little spare CPU capacity I/O optimization could cause an I/O bottleneck to be replaced by a CPU bottleneck.

15.2.4 Increasing parallelism

Increasing parallelism means running more things alongside each other, such as:

- ▶ Reducing the elapsed time of a set of jobs by running more of them alongside each other.
- ▶ Performing I/O in parallel, using techniques such as Stripes.
- ▶ Performing DB2 queries using CPU Query Parallelism to use multiple TCBs.
- ▶ Cloning batch jobs to work against subsets of the data.

To be able to run more work in parallel, you need adequate resources, most importantly spare CPU capacity and memory, and adequate I/O bandwidth.

15.2.5 Reducing the impact of failures

Reducing the impact of failures means ensuring that any prolongation of the run time of the batch workload by failures is minimized. For example, this includes such actions as:

- ▶ Making batch jobs more reliable by fixing problems in the code.
- ▶ Ensuring that recovery procedures are effective and swift.

15.2.6 Increasing operational effectiveness

Increasing operational effectiveness means ensuring scheduling policies are executed accurately and promptly, for examples:

- ▶ Using an automatic scheduler such as Tivoli Workload Scheduler.
- ▶ Using the NetView® automation product.

15.2.7 Improving application efficiency

Improving application efficiency means looking at ways to make the applications process more efficiently, such as:

- ▶ Replacing obviously long sequential searches by hash searches or binary searches.
- ▶ Replacing self-written processing with DFSORT processing as this is optimized for speed.

15.3 Non-window batch

As well as batch defined by scheduler to run within a window there are other kinds of batch (collectively known as *non-window batch*) such as:

- ▶ Event-triggered batch jobs
- ▶ Development batch, such as compilations and functional tests
- ▶ Ad-hoc batch

The considerations for non-window batch performance apply also to window batch. For example, the need to manage the use of resources is common to both.

Non-window batch shares many characteristics with other “transaction-like” workloads:

- ▶ The work consists of independently-scheduled pieces.
- ▶ There might be many such pieces of work in the system at the same time.
- ▶ Resource interlocks, such as database table accesses or data set processes, are largely absent.
- ▶ Work arrival rates might be similar to those of online transactions. (Morning and Afternoon peaks might be present.)
- ▶ Work placement within a Parallel Sysplex environment can be flexible.

Note: The term “transaction-like” here refers to the bounded-duration independent nature of the work rather than more sophisticated transactional attributes, such as the ability to roll back work.

Because of the “transaction-like” nature of non-window batch many of the standard performance management techniques are important, rather than the more topology-driven approaches outlined in “Seven key strategies” on page 304. These include:

- ▶ Workload Manager (WLM) goal setting and management
- ▶ Resource management, perhaps with WLM Resource Groups
- ▶ WLM Period Aging, so that shorter-running batch jobs are treated better than longer-running jobs
- ▶ Initiator management (analogous to, for example, CICS transaction resource limits such as MAXTASKS)
- ▶ Job Class management
- ▶ Job placement around the Parallel Sysplex, by analogy to Session Placement

Note: In Part 3, “Implement agile batch” on page 187, we present a number of important topics in relation the ability of running non-window or “anytime” batch.



Increasing concurrency by exploiting BatchPipes

Note: In this chapter, we refer to *BatchPipes/MVS* as simply *Pipes*.

Pipes is a relatively old product, but one which continues to provide useful function, especially for increasing concurrency. Its BatchPipeWorks component also can enable some simplification by replacing self-written code with standard functions.

This chapter includes the following topics:

- ▶ Basic function
- ▶ Implementation
- ▶ New Pipes connectors
- ▶ Additional Pipes functions

16.1 Basic function

Consider the common case of a batch job step that writes a new sequential data set, followed by another job step that reads it. Traditionally the second job (the “reader”) cannot start processing the data set until the first job (the “writer”) has finished writing to it. So the two job steps cannot run concurrently.

Figure 16-1 shows such a writer / reader pair. Job W is the writer and Job R is the reader. Time flows from left to right, so in this case it is clear Job R runs *after* Job W.

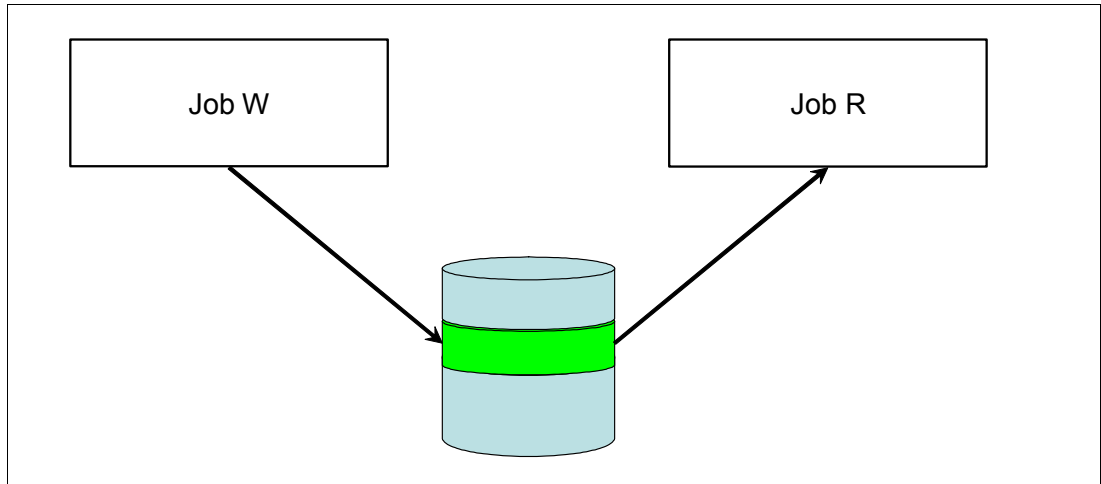


Figure 16-1 An unpiped writer/reader pair accessing a data set

With Pipes a short in-memory queue is created. This queue (or pipe) is written to by the writer job and read from by the reader job. As records become available in the pipe the reader can process them, removing them from the front of the queue. The writer places records on the back of the queue.

Figure 16-2 shows the same scenario as Figure 16-1 but with the disk data set replaced by a pipe. Again Job W writes the data and Job R reads it. This time Job W and Job R run together.

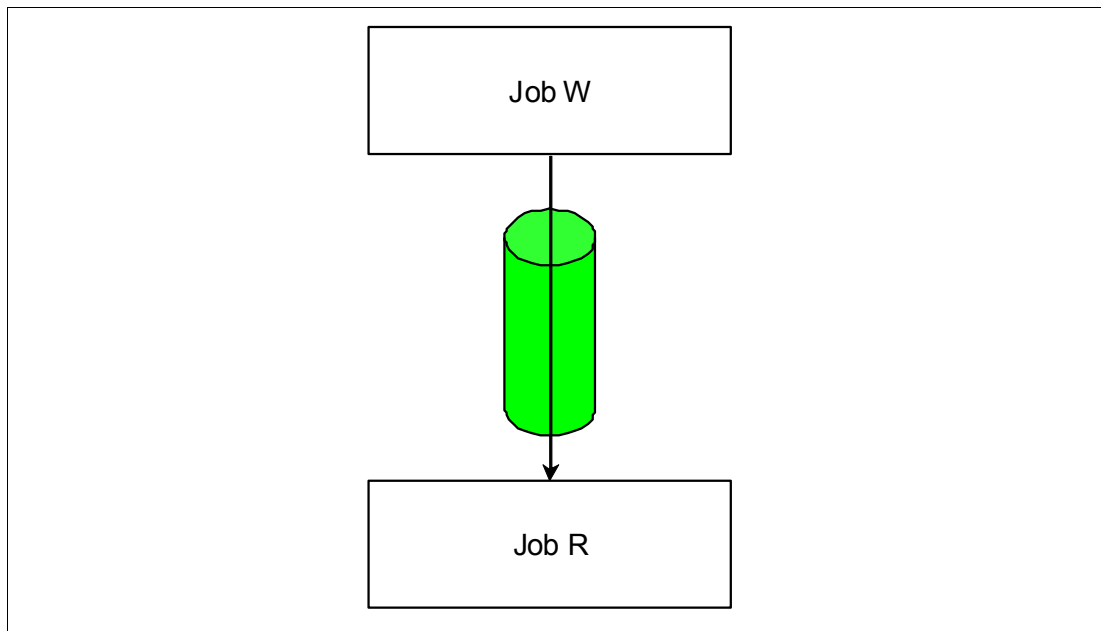


Figure 16-2 A writer/reader pair accessing a piped data set

The Pipes approach has a number of advantages:

- ▶ The reader and writer *must* run at the same time, increasing concurrency and shortening their combined run time.
- ▶ Whereas the traditional approach would have involved physical I/O the Pipes approach is “in memory”. This reduction in I/O has the potential to reduce run time.
- ▶ There is no disk or tape space requirement for the pipe. So this approach is essentially limitless.
- ▶ Where the physical data set would have been on tape, re-implementing as a pipe eliminates the tape mounts.

In this chapter, these above criteria are used to determine whether a technology is “pipeline-like.”

Job steps can participate in multiple pipes, whether as a reader, a writer, or both. Further, more than one reader can read from the same pipe (and more than one writer can write to the same pipe). This potential for creating complex topologies of pipes leads to the concept of a *pipeline*. Many pipeline topologies lead to more than two job steps being overlapped. Under these circumstances the potential for concurrency is even greater.

Pipes is supported for sequential file access using Basic Sequential Access Method (BSAM) or Queued Sequential Access Method (QSAM). It is not possible, for example, to pipe a Virtual Sequential Access Method (VSAM) data set between jobs. DFSORT automatically detects when a data set it is processing is a pipe and uses BSAM to process it. (DFSORT detects **any** subsystem data set and switches to BSAM.)

A sort is a good example of a processing operation that breaks the pipeline. Figure 16-3 illustrates such a situation.

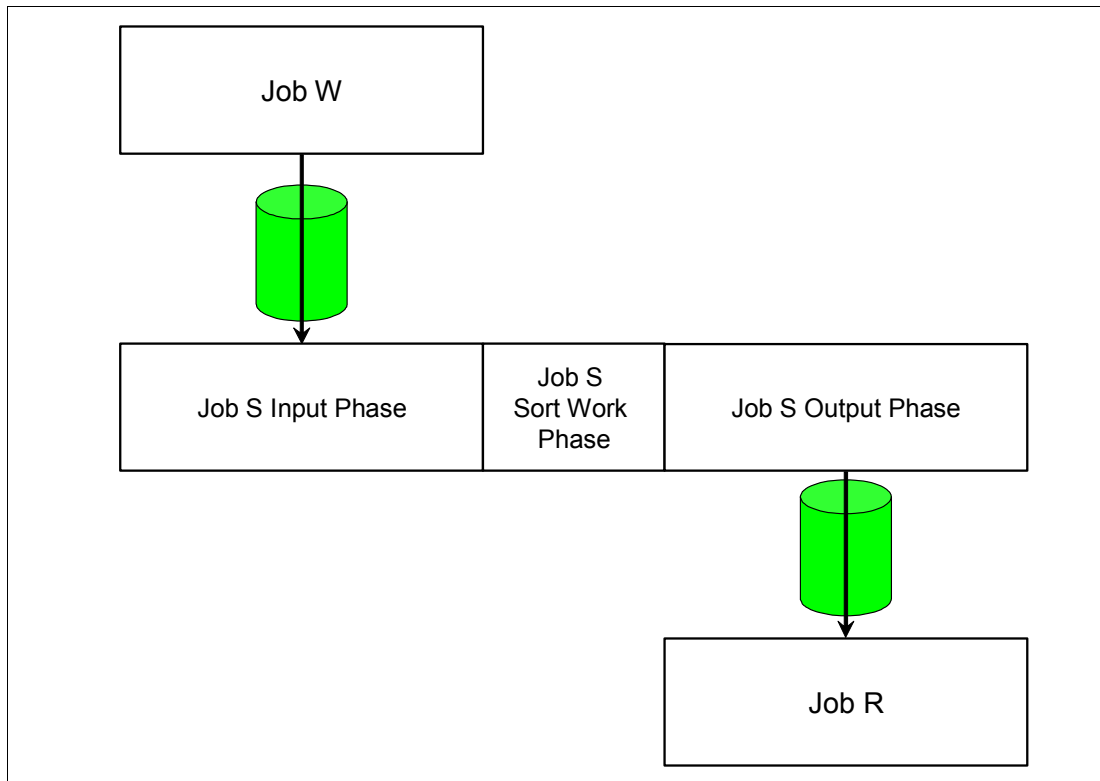


Figure 16-3 A sort operation in a pipeline

Suppose Job W writes a data set that Job S sorts and Job R reads the sorted data. Because the last record read into the sort might be the first record the sort writes there can be no overlap between the sort input and output phases. So Job W can be overlapped with the input phase of Job S but not with the output phase of Job S. Similarly Job R can be overlapped with the output phase of Job S but not the input phase. So Job W and Job R cannot overlap at all.

Pipes is able to use the *Coupling Facility* to create pipes between different z/OS images in a sysplex. See 16.4.1, “BatchPipePlex” on page 314 for more information about cross-sysplex pipes.

Pipes has additional record-processing capabilities, modelled on CMS Pipelines, described in 16.4.2, “BatchPipeWorks” on page 314. These can yield further concurrency and application simplification.

16.2 Implementation

This section is a basic introduction to implementing Pipes. There are two steps that need to be performed:

1. Set up the Pipes subsystem, outlined in 16.2.1, “Setting up the Pipes subsystem” on page 313.
2. Replace specific individual data sets by pipes, described in 16.2.2, “Implementing Individual Pipes” on page 313.

16.2.1 Setting up the Pipes subsystem

Pipes runs as a subsystem in a started task address space, whose name is the subsystem name. A popular choice for subsystem name is BP01. This address space must be started before jobs can create pipes.

You can control individual user ID's access to Pipes using the `PSP.ASFPIPE.ssnm` RACF FACILITY class where `ssnm` is the subsystem name. For individual pipes use RACF DATASET profiles. You can also control a user's access to BatchPipeWorks (described in 16.4.2, "BatchPipeWorks" on page 314) using the `PSP.ASFPFIT.ssnm` RACF FACILITY class.

To set up BatchPipePlex ensure the subsystems all have the same name and create the Coupling Facility list structure using your installation's Coupling Facility Resource Management (CFRM) policy. For more on BatchPipePlex see 16.4.1, "BatchPipePlex" on page 314.

Installations should enable SMF recording so that the performance of individual pipes can be monitored and managed.

16.2.2 Implementing Individual Pipes

Readers and writers for an individual pipe use a slightly modified DD statement. Often the only change is to add `SUBSYS=ssnm` to the DD statement. As a minimum all jobs using the pipe must:

- ▶ Specify the same pipe name, using the DSN parameter on the DD statement.
- ▶ Specify the same logical record length using the LRECL parameter on the DD statement.
- ▶ Specify the same record format using the RECFM parameter on the DD statement. This can be F, FB, V or VB. VBS data sets are not supported. It is highly recommended to use FB or VB.

The reader's block sizes must be at least as large as the writer's. IBM recommends that you do not code `BLKSIZE`. Not doing so means:

- ▶ For FB the system uses the largest value for block size that is a multiple of the record length and less than 32670.
- ▶ For VB the system uses the value 32760.

The block size determines how efficiently Pipes moves the data. The more records per block, the more efficient the movement of data is.

It is also necessary to arrange for the pipe's readers and writers to run at the same time, usually using the batch scheduler.

It is important to think through recovery scenarios, in case one of the jobs using the pipe fails.

The performance of an application that uses Pipes is important. Consider the case where a pipe's writer consistently outperforms the same pipe's reader. The writer will spend time waiting for the reader to process records. The application might run faster if a second (cloned) reader also read from the pipe.

16.3 New Pipes connectors

Two relatively recent APARs document new connectors to Pipes:

- ▶ The fix for APAR PK34251 allows the DB2 Template Utility to specify a pipe as input to the DB2 LOAD utility.
- ▶ The fix for APAR PK37032 allows users of IBM Communications Server's FTP functions to specify a pipe for PUT and GET requests.

Additionally, CICS SupportPac CA1J provides a connector to CICS transactions, allowing transfer of data from transactions using the standard COMMAREA mechanism.

Recently, the BPXWDYN Dynamic Allocation service was enhanced to support the SUBSYS parameter, which allows the JZOS ZFile Java class to allocate pipes in a similar manner to regular sequential data sets.

16.4 Additional Pipes functions

BatchPipes/MVS Version 2 has two additional sets of function which Version 1 didn't. (The SmartBatch product had these extra functions also but BatchPipes/MVS Version 2 is the currently marketed product.)

- ▶ BatchPipePlex allows pipes to be created that span the whole parallel sysplex.
- ▶ BatchPipeWorks allows additional concurrency and simplification.

16.4.1 BatchPipePlex

Many installations allow their batch work to spread across more than one LPAR in a parallel sysplex. With the basic Pipes implementation it is not possible for a writer to a pipe to be on one system and a reader on another. In the case where you cannot guarantee all the members of a pipeline will run on the same LPAR this could be a significant problem.

To pipe across systems within a parallel sysplex use *BatchPipePlex*. This uses a Coupling Facility structure to extend the queue that a basic pipe would implement.

The structure is called SYSASFP $ssnm$, where $ssnm$ is the name associated with the BatchPipes subsystems within the pipeplex (which must all have the same name.) For example, if all subsystems are called PSP1 the structure is called SYSASFPPSP1.

Because the structure has a fixed *single* name consider duplexing it to avoid it being a single point of failure.

Using BatchPipePlex is slower than using a basic pipe. Therefore it is recommended not to use BatchPipePlex for a single system pipe.

16.4.2 BatchPipeWorks

While the main purpose of Pipes is to provide increased concurrency, the product also has another useful capability: *BatchPipeWorks*.

BatchPipeWorks is a subset of the CMS Pipelines package, ported to z/OS. It allows the construction of pipelines from a palette of 100 built in stages as well as stages you can write yourself.

A *pipeline* is a series of programs called stages through which data flows. A pipeline consists of one or more stages and the output of one stage automatically becomes the input to the next stage. Each stage takes input records, performs a simple transformation on each of them, and passes the result on.

Examples of things you can do are:

- ▶ Select records based on criteria
- ▶ Reformat or change the contents of records
- ▶ Combine records
- ▶ Duplicate records
- ▶ Write your own REXX programs that change records
- ▶ Read data from and write data to data sets and the job log

Examples of built in stages are:

LOCATE	Selects records containing a specific string
CHOP	Truncates records at the specified length
SORT	Sorts the records
BPREAD	Reads records from a BatchPipes writer or a BatchPipes pipe
BPWRITE	Writes records to a BatchPipes reader or a BatchPipes pipe
BPCOPY	Makes copies of data flowing through the BatchPipes pipe
<	Reads from a data set
>	Writes to a data set

The last five stages introduce interesting possibilities:

- ▶ Writing data to both a pipe and a data set, perhaps to provide a “hardened” copy for downstream processing.

This is in fact more likely to be useful as a data backup rather than a recovery point for if the pipeline fails. This is because, at best, a partial set of output data would be written in the event of a failure. That would be difficult to recover from.

A second copy of a data set written to a pipe might be useful for downstream jobs that could *not* be scheduled to run in the pipe’s time frame.

- ▶ Writing two or more copies of the same data to two or more pipes, so that a corresponding number of reader jobs could read the data in parallel.
- ▶ Using a BatchPipeWorks pipeline to edit records written by a job before they are written to a data set (rather than a pipe). Such an arrangement is called a *half pipe* because it does not actually write to a pipe.

Figure 16-4 shows some of the possibilities. Job W writes to a pipe using a *fittings pipeline* with three stages or fittings. Fitting 3 also writes a hardened copy to disk.

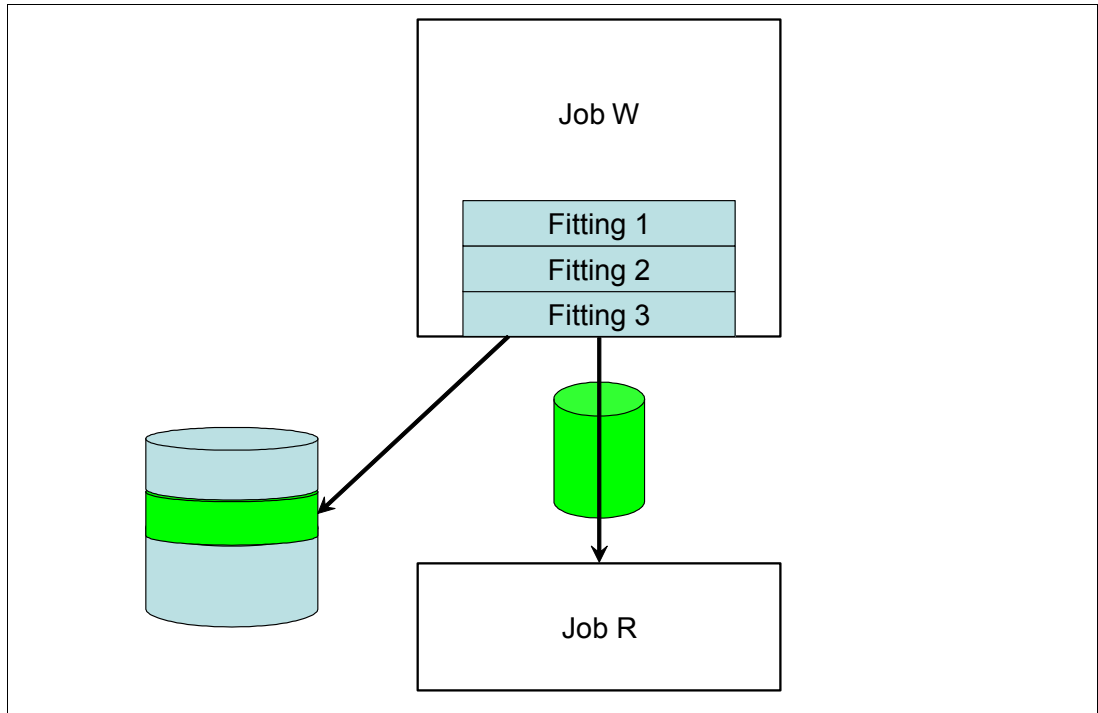


Figure 16-4 Fittings, including one that writes a hardened copy to disk

Here is an example of coding a fittings pipeline:

```
//OUTPUT DD DSN=MY.PIPE1,LRECL=80,RECFM=FB,
//      SUBSYS=(BP01,FIT='take 500 | chop 50 | bpwrite')
```

In this example, as the job writes a record it is passed to the pipeline. The first 500 records are kept and any subsequent records are discarded. These 500 records are chopped to 50 bytes in length and the records are written to the pipe MY.PIPE1.

Because BatchPipeWorks enables you to quickly add processing to an existing job using its already-present DD statements, you might be able to remove other home-written job steps and increase concurrency by using the BatchPipeWorks “data flow” programming style.

The next two sections discuss other techniques with capabilities that partially overlap those of BatchPipeWorks.

DFSORT similarities to BatchPipeWorks

Even without the use of IFTHEN, DFSORT invocations are a type of pipeline. Take the example of a SORT or COPY operation. (MERGE is only slightly different). A SORT can comprise the following stage-like processing elements:

1. Reading from the SORTIN DD
2. Throwing away the first *nnn* records with SKIPREC
3. Pre-processing records with an E15 exit
4. Keeping or throwing away records using INCLUDE / OMIT
5. Throwing away all records after the first *mmm* with STOPAFT
6. Sorting with SORT or copying with COPY
7. Summing (in the case of SORT) with SUM
8. Reformatting with OUTREC

9. Post-processing records with an E35 exit
10. OUTFIL processing

These processing elements apply in the sequence above (though it is normal to use only a few of them).

OUTFIL processing allows different “sub-pipes” to be used with each of several different output data sets. Each OUTFIL sub-pipe can comprise the following stage-like processing elements:

1. Record selection with STARTREC, ENDREC or SAMPLE
2. Record selection with INCLUDE, OMIT, or SAVE
3. Various actions such as reformatting with OUTREC and parsing with PARSE
4. Splitting the output stream with, for example, SPLIT

Again, these elements (where present) must be in the sequence above.

There is another level of pipelining: IFTHEN processing. With IFTHEN you can pass data through multiple reformatting stages. IFTHEN is available *within* INREC, OUTREC and OUTFIL OUTREC statements.

With the potential exception of sort work data sets in the SORT stage, intermediate data is not written to any intermediate data set and records are passed one at a time through the stage (except for SUM where their values are accumulated).

As previously mentioned, DFSORT input and output data sets can be BatchPipes pipes.

When DFSORT is invoked more than once in an ICETOOL step these are separate pipelines in that the results of one DFSORT invocation can be fed into the next DFSORT invocation but using only intermediate data sets.

UNIX pipes in batch with BPXBATCH

UNIX has its own pipeline support through the use of the pipe symbol (|), separating UNIX commands. Here’s an example, using BPXBATCH:

```
//B1 EXEC PGM=BPXBATCH,PARM='SH ls -a | tail +3 | nl'  
//STDOUT DD SYSOUT=*
```

In this example, the list of files in the working directory is generated. The top two lines of the output are thrown away and the remaining lines are numbered.

Note: The UNIX commands are not the same as the Pipes stages.

It is not possible to replace BatchPipeWorks pipelines with UNIX pipelines because the latter cannot read from or write to any DD statements other than STDIN and STDOUT and there are restrictions on the types of data sets STDIN and STDOUT can use.



Batch application design and patterns in WebSphere XD Compute Grid

This chapter introduces a set of design patterns and development approaches for batch applications. Designing applications can be a challenging task. Alternatively, the current problem must be solved; however, the implementation must be flexible enough to adopt future requirements easily. In addition, the slightest inefficiencies in processing a single record can be exacerbated by the sheer volume of records to be processed.

Object-oriented (OO) programming techniques, when applied well, is an important tool for building agile applications. However, OO is still a programming technique and does not provide any standard template of sorts to solve a particular class of problems. Design patterns provide a reusable template for solving a particular type of problem.

This chapter discusses the design patterns, which focus on separating concerns through encapsulation and componentization, principles that are important for building agile applications. The chapter also discusses how to share services across batch and OLTP, and includes a sample whose source code can be found in Appendix C, “Additional material” on page 453.

17.1 The Strategy pattern as the foundation for designing batch applications

Designing batch applications can be a challenging task. Two important design facts must be kept in mind: massive volume of records to be processed exacerbates any inefficiencies in the implementation; and agility is important for improving time-to-market, reducing maintenance costs, and improving application stability, where business services can be easily composed into new applications. By following a set of design principles, we can achieve both performance and agility.

The first design principle to follow is the *Strategy* pattern, where components within the application are decoupled through interfaces. Each component is tasked with solving a specific problem: reading data, processing data, or writing data. This decoupling enables different algorithms to be plugged into the application with zero impact to the other components. Figure 17-1 illustrates this concept.

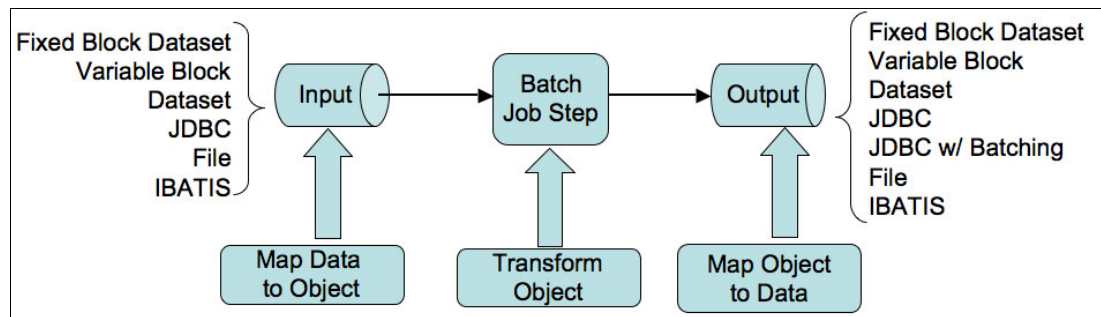


Figure 17-1 Applying the Strategy pattern to batch applications

Figure 17-1 illustrates three components to a batch job: *Input*, *Batch Job Step*, and *Output*. The Input component is a domain-object factory, which maps raw bytes into a *valid* domain object. The Batch Job Step component contains the business logic, including validations, transformations, and business rules that execute against a valid domain object. This component does not know, nor should it care, how the domain object was obtained, it simply expects a complete (valid) domain object. Finally the Output component's only task is to map a processed domain object to raw bytes. The Output component does not care what processing took place, only that the processed domain object is valid and can be persisted. Each component operates on a *single* record, which is important when discussing parallel processing strategies.

Strict enforcement of this encapsulation ensures agility and service reuse. For example, we can change the source of records from a file to a database by changing the input implementation. This change will be in complete isolation of the Batch Job Step and the Output components. Likewise, we can change the Output implementation to persist data to a fixed-block data set on z/OS instead of a text file, and this change will be unknown to the other modules. Figure two illustrates an example of applying the strategy pattern not only for encapsulating the input, process, and output components, but also within the process component (also know as the *kernel*) where validations can be plugged in.

As Figure 17-2 illustrates, the *Input Data Stream (Input DS)* produces valid input domain objects. One record is passed to the kernel for processing, where the kernel knows nothing about how that domain object was obtained. The kernel, also using the strategy pattern, will apply a set of validations and other business logic to the record. As new requirements are dictated, additional validations can be plugged into the kernel.

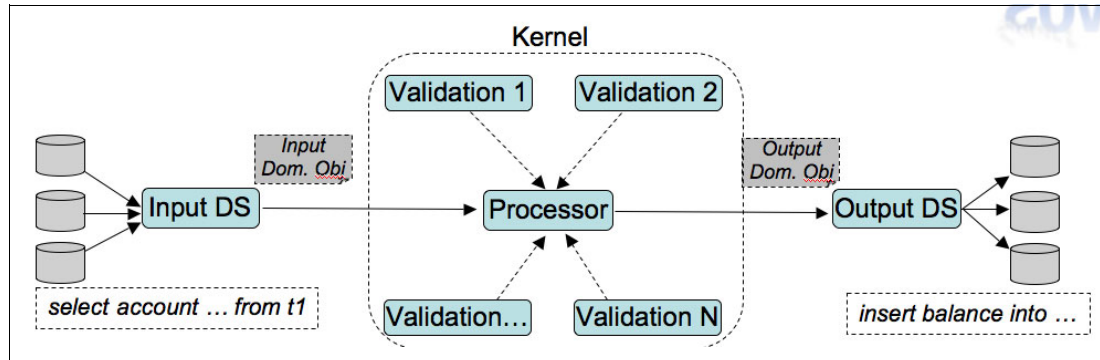


Figure 17-2 Strategy pattern applied to create a kernel that processes batch records

Finally, the kernel returns the output domain object to the *Output Data Stream (Output DS)* for persistence.

Validations and business rules that are applied to a batch application must be carefully designed. Validations typically make use of referential data, for example, in order to validate whether a bank account is active or not, the account number will get looked up in some database. In the context of batch, the cost of a validation is magnified because of the number of records that must be processed. If a validation takes 100 ms, and you must process 10 million records, the absolute fastest time the job can execute in is 1 million seconds. Caching, data-aware routing, and complex SQL queries are a few strategies that can be applied to reduce the complexity of the validations. For example, a more efficient way to process valid bank account records is to add a clause in the predicate of the SQL query, `select accounts from accountTable where account.status = 'active'`. Each record processed by the kernel will be active; therefore the validation can be removed.

There are two benefits to designing the kernel to operate on a single record whose origin is unknown: first, by writing each module to only operate on one record at a time, we can shift the burden of parallel processing outside of the application code and to the infrastructure; and second, the kernel can evolve to become a service that is shared across multiple execution styles, batch, online transactional processing, and so forth, without sacrificing performance optimizations that each paradigm provides.

Writing multi-threaded code can be difficult, but can significantly reduce the elapsed time of a batch job. Maintaining and debugging multi-threaded code though can be expensive. By designing a kernel that is agnostic to the source of valid domain objects, parallel processing can be handled by the infrastructure. As Figure 17-3 illustrates, the Input DS can be designed to accept parameters that dictate the range of data that should be retrieved. In this example, a constraint in the predicate of the SQL query is used, but byte ranges in files, record ranges in Fixed Block data sets, and so forth are also possible.

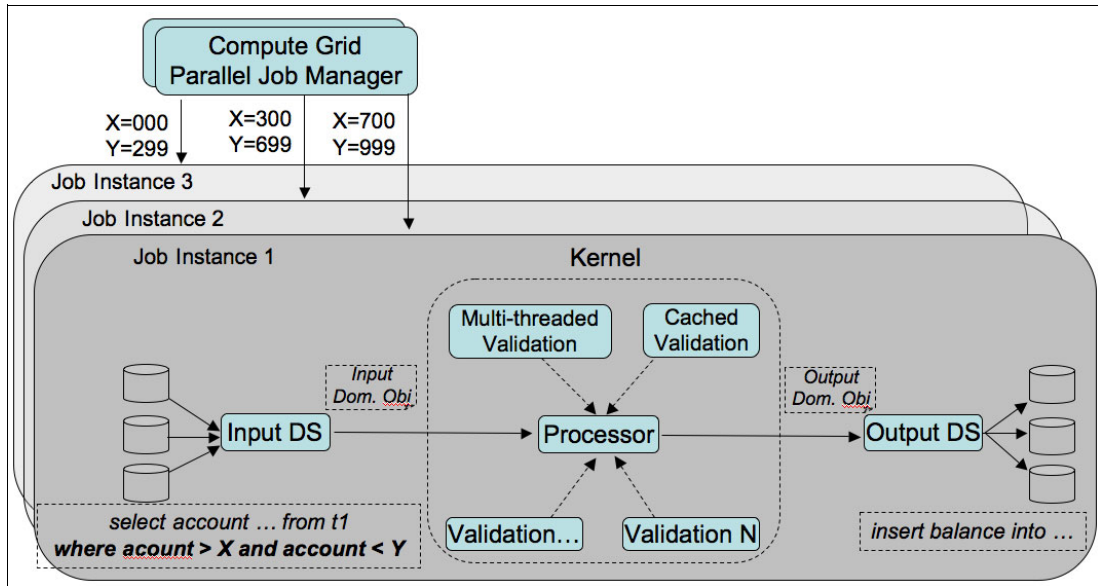


Figure 17-3 Shifting the burden of parallel batch from the application to the infrastructure

Some external parallel processing engine can create multiple instances of a batch job, in this case, the *Parallel Job Manager* component of WebSphere XD Compute Grid applies a partitioning algorithm that determines the range of data that each job instance will operate on. To maximize throughput, the partitioning strategy for a batch job should match how the data has been partitioned. More over, the partitioning strategy can independent of the application, where the application developer is not responsible for determining how many instances (or threads) must be executed, nor on what data each thread will operate. By shifting this burden to the infrastructure, workload management data, capacity metrics, service-level agreements, and database optimizations can be used to determine the ideal concurrency setting at a specific moment. The infrastructure optimizations, as well as the details of how the Parallel Job Manager operates, are outside of the scope of this chapter.

Performance of batch applications tends to be very important. Not only must batch applications complete within a specific (and often demanding) deadline, but also any inefficiency in the application is magnified. Using the strategy pattern and clearly separating out the roles of input, process, and output bottom-up performance analysis can help monitor and debug performance degradations. For example, by taking start and end times before and after each call to the Input component, we are able to get a rough idea of how much time is spent fetching records. Using the `System.currentTimeMillis` call can be highly inaccurate when processing one record, but when processing hundreds of thousands of records, this time becomes more accurate. Example 17-1 illustrates this concept.

Example 17-1 Measuring time spent on fetching records

```
// For each record perform the following logic as part of a loop

startTime = System.currentTimeMillis();
record = readRecord();
endTime = System.currentTimeMillis();
recordCount++;
totalReadTime = totalReadTime + (endTime - startTime);
startTime = System.currentTimeMillis();
processedRecord = processRecord(record);
endTime = System.currentTimeMillis();
totalProcessTime = totalProcessTime + (endTime - startTime);
```

```

startTime = System.currentTimeMillis();
writeRecord(processRecord);
endTime = System.currentTimeMillis();
totalWriteTime = totalWriteTime + (endTime - startTime);

// Now, include code to print average read/process/write/times

```

At the end of the job, we can calculate the average time spent reading/processing/writing records. A simple performance test could be to determine the amount of time each component spent completing their respective tasks, and identifying (relatively) where most of the time was spent in the batch job. For example, when executing a job and gathering these performance metrics, one can determine that 60% of the time was spent reading records while the remaining 40% were split evenly in processing and writing records.

The first place to further analyze for performance improvements could be the Input component. The performance analysis of components could be incorporated into a continuous-integration platform as well. Because the input, process, and output modules are independent from one another, each can be executed separately. For example, a system can be built for each module where the input test data is constant, each new version of the module is executed with the constant test data, and the elapsed time for each version is tracked, where any degradation in elapsed time indicates a performance regression. This bottom-up performance analysis can help ensure each module is highly optimized. By focusing on building high-performance components, the assembly of these components into new batch jobs helps ensure that these new jobs are already well optimized.

By building highly optimized, loosely integrated components, it is possible to build a library of modules that can be easily assembled into new batch applications. As Figure 17-4 illustrates, WebSphere XD Compute Grid xJCL definitions can be used to assemble the various input, processing, and output components into batch jobs. By building a bottom-up performance analysis process, each module will ideally be optimized, helping to ensure that the new jobs also perform well.

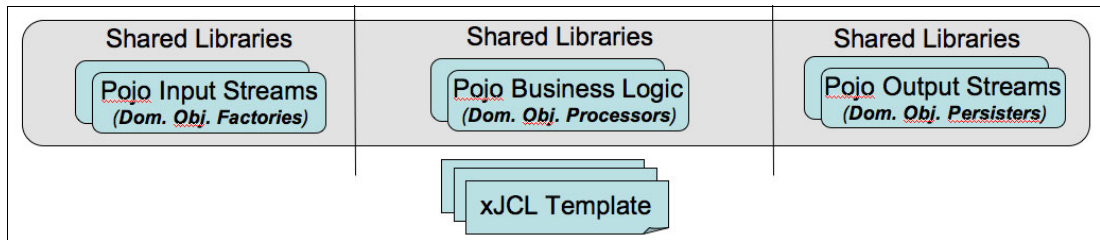


Figure 17-4 Libraries of functions can be created and assembled using some metadata

Further processes can be built around this library of functions, where application developers' focus on building more functions into the libraries, and business analysts can assemble new functions without having to develop or understand the underlying code. Figure 17-5 illustrates this concept.

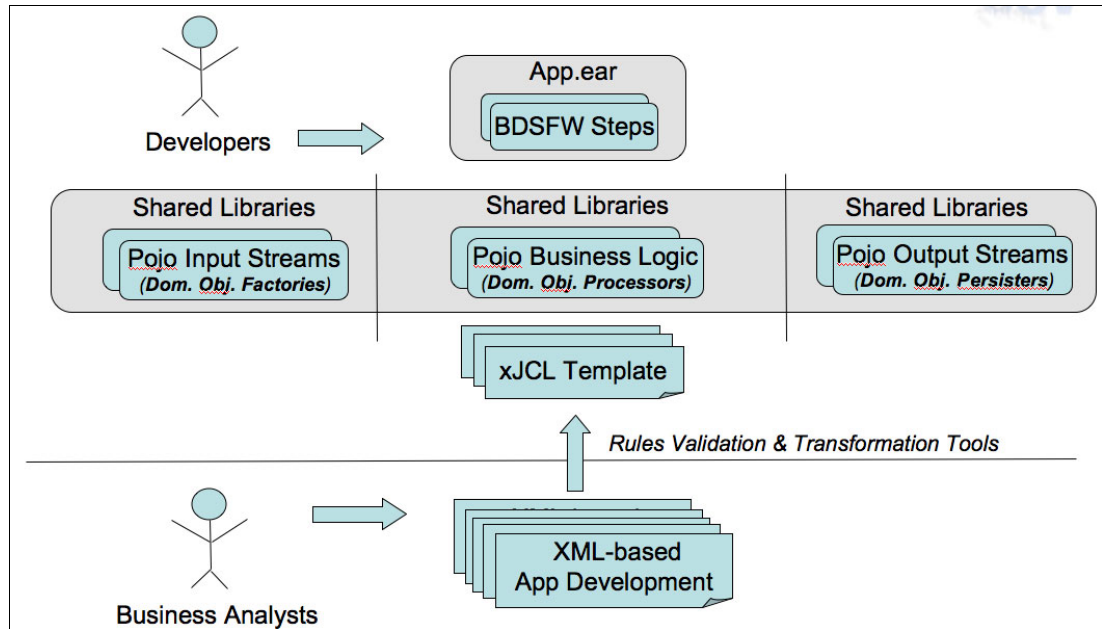


Figure 17-5 Domain-driven development & Business-Driven Assembly.

17.1.1 The Batch Data Stream Framework and its implementation of these patterns

The better the contract between an application and its execution container, the more container-managed services can be provided. In WebSphere XD Compute Grid, the contract is the programming model for implementing batch data streams and batch job steps. The J2EE programming model allows the EJB container to provide services such as security demarcation, transaction context management, caching, and so on. Similarly, the WebSphere XD Compute Grid batch container, a peer to the EJB container within an application server, integrates with batch applications through two programming interfaces: one for providing checkpoint and restart functionality for input and output data streams; and the other for influencing the checkpoint (transaction) interval for a batch job step. To take advantage of the various container services provided by the WebSphere XD Compute Grid batch container, applications must implement the appropriate interfaces.

The Batch Data Stream Framework (BDS Framework) provides a tight contract between the application and the batch container, incorporates best practices, provides design patterns using templates and callbacks that simplify the task of developing business logic, all while ensuring there is adequate decoupling between the business logic and the proprietary WebSphere XD Compute Grid APIs.

There are two components of the framework: first, templates for various input and output stream types (reading/writing with JDBC, MVS data sets, UNIX files, and so forth), where the nuances of checkpointing and restarting each of the input and output stream types have been implemented; and second, various application design patterns, such as the strategy pattern, and application services such as performance metrics, logging, threshold policies, and so on. The UML for the current version of the BDSFW can be found in Appendix C, "Additional material" on page 453. Details about how to implement BDS Framework applications can be

found in Chapter 8, “Implement new functionality using Java in WebSphere XD Compute Grid” on page 93.

17.1.2 Sharing business services across batch and OLTP

Separating the roles of input, process, and output as outlined in the previous section can help ensure that agile, high performance batch applications are built. The next step is to build services that can run in batch and OLTP infrastructures without sacrificing the performance optimizations available in each domain. Batch and OLTP workloads are fundamentally different; therefore there are very different strategies for maximizing the throughput of each workload type. Traditionally there are two fundamental types of batch workloads to consider: first, asynchronous random-access batch jobs; and second, asynchronous, sequential-access batch jobs.

Asynchronous, random-access batch jobs are similar to OLTP applications—data is accessed randomly and therefore the optimizations available include caching and data-aware routing. Random user-requests are passed to the infrastructure using a queue-based infrastructure such as WebSphere MQ. The Message Driven Beans (MDBs) waiting for new work in the queues will execute one message (that is 1 record) at a time.

Batch workloads typically read and write data sequentially whereas OLTP workloads read and write data randomly. Batch workloads access DB2 directly using JDBC whereas OLTP applications will use an object-relational mapping technology such as Java Persistence Architecture (JPA) or Hibernate.

There are four typical performance optimizations that are applied for batch workloads:

- ▶ A single select query is executed to retrieve the rows to be processed; the database cursor is marked to remain open across transactions (also known as *Cursor Holdability*). The anti-performance approach is to issue many smaller select queries, where each query is demarcated by the global transaction managed by the application server’s transaction manager; upon a global transaction commit, the cursor is closed by the database.
- ▶ Tuning the DB2 z/OS buffer pools to hold larger blocks of data. Because the batch workloads are processing data sequentially, larger buffer pools enable more data to be prefetched from the database. Refer to “Buffer pool size” on page 218 for more details on this topic.
- ▶ Data prefetching threads within the application server, where background threads are continuously prefetching data to ensure the thread of execution is not blocked and waiting on I/O.
- ▶ Using the JDBC batch features of prepared statements in Java. Because data will be written in bulk, N prepared statements can be grouped and executed together as opposed to executing one statement at a time. There are a number of advantages here: first, there are M/N remote calls to the database when using JDBC batch versus M remote calls, where M is the total number of statements to be executed and N is the size of each group of statements batched together; second, the database is able to optimize the execution of the statements. Database products such as DB2 for z/OS, DB2 UDB, and Oracle 10g have different optimal batch values. A simple performance test was run with DB2 for z/OS, where 10 million business records were inserted, varying the JDBC batch size from 1 to 50. Figure 17-6 on page 326 illustrates the results, where we found a batch size of 20 records was optimal and effectively doubled throughput from ~10,000 records per second to ~21,000.

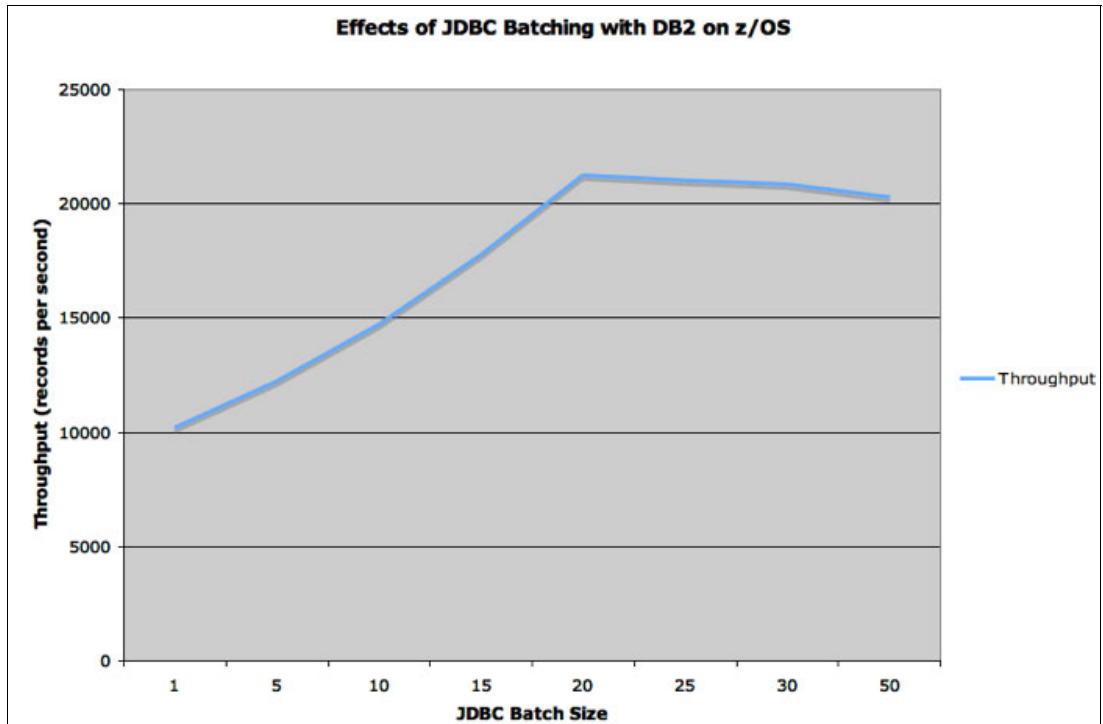


Figure 17-6 Performance benefits of applying JDBC batching

The Strategy pattern serves as a foundation for sharing business services across batch and OLTP. The kernel described in the previous section can be the service that is shared across batch and OLTP. The kernel is designed to be *data injected*, where the data to be processed is passed to the kernel by some domain object factory. The alternative approach is a service that follows a *data acquired* pattern, where the service will explicitly retrieve the domain object that is to be processed. The latter type of service is typically found in OLTP applications (Web Services, EJBs, and so forth).

Figure 17-7 illustrates how services can be shared across batch and OLTP. The shared service is data injected, where the domain object (input Data Transfer Object or iDTO) is passed into the service by some external mechanism.

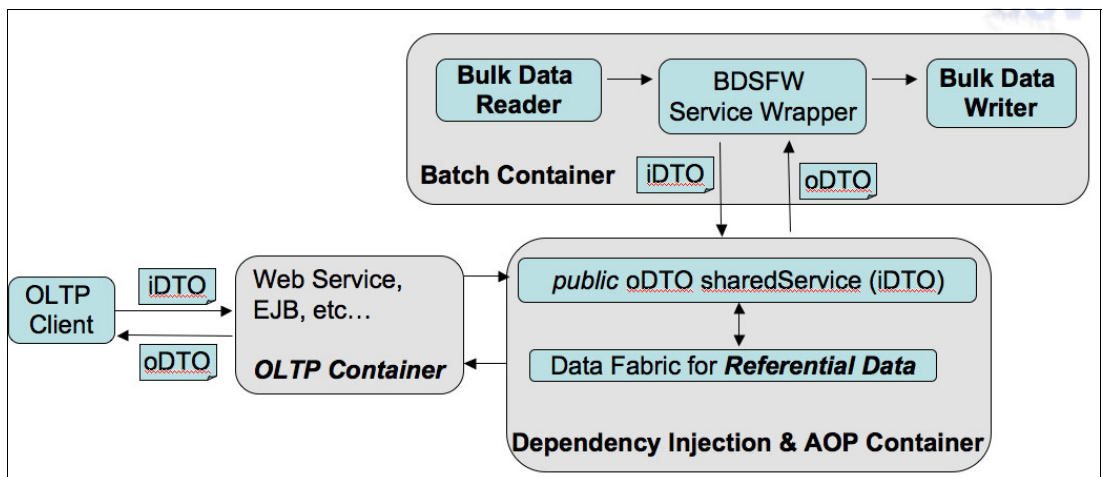


Figure 17-7 An approach for sharing services across batch and OLTP

The processed domain object (oDTO, which stands for “output Data Transfer Object”) is returned by the shared service to the caller. When this service is executed in OLTP mode, an OLTP wrapper (Web service, EJB, and so forth) will obtain the iDTO, typically through some OLTP-oriented data access layer, and invoke the shared service. When executing in batch mode, bulk data readers and writers, managed by some batch container, will feed records into the shared service. A more concrete example of this shared-services architecture is available in Appendix C, “Additional material” on page 453.

A key component to modern batch processing is the use of container-managed services, where the batch application only contains business-specific code. J2EE application servers provide several containers: Web Container for managing servlets, EJB container for managing Enterprise Java Beans, and so forth. These containers demarcate transactions, enforce security roles, integrate with application server services such as connection and thread pooling, and so forth. The role of containers, and maintaining their responsibilities is important in terms of sharing services across batch and OLTP. WebSphere XD Compute Grid delivers a batch container, which is a peer to the EJB and Web containers within an application server. The batch container provides services like checkpoint/restart, where batch jobs can be restarted at some point in time with complete transactional and data integrity, as well as enforce security identities, among many other functions. Figure 17-8 illustrates the role of OLTP containers and the batch container when sharing business services across both of those domains.

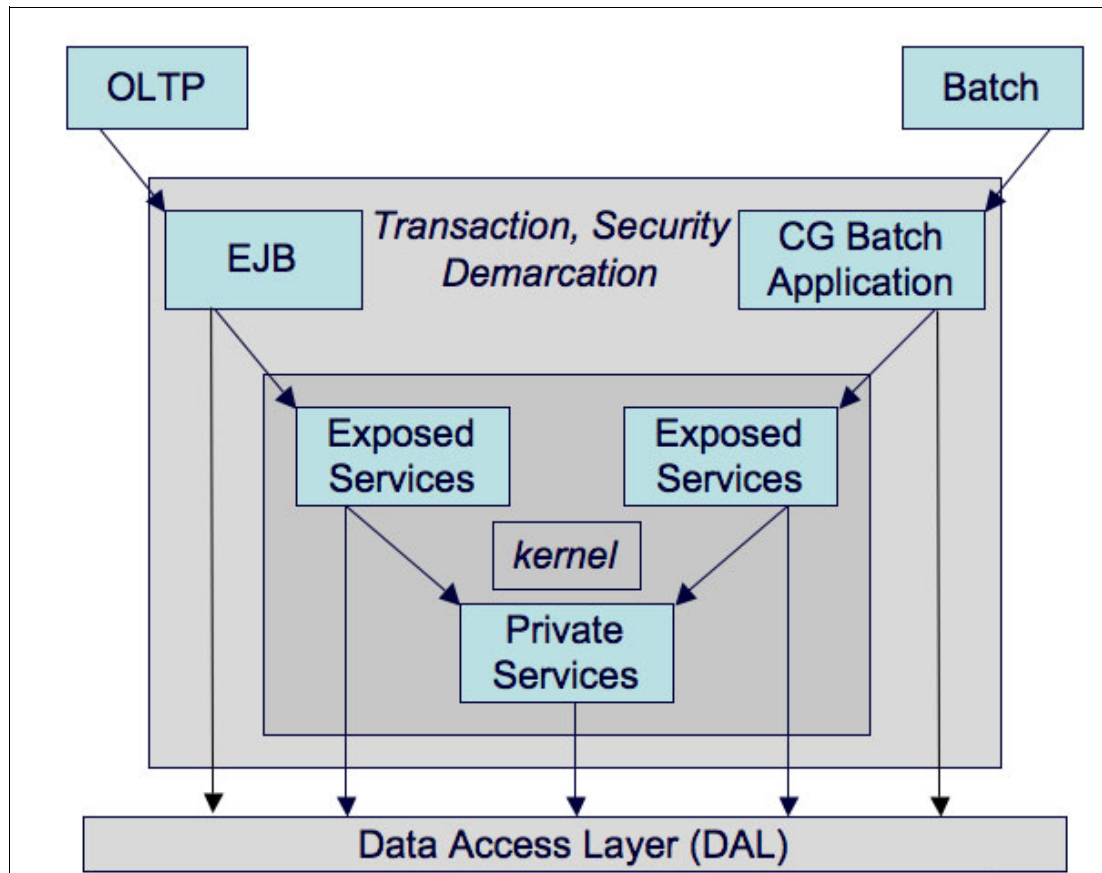


Figure 17-8 The role of containers when sharing business services across batch and OLTP.

Services shared across OLTP and batch should be written to assume that there is a transaction context and a security context on the thread of execution. It is then the burden of the execution container to enforce transactional and security semantics.

The following example demonstrates how to share services across batch and OLTP. This example shares an account reconciliation service across batch and OLTP, where the BDS Framework is used to integrate the service with the WebSphere XD Compute Grid batch container. To share the service, the application architecture is divided into several sections. The application kernel is composed of the domain object, the record-processor that is the shared business service, and the several business validations. The kernel application is then wrapped by code specific to each execution paradigm: a batch step wrapper that connects batch data-streams to the kernel service; and an OLTP wrapper that interacts with the data-access layer to retrieve a single business record to be processed. Figure 17-9 depicts the kernel, composed of the shared service and shared business validations.

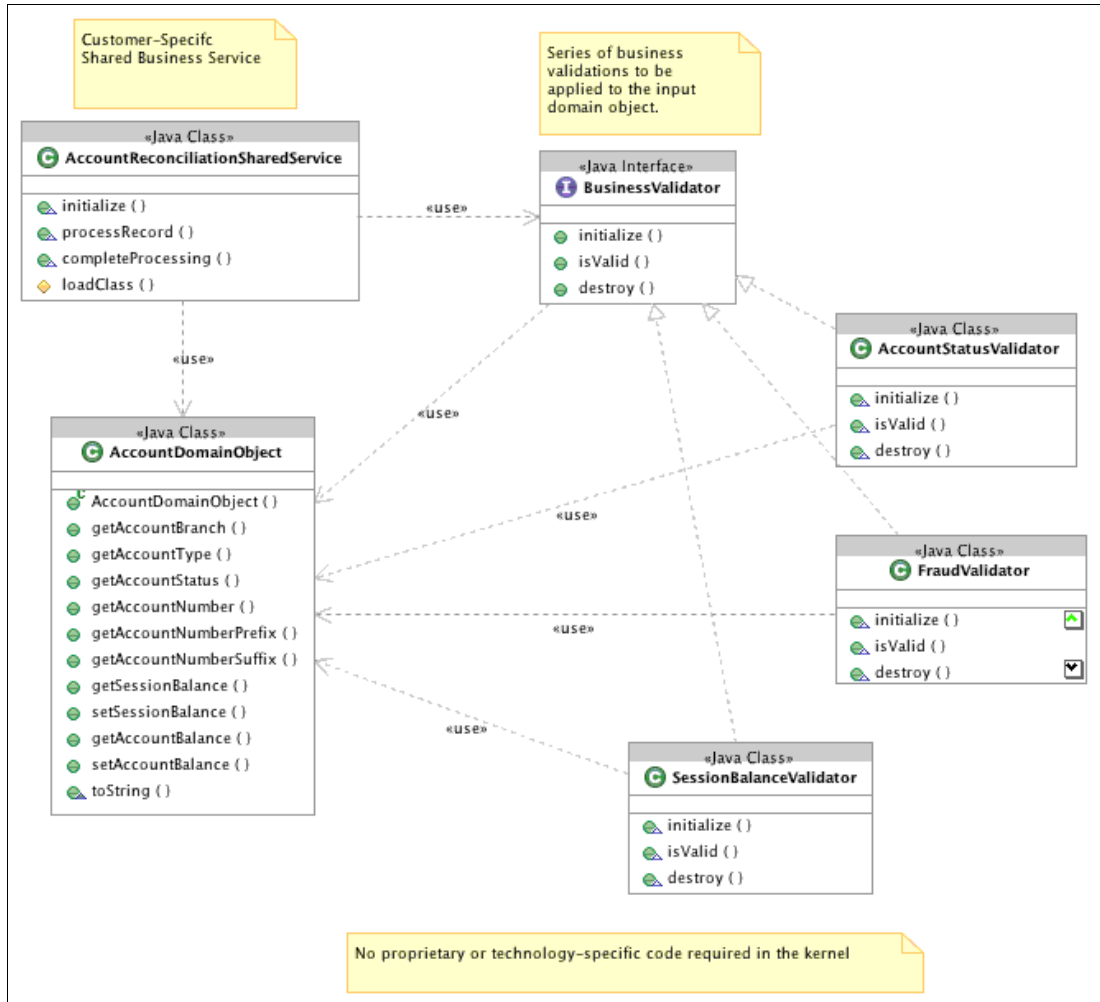


Figure 17-9 Class diagram illustrating the kernel of the shared-service

The shared service, `AccountReconciliationSharedService` in this case, is designed to be indifferent of the source of the data to be processed; the service accepts a business record (a domain object), applies business validations to that business object, and either executes the business logic or signals to the caller that the record was not processed.

Example 17-2 illustrates an example of a shared-service implementation, where the method `Object processRecord(Object record)` is the service method invoked from both batch and OLTP mode.

Example 17-2 Source code for the shared service

```
package com.ibm.websphere.batch.samples.sharedservice.steps;

import java.math.BigDecimal;
import java.util.Properties;

import com.ibm.websphere.batch.devframework.steps.technologyadapters.BatchRecordProcessor;
import com.ibm.websphere.batch.samples.sharedservice.domainobjects.AccountDomainObject;
import com.ibm.websphere.batch.samples.sharedservice.validators.BusinessValidator;

public class AccountReconciliationSharedService implements BatchRecordProcessor {

    public static final String accountStatusValidatorKey = "ACCOUNT_STATUS_VALIDATOR";
    public static final String fraudValidatorKey = "FRAUD_VALIDATOR";
    public static final String sessionBalanceValidatorKey = "SESSION_BALANCE_VALIDATOR";

    protected BusinessValidator accountStatusValidator;
    protected BusinessValidator fraudValidator;
    protected BusinessValidator sessionBalanceValidator;

    protected String defaultAccountStatusValidator =
"com.ibm.websphere.batch.samples.sharedservice.validators.AccountStatusValidator";
    protected String defaultFraudValidator = "com.ibm.websphere.batch.samples.sharedservice.validators.FraudValidator";
    protected String defaultSessionBalanceValidator =
"com.ibm.websphere.batch.samples.sharedservice.validators.SessionBalanceValidator";

    public void initialize(Properties props) {

        String accountStatusValidatorImplClass = props.getProperty(accountStatusValidatorKey,
defaultAccountStatusValidator);
        accountStatusValidator = (BusinessValidator) this.loadClass(accountStatusValidatorImplClass);
        accountStatusValidator.initialize(props);

        String fraudValidatorImplClass = props.getProperty(fraudValidatorKey, defaultFraudValidator);
        fraudValidator = (BusinessValidator) this.loadClass(fraudValidatorImplClass);
        fraudValidator.initialize(props);

        String sessionBalanceValidatorImplClass = props.getProperty(sessionBalanceValidatorKey,
defaultSessionBalanceValidator);
        sessionBalanceValidator = (BusinessValidator) this.loadClass(sessionBalanceValidatorImplClass);
        sessionBalanceValidator.initialize(props);

    }

    public Object processRecord(Object record) throws Exception {
        AccountDomainObject obj = (AccountDomainObject) record;

        if ((accountStatusValidator.isValid(obj)) && (this.fraudValidator.isValid(obj))&&
(this.sessionBalanceValidator.isValid(obj)))
        {
            BigDecimal sum= obj.getSessionBalance().add(obj.getAccountBalance());
            obj.setAccountBalance(sum);
            obj.setSessionBalance(BigDecimal.ZERO);
            return obj;
        }
        else return null;

    }

    public int completeProcessing() {
```

```

accountStatusValidator.destroy();
fraudValidator.destroy();
sessionBalanceValidator.destroy();
return 0;
}

protected Object loadClass(String className)
{
    try
    {
        Object retVal = Thread.currentThread().getContextClassLoader().loadClass(className).newInstance();
        return retVal;
    }
    catch (Throwable t)
    {
        throw new RuntimeException(t);
    }
}
}
}

```

When processing in batch mode, the Batch Data Stream (BDS) Framework connects the data reader and writer batch data streams to the batch record processor. Figure 17-10 illustrates the batch components of the application.

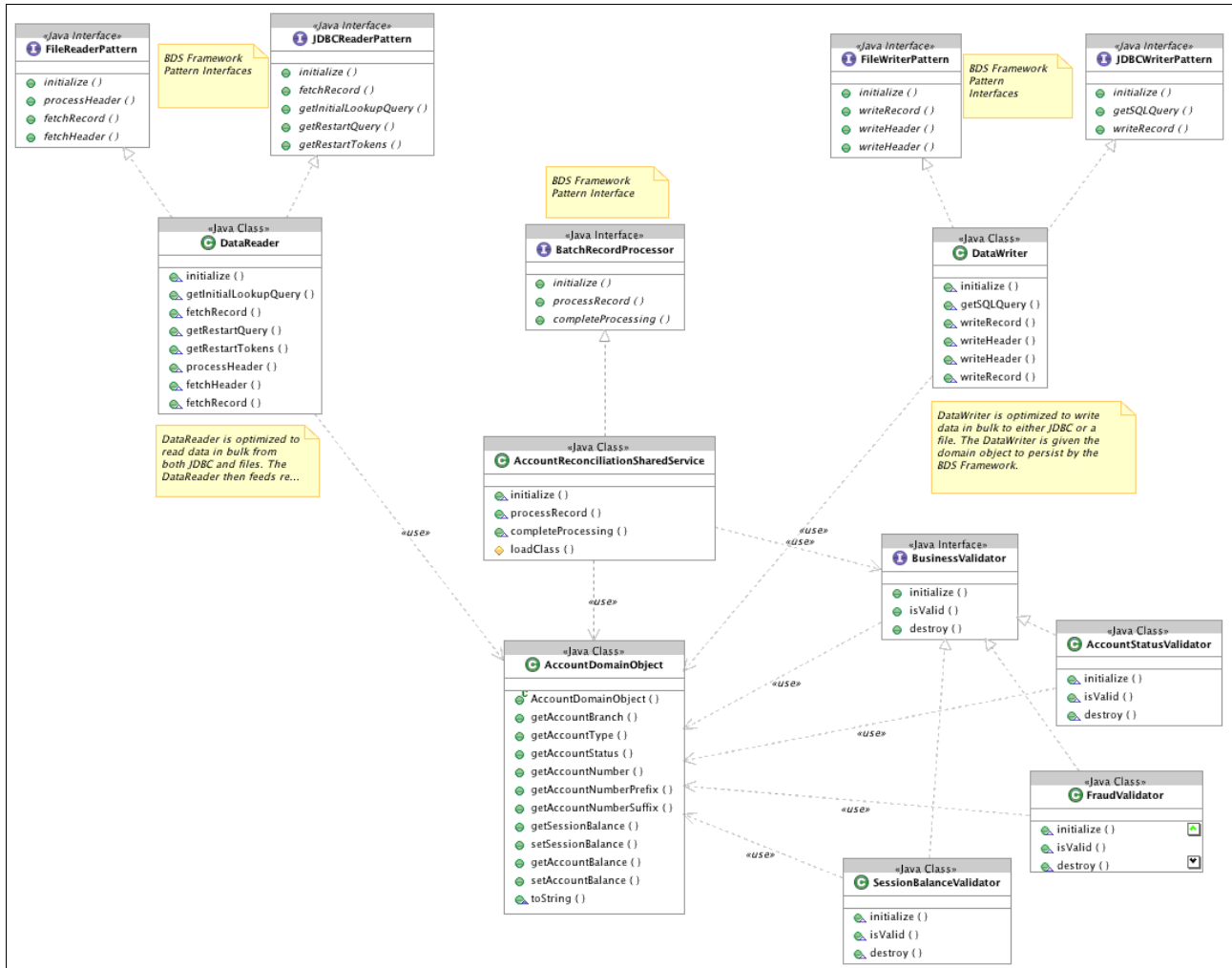


Figure 17-10 Class diagram for the batch wrappers to the shared service

The classes `DataReader` and `DataWriter` implement the relevant reader and writer patterns from the BDS Framework. The specific roles of the classes used in batch mode are:

- ▶ `DataReader`

The role of this class is to map the raw data to the `AccountDomainObject`. The input can be from one of two types of sources: a file and a database. Therefore this data reader must implement both the `FileReaderPattern` and `JDBCReaderPattern` interfaces from the BDS Framework.

- ▶ `AccountReconciliationSharedService`

The role of this class is to process each batch record which are `AccountDomainObjects`. Each batch record must be validated; `AccountReconciliationSharedService`, therefore, passes the `AccountDomainObject` to each validator specified. If any validations fail, `NULL` is returned to the BDS Framework. If all validations pass, the account balance of the domain object is reconciled and processed record returned to the BDS Framework, which will write the output to the `outputStream`.

- ▶ `DataWriter`

The role of this class is to map the domain object to raw data and write to the appropriate location. In the case of this batch step, the output type can either be a database or a file, therefore `SalDOSDataWriter` implements both the `JDBCWriterPattern` and `FileWriterPattern` interfaces in the BDS Framework.

When operating in OLTP mode, the shared service is wrapped with code whose purpose is to interface with the OLTP Data Access Layer (DAL) to retrieve the relevant business record to be processed. The OLTP wrapper then invokes the shared service, passing the acquired business record as a parameter, similar to how the service would be invoked in batch mode. The important difference here is that when in batch mode, the input and output streams are optimized to read and write sequential data in bulk. When in OLTP mode, the DAL is optimized to read and write a single, random business record.

Figure 17-11 illustrates the OLTP wrapper to the shared service.

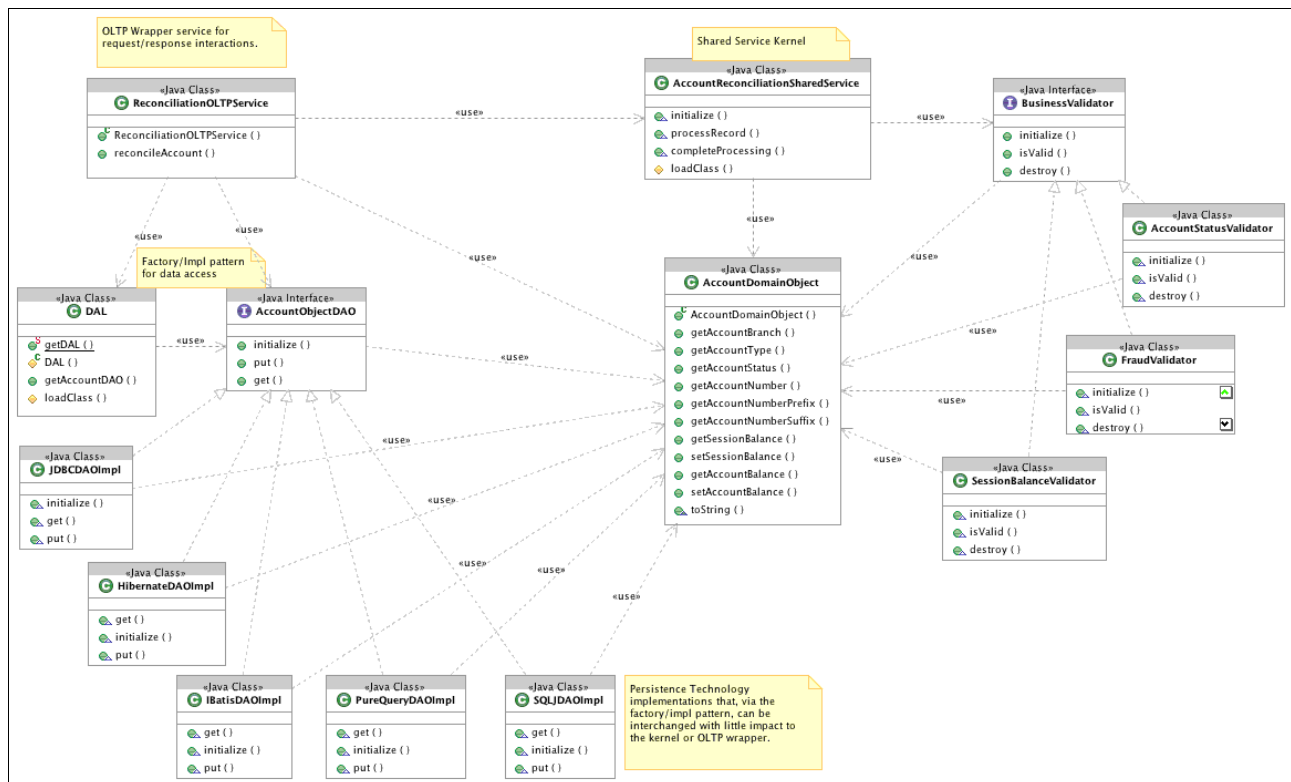


Figure 17-11 Class diagram illustrating the OLTP wrapper to the shared service

As shown in this diagram, the AccountReconciliationSharedService OLTP wrapper interacts with the Data Access Layer (DAL) and the AccountObjectDAO Data Access Object (DAO). Using the common factory/impl design pattern, we are able to interchange the various persistence technologies (JDBC DAO, Hibernate DAO, Pure Query DAO, and so forth) in a way that is transparent to the rest of the application. Example 17-3 describes the OLTP wrapper implementation.

Example 17-3 Source code for the OLTP wrapper to the shared service

```

package com.ibm.websphere.batch.samples.sharedservice.services;

import java.util.Properties;

import com.ibm.websphere.batch.samples.sharedservice.dal.AccountObjectDAO;
import com.ibm.websphere.batch.samples.sharedservice.dal.DAL;
import com.ibm.websphere.batch.samples.sharedservice.domainobjects.AccountDomainObject;
import com.ibm.websphere.batch.samples.sharedservice.steps.AccountReconciliationSharedService;

public class ReconciliationOLTPService {

    protected AccountObjectDAO accountDAO;
    protected AccountReconciliationSharedService reconcileService;

    public ReconciliationOLTPService()
    {
        accountDAO = DAL.getDAL().getAccountDAO();
        reconcileService = new AccountReconciliationSharedService();
        reconcileService.initialize(new Properties());
    }

    public boolean reconcileAccount(String accountId)

```



```

    {
        try
        {
            AccountDomainObject obj = accountDAO.get(accountId);
            obj = (AccountDomainObject) reconcileService.processRecord(obj);
            if (obj == null)
                return false;
            else
            {
                accountDAO.put(obj);
                return true;
            }
        }
        catch (Throwable t)
        {
            throw new RuntimeException("Failed to reconcile account. " + t);
        }
    }
}

```

The service, upon initialization, gets a handle to the DAL and the configured DAO. The `reconcileAccount()` method is then invoked in an OLTP context; where the service retrieves the business record to reconcile from the DAO using the primary key passed to it. That business record, which is the shared domain object, is then passed to the shared business service in the same way the batch wrapper would hand a business record. As previously stated, the shared service was written to be indifferent about the source of the business record to be processed; therefore, the same business validations and logic are used from both the OLTP and batch contexts.

Figure 17-12 depicts the big picture, the OLTP and batch domains, for the application.

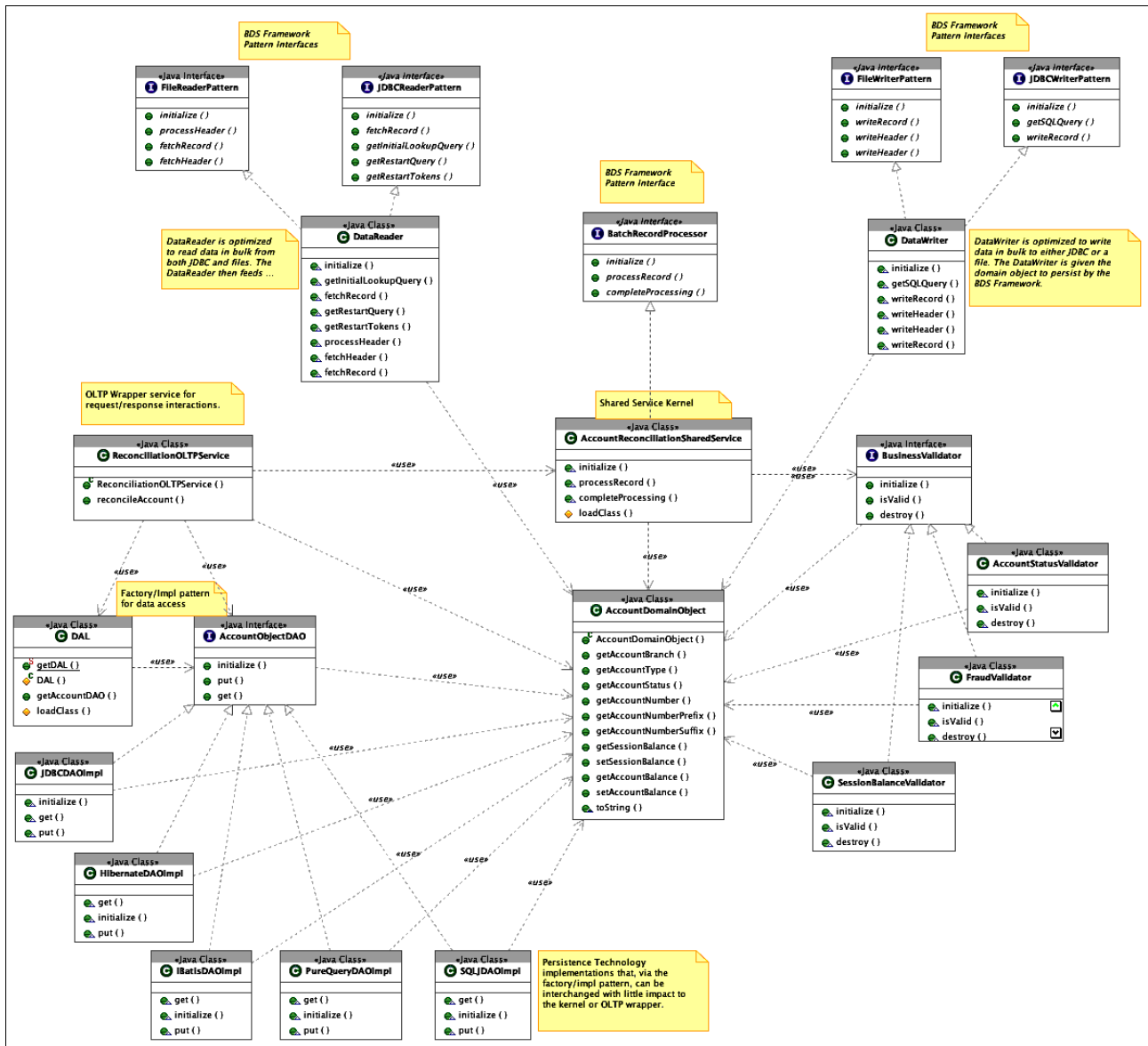


Figure 17-12 Class diagram illustrating the entire application (OLTP and Batch wrappers to the shared service)

Ultimately, the domain object becomes critical to this pattern for shared services. In this example, the same domain object is used across all of the processing, but that isn't a requirement. The reader (batch data reader or the OLTP DAL) can acquire an input domain object that is to be passed to the shared service for processing. The shared service can apply its validations to the input domain-object, apply business logic or whatever other transformations, and return to the caller an output domain-object. The batch data writer (or the OLTP DAL) will then persist the output domain-object.

A potential optimization can be made to reduce the number of validations executed when in batch mode. Specifically, more optimized queries could replace some (or all) of the business validations. For example, when in OLTP mode, a validation can be executed that asserts the account to be processed is indeed "active". When executing the service in batch mode however, a more optimized query, where a *where* clause is added limiting the data selection to only "active" accounts could be execute (for example `select * from table 1 where`

`account_status = 'active'`). In this case, the validations registered with the shared service would be different; this only implies that the technique for initializing the shared service should be smarter, where the required business validations are configured upon initialization (or can be dynamically modified).

17.2 Conclusions

By following the proposed application design guidelines, it is possible to not only share business services across multiple execution styles, but doing so without sacrificing any of the performance optimizations each domain provides. More over, well-designed applications enable flexibility, where services can be reused and composed into new business functions. To achieve this requires strong application architecture skills, as well as a deep understanding of the nuances of each execution paradigm—batch, OLTP, real-time, and so forth.



Java performance best practices

In this chapter, we discuss certain important aspects with respect to performance of Java in batch. It includes the following topics:

- ▶ Java performance in common
- ▶ Stand-alone Java batch

18.1 Java performance in common

In this section, we discuss performance aspects that affect Java on z/OS in common.

18.1.1 Garbage collection

An important aspect from a Java performance point of view is the garbage collection. If the garbage collection takes too much time of the CPU time, we have to tune it. As this is very well explained in the Java 6.0 Diagnostics Guide we, do not discuss this in more detail here. The Java 6.0 Diagnostics Guide can be found at:

<http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/diagnosis/diag60.pdf>

There is also an article at developerWorks at:

http://www.ibm.com/developerworks/java/library/j-ibmjava3/index.html?S_TACT=105AGX02&S_CMP=EDU

Note: We saw that the gencon policy can be very attractive in many cases for Java batch applications because batch applications tend to create many objects (for example a data objects) just for a short time. Of course, this heavily depends on the application and has to be verified with garbage collection policy comparisons in each situations by an analysis of the verbose:gc trace.

Another way to reduce garbage collection in batch is to reduce object creation by reusing objects.

New Java 6 features, such as compressed references and large pages, significantly reduce the cost of garbage collection. See:

http://www.ibm.com/partnerworld/wps/whitepaper/systemz/java_websphere/performance

18.1.2 Profiling

Besides JVM tuning, we can achieve a lot of in tuning the Java batch application itself. Therefore, we have to analyze the Java application at run time with a profiler. A very efficient profiler is JinsightLive for IBM System which is available for free under

<http://www.alphaworks.ibm.com/tech/jinsightlive>

JinsightLive allows us to create a profile trace of a running Java application under z/OS. With the help of the visualizer, we can then analyze the trace with graphical representation of the application.

Based on patterns like a long running loop that we can see in the graphics, we can identify hot spots that consume too much CPU. Based on these findings, we next have to verify whether those areas cannot be coded more efficiently.

Note: It can be useful for someone who did not write the Java code to analyze the code because the developer might not see certain hot spots because of reasons for coding it in that way.

For further information about how to use JinsightLive, refer to the Web site.

Tip: In contrast to OLTP J2EE / Java EE applications where we normally only trace one transaction for profiling with JinsightLive, we create a trace of the whole application in batch. Those traces can become very huge!

Because batch often consists of repetitive patterns implemented in loops, we recommend to modify the application for tracing purposes to run just one iteration of a loop.

18.2 Stand-alone Java batch

In this section, we talk about stand-alone Java batch specific performance aspects which is mainly the impact of JVM startup costs.

18.2.1 JVM startup cost

In each stand-alone Java step a JVM is created, regardless of the launcher we are using. For a simple HelloWorld, we made the following measurements:

Table 18-1 JVM Startup cost

JVM version	Total TCB CPU Time (in CPU minutes)
5.0 SR5	0.02
6.0 SR3	0.02

Because batch applications are normally long running jobs, this CPU time is negligible. It only becomes a problem for very short running stand-alone Java batch applications. In that case, a Java batch container like WebSphere XD Compute Grid helps because it reuses its JVMs for the batch applications.

We mention the JIT, shared classes, Ahead of Time (AOT) compilation and JNI as areas to look into for optimizing performance.

Important: There is “one size fits all” solution and different applications might benefit from different settings.

Just In Time (JIT) compiler

Another aspect is the Just In Time compiler (JIT). The JIT needs a certain time to optimize code. If we know that we have a very short running stand-alone Java batch application, we can use the following JVM parameter:

```
-XQuickstart
```

This parameter causes the JIT compiler to run with a subset of optimizations. This quicker compilation allows for improved startup time for short running applications.

Also, putting the JIT DLL in the LPA has shown to give around 5% improvement in HelloWorld startup.

Shared classes

Another way to improve JVM startup time is to use *shared classes*. The shared class cache is intended to share classes across multiple JVMs. Because it persists across JVM restarts, it

can also help to reduce startup time of a single JVM.

When the JVM is started the first time, it creates the shared class cache identified by a name as a JVM parameter. The next time it is started, we can tell the JVM to use this shared class cache again by telling it the name of the already existing cache.

For more information about how to use shared classes see:

http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/topic/com.ibm.java.doc.user.zos.60/user/shc_overview.html

Ahead of Time (AOT) compilation

Also, as of Java 6, one can use Ahead-of-time (AOT) compilation. This is similar to JIT compilation, but it does not take runtime information into account. But the big advantage is, that the AOT compiled code can be stored in the shared class cache. Therefore, it survives JVM restarts and helps to decrease the warm-up costs for the JVM.

Note: Although AOT compilation will show improvements over interpreted code, we do not recommend to disable the JIT compiler. The AOT compiled code might still be JITed to higher levels of optimization.


The following developerWorks article talks about AOT compilation, along with shared classes usage:

http://www.ibm.com/developerworks/java/library/j-sharedclasses/index.html?S_TACT=105AGX02&S_CMP=EDU#N1010B

Java Native Interface (JNI) best practices

Batch applications tend to use JNI significantly to access existing code. The following developerWorks article provides best practices that are helpful in improving performance.

http://www.ibm.com/developerworks/java/library/j-jni/index.html?S_TACT=105AGX02&S_CMP=EDU



Increasing batch efficiency by using performance instrumentation

This chapter outlines the types of instrumentation available for tuning batch and managing its performance, whether window batch or non-window batch. With so much instrumentation available, as this chapter demonstrates, it is important to structure its use in a way that is helpful to any batch performance improvement project. The following sections are sequenced accordingly:

- ▶ System-level and WLM workload SMF
- ▶ DB2 Subsystem-level instrumentation
- ▶ Batch suite instrumentation
- ▶ Job-Level SMF
- ▶ Other job-level instrumentation

Of course, some of these types of instrumentation might be irrelevant, for example if your batch does not run against DB2. Another example is the case of batch which is not part of a specific suite.

Note: This chapter mainly deals with raw sources of instrumentation, such as Systems Management Facilities (SMF) records. Reporting tools, such as the RMF Post-processor, are only briefly mentioned.

In this chapter, the following example illustrates the convention used:

“74-1” denotes “SMF Type 74 Subtype 1 records”.

This chapter includes the following topics:

- ▶ System-level and WLM workload SMF
- ▶ DB2 Subsystem-level instrumentation
- ▶ Batch suite instrumentation
- ▶ Job-Level SMF
- ▶ Other job-level instrumentation

19.1 System-level and WLM workload SMF

As with examining other workloads it is important to look at system-level instrumentation. Similarly, with relatively minor differences, batch shares WLM Workload and Service Class level instrumentation.

The standard system-level instrumentation comes from RMF and can be formatted into reports using the RMF Post-processor.

Table 19-1 shows the main SMF record types used for system- and workload-level analysis. There are other RMF-written SMF record types but these are the major ones for analyzing the environment in which batch runs.

Table 19-1 SMF records used for system- and workload-level tuning

Record Type	Area	Description
70	CPU	CPU at the z/OS image level, including zIIP and zAAP. LPAR descriptions, LPAR CPU time and LPAR memory allocation (but not memory use) Processor complex information
71	Memory and Paging	Memory use at the z/OS image level and paging information at the z/OS image level.
72	Workload	information about goal attainment and memory and CPU usage at the Service Class Period level. CPU information includes zIIP and zAAP (as well as crossover information).
74-1, 74-5 and 74-8	Disk, Cache and Tape	74-1 gives basic response time and I/O rate information at the volume level - for both disk and tape volumes. 74-5 gives disk cache information at the volume level. 74-8 gives detailed cache controller information.
74-2 and 74-4	XCF and Coupling Facility	74-2 gives XCF information at the XCF member, transport class and path level. 74-4 gives Coupling Facility information at the Coupling Facility, path and structure levels.
75	Page Data Set Activity	Page data set activity at the individual page data set level.
78 Subtype 3	I/O Queuing	Gives information about channel connectivity to disk and tape control units and the degree of I/O request queuing to each control unit.

RMF writes records on the RMF interval. So you can plot the major numbers on a graph by time of day across the window.

Another layer below the Service Class level is the address space level. “Job-Level SMF” on page 346 discusses jobs but the 30-2 and 30-3 records instrument address spaces in a way which can best be described as allowing “drill down” further from the Type 72 level. Because SMF 30-2 and 30-3 have WLM Workload and Service Class in you can break down a Service Class’ CPU time into individual address space CPU time. In the batch context this is particularly useful for understanding which started tasks are consuming CPU and how this varies through the batch window.

SMF 30-2 and 30-3 also record the Report Class the address space runs in.

19.2 DB2 Subsystem-level instrumentation

Tuning a DB2 subsystem is often the simplest method for speeding up the batch jobs which use it. For example, increasing the buffer pool sizes overnight when other applications do not need the memory can often speed up many batch jobs with one simple action.

In Figure 19-1 the amount of memory used by other applications (the “Other Apps” data series) varies by time of day as the workload increases and decreases. Overnight the other applications use much less memory than during the day. Accordingly the DB2 ALTER BUFFERPOOL command is used in the evening to increase the size of each of the three buffer pools first from 3 GB to 4 GB and later to 5 GB. In the morning the buffer pool sizes are reduced in two stages from 5 GB each to 3 GB each.

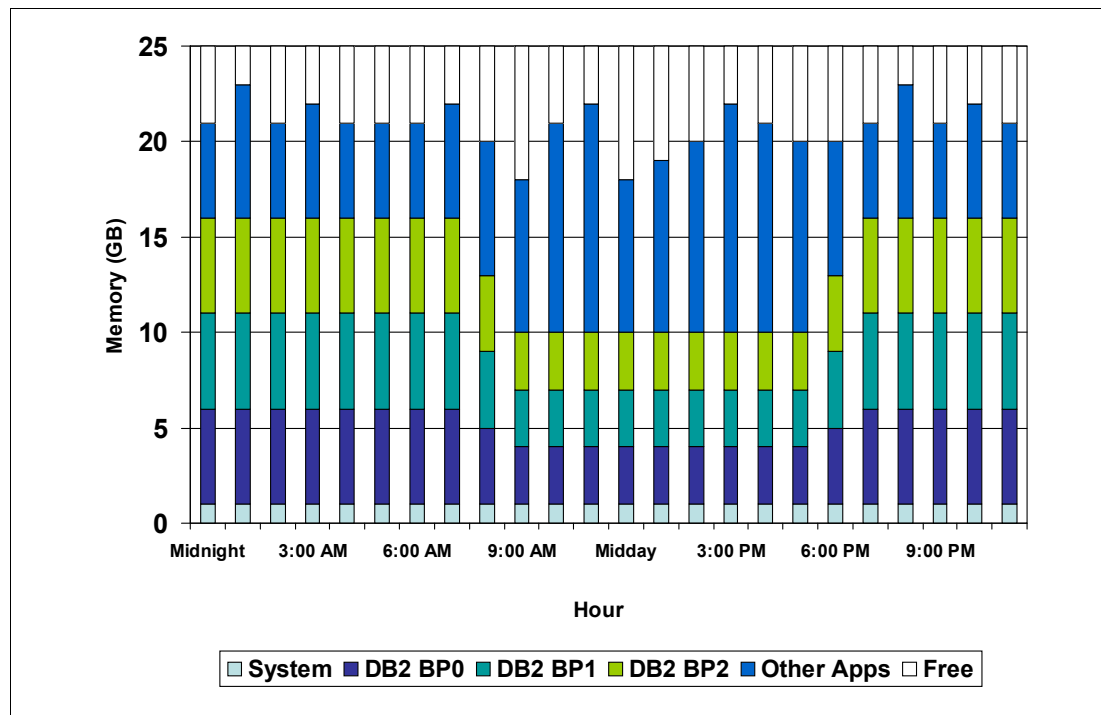


Figure 19-1 Buffer Pool Memory Sizes Changing As System Conditions Permit

However, to conclude increasing buffer pool sizes (or making any other tuning change) needs instrumentation. For DB2 the main subsystem-level instrumentation is *Statistics Trace*. Additional information can be found in the DB2 Catalog. Still more information related to the subsystem (but not necessarily the application) comes from other SMF record types. Most prominent among these non-DB2 SMF record types is 42-6.

19.2.1 DB2 Statistics Trace

DB2 provides information at the subsystem level using *Statistics Trace*, which appears in SMF on an interval basis as Type 100 and Type 102 records. The interval length is given by the STATIME DSNZPARM parameter. Originally the default value for STATIME was 30 minutes (an almost totally useless default). From DB2 Version 8 onwards the default is (a much more useful) 5 minutes.

Fields which show counts (known as “counters”) are cumulative. That is it is necessary to subtract the value for the field in two successive records to obtain the count of activities in the STATIME interval. Reporting tools, such as DB2 Performance Experts Statistics Report, perform the subtraction before reporting the count for that interval.

Fields which are snapshots can be used directly.

Some of the areas of DB2 activity instrumented by Statistics Trace are:

- ▶ Buffer Pools
- ▶ Group Buffer Pools
- ▶ EDM Pool
- ▶ Relational Data System (RDS) Sorting
- ▶ Logging
- ▶ Locking
- ▶ Virtual Storage

While Statistics Trace provides a useful set of instrumentation for general subsystem tuning it should be supplemented with application-level instrumentation, most notably Accounting Trace. (See 19.4.2, “DB2 job-level Accounting Trace and deeper” on page 348 for more information about Accounting Trace.).

19.2.2 DB2 Catalog

The DB2 Catalog is a set of critically-important DB2 tables, controlling many aspects of a DB2 subsystems’ operations. There is one Catalog shared by all members of a DB2 Data Sharing Group.

The Catalog contains such information as:

- ▶ Table and index definitions, including columns and keys
- ▶ Tablespace and indexspace definitions, including the buffer pool number the table space or index space is in and partitioning information
- ▶ View definitions
- ▶ Plan and package definitions, including SQL statements for Static SQL
- ▶ Stored procedure, user-defined function (UDF) and trigger definitions
- ▶ Utility run information

This information can be useful in studying a DB2 subsystem environment and the applications that run in it.

19.2.3 SMF 42-6

SMF 42-6 records are written on the SMF interval (by default every 30 minutes). A group of such records are written for each address space, instrumenting every data set OPENed by

the address space in the interval for which at least 1 I/O was performed. More precisely, every such data set's volume for which at least 1 I/O was performed.

The information includes:

- ▶ Data set name and volume serial number (volser)
- ▶ I/O count
- ▶ Response time components - Connect, Disconnect, Pend, IOSQ
- ▶ Estimates of cache hits and misses

These items make the 42-6 a very valuable SMF record for data set performance analysis.

Note: SMF 42-6 records are not created by DB2 but rather by DFSMS/MVS. So, while they are written for DB2-owned data sets, they are also written for non-DB2 data sets.

19.2.4 Putting Statistics Trace, DB2 Catalog and SMF 42-6 together

DB2 Statistics Trace instruments a number of key areas where data set access is important for performance, which includes:

- ▶ Buffer pools and group buffer pools
- ▶ EDM Pool
- ▶ Sort work pool
- ▶ Logging

Suppose you need to explain the I/O rate to disk for a specific buffer pool. (Statistics Trace might lead you to the conclusion that this needs an explanation.)

You can use the DB2 Catalog to ascertain which tablespaces and indexspaces are in this buffer pool. (You can get other information about these DB2 objects, such as partitioning information and DB2's view of their size, from the Catalog.)

Using DB2's data set naming convention for tablespaces and indexspaces and the address space name for the DBM1 address space for the DB2 subsystems you can select the SMF 42-6 data for just those data sets, including the I/O rate. (You can get other information as well, such as DFSMS' estimation of the effectiveness of disk cache for the data sets.)

Note: In addition to the SMF 42-6 information for the DB2 object you can use the IDCAMS DCOLLECT information for the data sets associated with the object to obtain the DFSMS/MVS view of the space used by the DB2 object.

19.3 Batch suite instrumentation

SMF 30-5 records give comprehensive information about individual job runnings. This information includes:

- ▶ When the job ran
- ▶ On which system the job ran
- ▶ How much CPU the job used, including zAAP time, zIIP time and eligible CPU in both categories but which was executed on a general-purpose processor instead
- ▶ How many EXCPs were performed and (using some analysis) how many were to disk and how many to tape

- ▶ Whether the job ended successfully, in so far as we can tell from return codes and ABEND codes
- ▶ The difference in timestamps for reading in and execution, which would enable an analyst to discern whether there is a shortage of initiators.

This information enables several different kinds of analysis. Among them are:

- ▶ Drawing a Gantt chart of an application based on job names and run times
- ▶ “Toplisting” jobs, such as the most CPU-burning jobs or the longest-running jobs, or the jobs that do the most tape processing (whether by mounts or tape EXCPs).

19.4 Job-Level SMF

As indicated in 19.3, “Batch suite instrumentation” on page 345, you can obtain lots of information about job runnings using SMF 30-5.

But you can drive this down further - to the step level - using SMF 30-4. As well as most of the information available at job level in SMF 30-5 you can obtain such information as:

- ▶ When the step ran and what resources it consumed
- ▶ What the step-level program was
- ▶ Step return code

In addition to understanding the sequence of steps in a particular job you can perform analyses such as:

- ▶ Which are the most CPU-intensive or longest-running steps
- ▶ Which steps use IDCAMS
- ▶ Which steps might use DB2 (because the program name is IKJEFT1B, although that might equally denote a REXX step)

19.4.1 Data Set OPENS And CLOSEs

Both VSAM and non-VSAM data set OPEN and CLOSE information can be obtained for a job step. The VSAM and non-VSAM cases are different:

- ▶ For VSAM data sets there are two distinct record types for OPEN and CLOSE events. OPEN events are recorded in SMF 62 records. CLOSE events are recorded in SMF 64 records. Neither 62 nor 64 records contain the step number or name. To identify the step you need to perform time stamp analysis with SMF 30-4 records, using the job name as a key.
- ▶ For non-VSAM data sets there are again two record types: SMF 14 for read and SMF 15 for write. Both 14 and 15 records encompass both the OPEN and the CLOSE event for a data set. Again there is no step information so a merge is required with SMF 30-4 records, with job name as the key.

These four record types provide a wealth of information for conventional data sets but this information is different (and differently organized) between the VSAM and non-VSAM cases.

For example, the Type 62 and 64 records contain very useful information about VSAM access patterns and buffering.

As indicated in 19.2.3, “SMF 42-6” on page 344 you can obtain performance information about an interval basis for a disk data set from DFSMS’ perspective using SMF 42-6. You can

also obtain this information for when a disk data set is CLOSEd, again from SMF 42-6 records. 42-6 information is a very useful addition to that in Types 14, 15, 62, and 64.

Life Of A Data Set (LOADS)

Life Of A Data Set (LOADS) is a technique for using the OPEN and CLOSE signatures of data sets to discern technologies that might speed up the jobs that use the data set, and perhaps allow overlap.

Here are some examples:

- ▶ For a sequential data set a single writer of the entire data set followed by a single reader of the entire data set might suggest a BatchPipes/MVS pipe.
- ▶ Repeated DFSORT operations against the same input data set might indicate OUTFIL to be a useful technique.
- ▶ A VSAM data set where the processing is not conducive to any other technique but where the statistics in the Type 62 and 64 records for the data set suggest VSAM LSR buffering might suggest Batch LSR or System-Managed VSAM Buffering.

Using a Gantt chart for the LOADS depiction of a data set might reveal such patterns.

Figure 19-2 on page 348 depicts such a Gantt chart. In this example the life of the data set MYJOB.PAYROLL.LEDGER.FILE is depicted. It is a 5 million block tape data set. The tape block size is 32 KB. The Gantt chart shows Job W writing the data set, and Job R1 and Job R2 reading the data set. (From EXCP counts or perhaps the use of a known utility such as DFSORT COPY it is often possible to tell the data set is read **in its entirety** both times.)

This is the complete life of the data set (other than the recycling of the tapes at some possible distant point) so we can think of speed ups that only involve these three jobs. If it were possible, because of what we know about the jobs, to overlap them if we could remove the dependencies this data set creates we would be able to run a lot faster.

One such solution is to pipe separate copies of the data set from Job W to Job R1 and Job R2. Another, less beneficial, technique might be to overlap Job R1 and Job R2 by some kind of “shared reading” implementation.

Note: There is a gap between when Job R1 finishes with the tapes and when Job R2 starts with them. This should be investigated. If this delay cannot be removed and so Job R2 cannot be overlapped with Job R1 it might still be beneficial to overlap Job R1 with Job W1. In this case you would arrange for a second copy to be written in parallel to tape.

There are potentially other speed ups. For brevity, we do not cover those here.

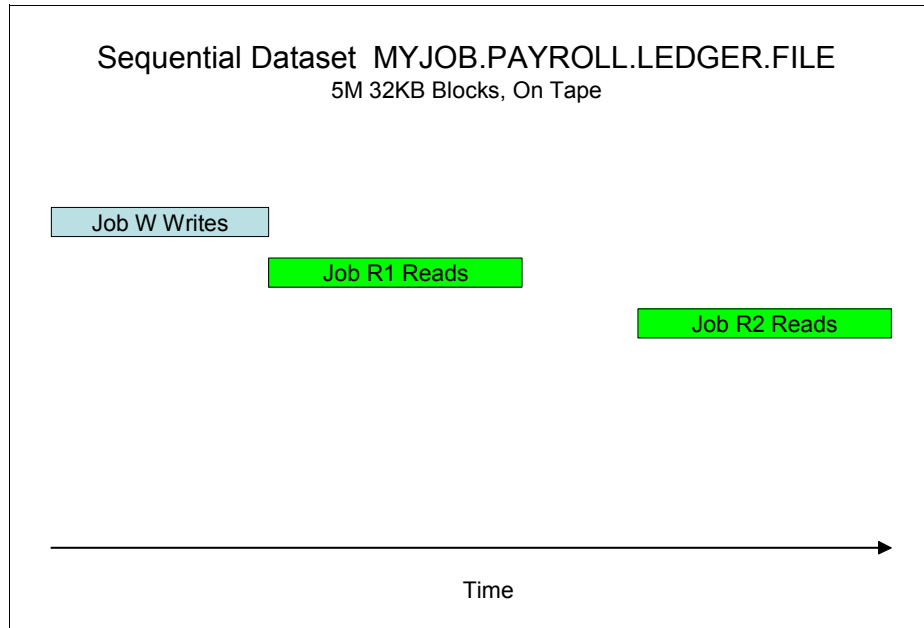


Figure 19-2 Example Of LOADS For A Sequential Tape Data Set

This example shows some of the power of the LOADS technique in designing speed ups.

19.4.2 DB2 job-level Accounting Trace and deeper

When enabled, DB2 writes information about run times to Accounting Trace (externalized as SMF 101). For useful DB2 batch performance analysis trace classes 1, 2, and 3 must be enabled. DB2 can write package-level information if, in addition, trace classes 7 and 8 are enabled.

At the plan level the 101 record has around 20 different buckets for where time is spent. At the package level a similar number of buckets are available. A very good depiction of DB2 batch step run time is to create a grid, with time buckets across the top and plan and package names down the side. With this depiction it is easy to conclude such things as “I need to work on locking in Package PK1 in Plan PL3” and “for all plans and packages in this step I need to buffer data better”.

At the plan invocation level you can examine buffer pool statistics at the individual buffer pool level. This enables you to draw conclusions like “the majority of the jobs I care about might be expected to benefit if Buffer Pool 3 were increased” or “Package PK2 in Plan PL2 spends a lot of time accessing data on disk”.

Time buckets include:

- ▶ CPU, whether in DB2 or in the application (when the application has connected to DB2)
- ▶ I/O time, whether write or read, and whether synchronous or asynchronous-but-not-overlapped.
- ▶ Waiting for locks or latches
- ▶ Waiting for miscellaneous DB2 services

The *unknown time*, which is the time that you get when you subtract the known time buckets from the overall time, can have many causes, but the most common cause is CPU queuing and paging.

Accounting Trace has information about Data Sharing aspects of response time, such as waiting for Global Locks.

It is possible for a single DB2 invocation to cause multiple SMF 101 records to be produced, whether through parallelism or because there are more DB2 packages invoked than can fit in a single record.

For CPU- and sysplex-query parallelism DB2 can write detailed records, one for each parallel task, or roll up all the parallel tasks into a parent record and a rollup (child) record. For batch tuning it is sometimes useful to have the individual one-per-task records: When the tasks aren't balanced each will record different run times, so you can spot the imbalance and reduce it.

Correlating DB2 Accounting Trace and jobs and steps

DB2 Accounting Trace (SMF 101) records have a time stamp but they do not have a job name. Instead they have a correlation ID. In many cases the correlation ID is the job name. Examples of this include the TSO Attach and DB2 Utility jobs. In other cases the correlation ID is different from the job name. The most notable cases of this are IMS-originated batch. To match the correlation ID to the job name knowledge of the IMS setup is required.

Step number is not in the SMF 101 record. If you have deciphered the correlation ID you can use time stamp analysis to associate a job's SMF 101 records with their steps.

Some batch DB2 applications access DB2 with more than one plan invocation (but not at the same time) so sorting the records by time stamp within each step gives a good view.

It is also worth subtracting the DB2 elapsed times and CPU times from the step times. Sometimes there is significant additional processing outside of DB2 that is revealed by this technique.

Figure 19-3 on page 350 shows an example of how the time of a DB2 batch job can be broken down. It differs from the standard Accounting Trace (Long) Report in that it is simplified and the step-level information is interspersed with the appropriate DB2 Accounting Trace information. All timings in this example are in minutes.

In this example STEP010 is the significant step (with STEP020 much shorter). Plan PL1 is the only DB2 plan in STEP010. Within PL1's execution in STEP010 the majority of the time is in package GHJG0101. In particular CPU time (38 minutes) and Synchronous Database I/O time (31 minutes). Async Read I/O time is also significant at 10 minutes. Package HHJG0212 is also significant, using 11 minutes of CPU.

In this artificial example there is no CPU Queuing time, so the numbers do all add up to the headline totals. This is somewhat unusual and the example is constructed this way to make the more significant points clear.

Note: In this example other components of response time have been suppressed, as have other packages. The individually insignificant packages have been added together.

STEP010 – 02:35 to 04:15, Elapsed: 100 mins, CPU: 55 mins, EXCPs 1.8M

Plan PL1: 02:36 to 04:15, Elapsed: 99 mins, CPU 55 mins

Package Name	CPU Time	Sync Database I/O	Async Read I/O	Async Write I/O	Lock / Latch Wait	...
GHJG0101	38	31	10	1		
HHJG0212	11		1		1	
...	6					

STEP020 – 04:15 to 04:18, Elapsed: 3 mins, CPU: 2 mins, EXCPs 1287

Plan PL2: 04:15 to 04:18, Elapsed: 3 mins, CPU 2 mins

Figure 19-3 Sample DB2 job time breakdown report using DB2 Accounting Trace

Examining DB2 job performance more deeply

For very detailed DB2 application performance analysis enable DB2 Performance Trace. The CPU cost of Performance Trace is generally significant, it might delay the batch jobs for which it is enabled, and a large amount of SMF data is likely to be written.

So use Performance Trace sparingly, focusing on job steps where it is necessary to tune the SQL issued by the application. The information, for relevant jobs, is very valuable.

One of the most useful pieces of information Performance Trace can tell you is which SQL statement is significant (by statement number with plan or package). With this information you could use the DB2 Explain facility to analyze SQL Access Paths in detail.

You can enable Explain in two ways:

- ▶ Binding a plan or package with Explain enabled
- ▶ Explain a stand-alone SQL statement

With either method rows are placed in a special DB2 table: PLAN_TABLE. This table can be queried to obtain information about how DB2 intended to run the SQL statement. Usually several rows are produced. In skilled hands this can be very effective instrumentation for SQL tuning.

Two DB2 Catalog tables, SYSIBM.SYSSTATEMENT and SYSIBM.SYSPACKSTMT, can be queried. Given the plan name, statement number and (perhaps) package name you can extract the SQL statement text. As mentioned above, to obtain the statement number and (perhaps) package name, DB2 Performance Trace can be invaluable. (This is also true for Explain.)

Most rows in PLAN_TABLE refer to specific indexes and tables. Using the DB2 Catalog you can relate these to specific indexspaces and tablespaces. With this information you can use SMF 42-6 to look at the I/O performance for the relevant DB2-owned data sets.

Note: SMF 42-6 records will have the DB2 subsystem's DBM1 address space name in, not that of the batch job.

19.4.3 DFSORT

DFSORT has its own SMF record: Type 16.

An installation can specify for a SMF 16 record to be produced whenever a DFSORT operation completes (whether directly-invoked or invoked by ICETOOL). There are two kinds of SMF 16 records:

- ▶ When SMF=SHORT is in effect or if SMF=FULL is in effect but the DFSORT operation terminated abnormally.
- ▶ When SMF=FULL is in effect.

SMF=FULL is recommended for serious DFSORT tuning work as it contains additional information, such as a section for each SORTIN, SORTOUT and OUTFIL data set. While some of this information duplicates that available with data set SMF records (as outlined in 19.4.1, "Data Set OPENS And CLOSEs" on page 346) some of it is additional.

An example of additional information is the number of records read from or written to the data set. Record counts are important when tuning DFSORT activities as one of the major themes is to reduce the number of records passing through each successive processing stage.

Other examples include the exit-processing specifications, major types of DFSORT control statement (such as SUM), and whether EQUALS is specified. (EQUALS requires DFSORT to preserve the order of records which have the same sort key, which can degrade performance.)

Because the Type 16 record has the step number as well as job name it can be directly related to a specific job step. To re-create the sequence within the step, because there can be several DFSORT operations, sort the records by time stamp. JZOS and ICETOOL, as two examples, can be used to drive multiple DFSORT operations within a single step.

For sorting operations the number of intermediate merges would ideally be only 1. z/OS Release 10's DFSORT level introduces in SMF 16 the number of intermediate merges (in field ICEINMRG). If this number is greater than 1 the sort might benefit from an increased virtual storage specification. Also in Release 10 a new message became available: If additional intermediate merges are required the job log will also contain the message:

```
ICE247I INTERMEDIATE MERGE ENTERED - PERFORMANCE MAY BE DEGRADED
```

19.4.4 BatchPipes/MVS

BatchPipes/MVS has its own SMF record, Type 91, with a number of subtypes:

- ▶ Subtype 11 records the OPEN of a pipe by a job
- ▶ Subtype 12 is an interval record for that job
- ▶ Subtype 13 records the CLOSE
- ▶ Subtype 14 records pipe creation
- ▶ Subtype 15 records pipe deletion

You can use the 11 and 13 records to perform “pipe balancing” (minimizing reader-empty and writer-full wait times) as well as understanding the characteristics of a pipe. The whole pipeline topology can be reconstructed from Type 91.

Because the step number is recorded, as well as the job name, you can readily identify which step the pipe was OPENed in and whether for read or for write. Further time stamp analysis to determine the order in which pipes are OPENed and how far into the step they are OPENed and CLOSEd is useful.

19.4.5 DFSMShsm functional statistics

DFSMShsm can record activities such as data set recalls from Migration Level 2 (ML2) in its own Functional Statistics Record (FSR), written to SMF. This record is usually Type 241 but the installation can change this.

The step name and number are not in the record but can be discerned by time stamp analysis.

This record is mainly useful for understanding which DFSMShsm activities delay a job and for how long. For example the start and end of a data set recall from ML2 can be discerned. If recalls are shown to be a problem steps can be taken, such as scheduling a recall before the job needs the data set.

19.5 Other job-level instrumentation

In addition to SMF records there is other instrumentation available. Some of the major sources of information are described below.

It is also worth noting that source code for the job steps represents a very useful source of information. Indeed the discovery of a lack of source code might be the compelling event that drives an installation to re-implement portions of their batch.

19.5.1 SYSIBM.SYSCOPY for DB2 utility jobs

Most DB2 utilities insert one or more rows into the SYSIBM.SYSCOPY table in the DB2 Catalog. Each row contains, amongst other things, the job name and a time stamp. This can be used, together with SMF 30-4 Step-End record timestamps, to determine which DB2 utility executions were run in that step. This often explains the use of other facilities, such as DFSORT, and can provide opportunities to tune the step from the DB2 Utilities perspective.

19.5.2 Tivoli Workload Scheduler information

With Tivoli Workload Scheduler you can obtain information at two levels:

- ▶ Application Definitions
- ▶ Operations in Real Time

Note: Tivoli Workload Scheduler can manage operations on more than just z/OS systems. Information in this section applies to both z/OS and non-z/OS operations.

Application Definitions

The Application Description (AD) ISPF panel provides long term information about applications, as well as allowing you to change the applications' definitions. You can obtain information about such things as:

- ▶ Operations
- ▶ Dependencies
- ▶ Special resource requirements
- ▶ Run cycles
- ▶ Error tracking rules
- ▶ Operator instructions.

You can also calculate the critical path for an application.

Operations in Real Time: The current plan

The Query Current Plan (QCP) ISPF panel provides information about the current status of current production. You can request detailed or summary information about individual applications, operations, or workstations (such as z/OS images). With QCP you can also obtain summary information concerning all operations. The QCP panel looks at the current plan, which is continuously updated as the operations are processed. You can use the QCP panel to:

- ▶ Determine why an operation has not been started.
- ▶ Provide status information.
- ▶ Display a list of operations that have ended-in-error.
- ▶ Decide if intervention is required to speed up the processing of specific applications.

You can display the applications that are most critical and those that have missed, or are close to missing, the defined deadline.

- ▶ Check information before making modifications to the current plan.
- ▶ Display a list of all dependencies for an operation.

This function is of particular benefit to quickly identify which outstanding predecessors are not completed.

- ▶ Determine the impact of an operation that has ended in error.

The Tivoli Job Scheduling Console is another interactive interface (running on, for example, Windows) that enables you to monitor and control objects scheduled in the current plan.

19.5.3 Step-Termination Exit

A routine installed at the IEFACTRT exit point receives control from the system when a job or job step terminates, whether normally or abnormally.

Every time an SMF record of certain types is about to be written the exit is called, with a pointer to the record in memory. The types are 4, 5, 30, 32, 34, and 35. Most installations only write Type 30 records. At step- or job-end there might be more than one Type 30 record and the exit is called for each.

The exit has access to the whole SMF record's data, including zIIP and zAAP CPU information. Most installations have an IEFACTRT routine that provides some basic information about the step.

IBM supplies two sample exit routines for this exit point. One of them, IEEACTRT, writes a summary of the step in the job's log. The IEEACTRT sample shipped with z/OS Release 7 includes support for zAAP and zIIP time.

In the absence of SMF analysis, perhaps because it hasn't been processed yet, the output from this exit can be a useful piece of instrumentation.

19.5.4 System Log

The System Log (SYSLOG) is normally viewable using the System Display and Search Facility (SDSF), using the LOG option. It records system messages and a wide variety of other information in a format that might be more accessible than SMF.

This information includes;

- ▶ Tape mount and demount information
- ▶ Operator messages and replies
- ▶ Job run times
- ▶ DFSMSHsm activity
- ▶ SLIP traps firing
- ▶ Security violations
- ▶ WebSphere MQ subsystem status and connection messages
- ▶ DB2 subsystem messages
- ▶ SMF events, such as dump data set switching

SYSLOG is not designed for batch tuning but sometimes the additional information it can provide helps to explain the performance of a batch job or the environment it runs in.

Reduce batch complexity

Batch programs, jobnets, and batch windows can be very complex from an application coding point of view. Applications programs sometimes take on too much of a burden in tasks such as database and file I/O, conversion of data and file transfer. Many times, JCL is used for application logic. All of this can make a batch process difficult to maintain and less agile. There is a lot of middleware on the market that helps abstracting non-business logic out of the application code and at the same time makes the application programs easier to maintain using sophisticated tools.

In this part of the book, we discuss a few areas in which middleware can help to improve the maintainability and agility of the batch process. This part includes the following chapters:

- ▶ In Chapter 20, “Reduce batch complexity using a Business Rules Management System” on page 357, we discuss how IBM JRules can help to maintain business rules in a very agile manner.
- ▶ In Chapter 21, “Reduce batch complexity using middleware for transformation logic” on page 371, we discuss how IBM WebSphere Transformation Extender can help in developing and maintaining transformation logic using a light runtime environment on z/OS and sophisticated development tools.
- ▶ In Chapter 22, “Reduce batch complexity by eliminating custom file transfer logic” on page 397, we discuss IBM WebSphere File Transfer Edition (FTE), which you can use to manage (batch) file transfers.
- ▶ In Chapter 23, “Reduce complexity by exploiting DFSORT / ICETOOL” on page 405, we discuss two z/OS technologies that can help to manage complexity in sort operations within your batch jobs.

Note: Complexity with regards to data access can also be reduced by implementing a smart Information Management architecture. Refer to Chapter 12, “Create agile batch by optimizing the Information Management architecture” on page 189 for an overview of the IBM Information Management solutions available for this purpose.



Reduce batch complexity using a Business Rules Management System

An organization's success depends upon its agility for responding quickly to today's complex, ever-changing markets and regulatory climate. For example, a company's ability to implement new pricing models or change product or service features to meet or beat the competition is a key differentiator in ensuring business success.

Many organizations today have their business policies and business rules automated in applications that were developed over the course of years or even decades, making them difficult for a person to comprehend. As these software assets mature, they tend to become increasingly complex. Unfortunately, this complexity is compounded by a decline in technical and business understanding of how these assets support business goals and priorities. This lack of adaptability leads to a growing misalignment between what an application does, or can do and what the business requires.

The application might become less transparent and the question of how quickly a change can take place becomes the issue. In addition, identifying the exact change that is required for the business without affecting the other processes that are embedded in the large complex applications is even more complicated. For instance, can you be certain that all possible rules to determine and calculate rebates are known and identified? What are the other rules that are dependent upon what seems to be a single or simple change in one program?

The following list some key issues with regards to a corporation's business policies and business rules in System z applications:

- ▶ Business rules are many times implemented as "hard code" and in an inflexible manner, making them hard to maintain, analyze, and manage.
- ▶ As a result of the previous bullet, there is insufficient insight in what business rules are running the business applications today. The business rules were developed and maintained like all the processing code that was developed. Additionally, when business rules are implemented as hard code and developed using regular coding tools (for

COBOL, Java, and so forth), business rules are not recognizable as such anymore in the programs being generated and they just get buried between the rest of the code.

- ▶ In traditional applications reuse of business rules is limited or practically non-existing. Many business rules are imbedded in COBOL and PL/I programs and cannot be accessed individually. When another business unit needs a similar or the same business rule as the one that is already imbedded in an existing program, typically the developer takes a copy of that piece of code, modifies it and re-implements it inside another COBOL or PL/I program. In addition, there are usually two code bases: one for OLTP processing and one for batch processing. Again, here is an example in which sharing the business rules between OLTP and batch processing would be extremely beneficial but unfortunately in the past has not occurred. Here is where a combination of an SOA approach and use of a BRMS (business rules deployed as decision services) can help.
- ▶ The “Waterfall” application development process has its risks, that is the business rule developer (who usually is a programmer with COBOL or Java skills) has to understand and interpret the business rules designed by the business analyst and then include it in hard code. This approach is risky and can lead to an inflexible implementation of the business rules designed by the Business Analyst.
- ▶ Time to market is not what it should be. A simple change in a straightforward business rule could take weeks or even months, depending on the complexity of the existing code. This hurts the company’s performance and frustrates the Line-Of-Business Managers.

Figure 20-1 illustrates the context of both business rule and regular software life cycle management.

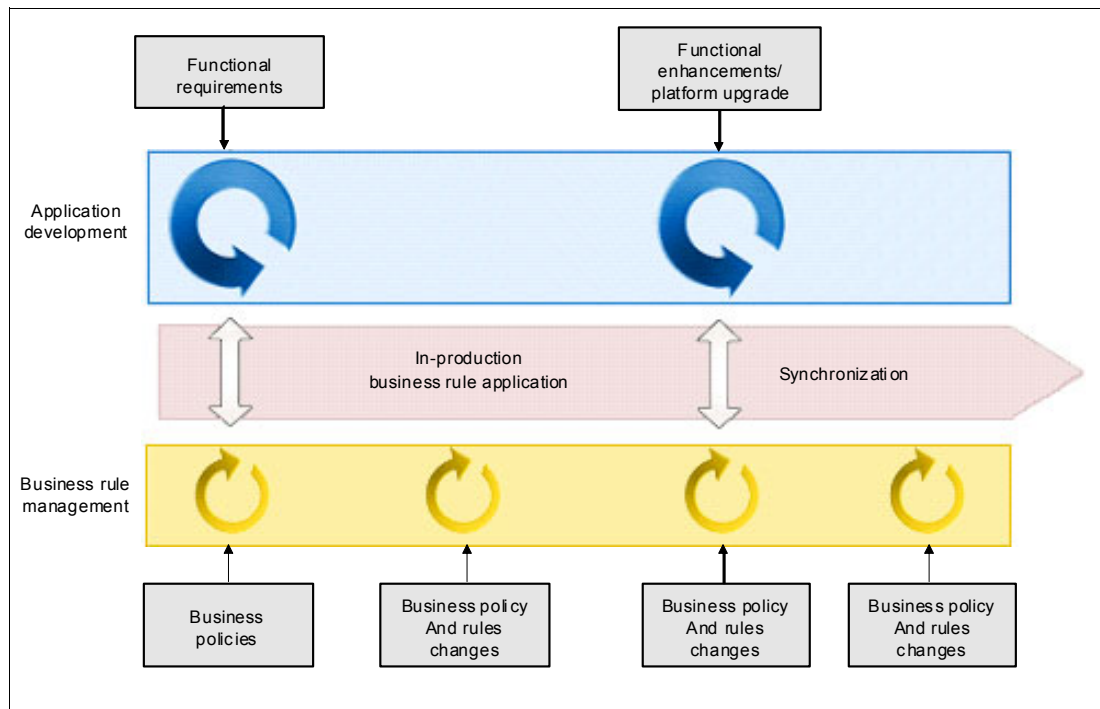


Figure 20-1 Business rule and software life cycle management

To address these issues, corporations have looked to modernize their business applications on System z. These applications modernization projects generally include addressing the following goals:

- ▶ Increase innovation and productivity, by introducing new tools and technologies
- ▶ Increase business agility, by introducing a new application architecture, many times based on SOA concepts, but also better application life cycle management tooling
- ▶ Support growth initiatives
- ▶ Access and reuse the business value that resides in existing systems, primarily by using SOA concepts and rich development tooling
- ▶ Reduce application maintenance costs, by reducing redundancy, improving software quality and more productive application life cycle management tooling

These projects focus on how the core System z applications can address the need to respond rapidly to emerging opportunities. In order to manage agile solution delivery, it is essential to be able to understand line of business applications in terms of the business rules they implement and the effect of rule changes on key business processes.

With the advent of a Business Rule Management System (BRMS) for enterprise application languages such as COBOL on System z, agile responsive solution adaptation has finally become a reality. Through the identification and externalization of application business rules, conditions can be defined that will result in new application behaviors offering a quick and productive route to greatly enhance the business responsiveness of CICS, IMS and batch operations by putting the business owner in control of application behavior change and at the same time retaining good governance and change management.

This chapter provides insight into how to reduce batch complexity using a BRMS system. It includes the following topics:

- ▶ Introduction to Business Rule Management
- ▶ Overview of IBM WebSphere ILOG WebSphere BRMS
- ▶ Using ILOG BRMS on System z
- ▶ Using ILOG BRMS in batch

20.1 Introduction to Business Rule Management

Business rules are widely recognized as representing valuable “organizational DNA,” and the advantages of using a BRMS as an alternative to conventional coding. Business Rule Management addresses the issues of agility and true value for the business. Applications constructed with Business Rule Management Systems deliver flexibility and agility, and enhance the entire policy management infrastructure of the enterprise.

To help introduce the concept of a BRMS, here are some key terms and definitions:

- ▶ Business policy

Every organization has people responsible for setting the policies by which the organization does business. A *business policy* is a statement of guidelines governing business decisions. An insurer can have an underwriting policy, for example, that says, “An existing customer can be eligible for discounts for additional policies if that customer is the main insurance holder and has no disputed claims.”

- ▶ Business rules

The specific statements that enforce a policy are *business rules*. Policies are translated into business rules, the detailed conditions and actions that unambiguously enforce the policy. The business rules expand upon the policy by stating in detail the circumstances under which the policy is applicable and the actions that enforce it. A business policy can be translated into many business rules. In the insurance underwriting policy described above, for example, the rules need to define the terms of the policy (for example, existing customer and disputed claim). Regional regulations might require the rules to vary from region to region, and the amount of the discount can change over time as underwriting experiences more or less risk.

- ▶ Business Rule Management System (BRMS)

A Business Rule Management System (BRMS) provides the software to define and manage business rules through the business rule life cycle. The IBM WebSphere ILOG BRMS solution provides functionality for authoring, testing, analyzing and deploying business rules. Its repository provides full management functionality for business rules, including versioning, baseline management, security and metadata.

20.2 Overview of IBM WebSphere ILOG WebSphere BRMS

IBM WebSphere ILOG BRMS is a suite of tools that business and IT use to build systems in which business rules are authored, modified and managed independently of the underlying software system. *ILOG BRMS* is the IBM technology for creating, maintaining, and implementing decision services that provide the following functions:

- ▶ A convenient communication channel between IT and business teams
- ▶ Easy implementation and reuse of business rules across the enterprise
- ▶ Flexible options for progressive IT modernization

When business rules are managed outside the application (meaning not embedded inside application programs), a required rule change can be assessed, implemented and tested in a much shorter time frame. ILOG BRMS effectively and efficiently delivers full life cycle management for business rules. Managing business rules throughout the rule life cycle requires more than just authoring, testing and deploying rules for a specific business area. Equally important is effective management of changes to rules after they have been deployed to achieve time and cost savings.

Figure 20-2 shows the business rule life cycle.

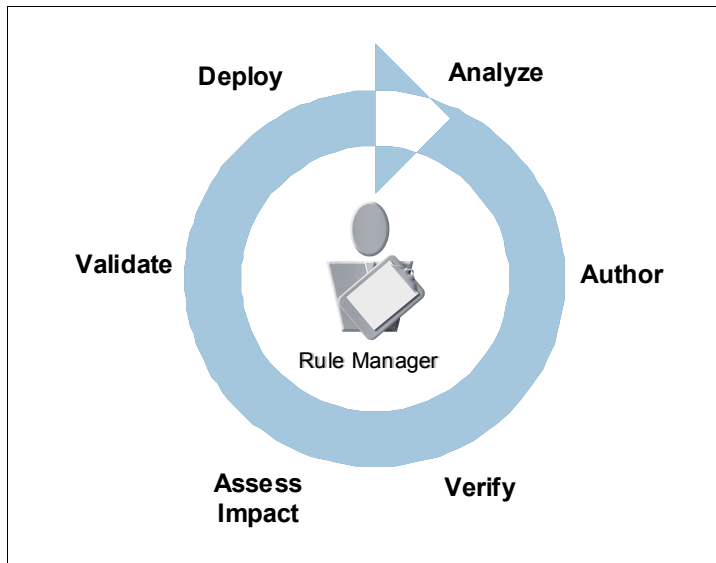


Figure 20-2 Business rule life cycle

The advantages gained to quickly identify which rules need to be changed due to a change in business policy, and then test the proposed changes prior to going into production, provides the agility and competitive advantage to organizations.

The ILOG BRMS provides ways to:

- ▶ Organize business rules
- ▶ Make business rules accessible and searchable by relevant criteria
- ▶ Version business rules and maintain an audit trail
- ▶ Analyze business rules for consistency, completeness and business efficiency
- ▶ Test the implementation of business rules to ensure it is faithful to the business intent

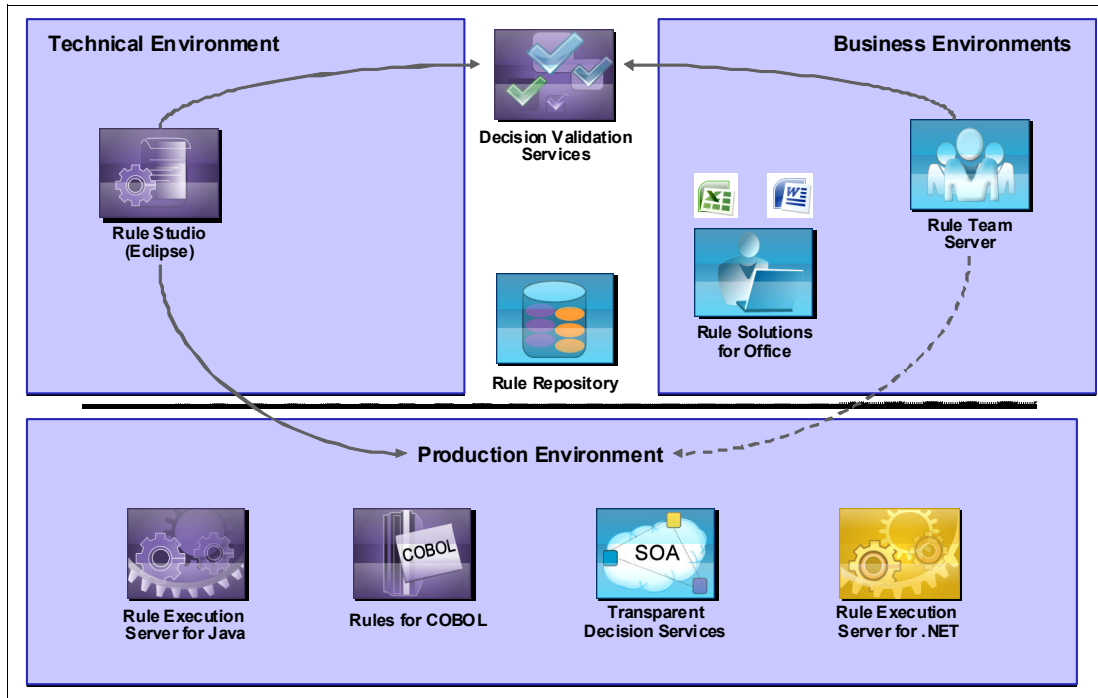


Figure 20-3 IBM WebSphere ILOG BRMS solution components

This functionality is included in a suite of tools within the ILOG BRMS, as shown in Figure 20-3:

- ▶ *Rule Studio* is the business rule application development tool for developers, modelers and architects. Rule Studio is fully integrated with Eclipse, the leading framework for application development tools. It also works with IBM Rational Application Developer, Rational System Architect, and WebSphere Integration Developer. All rule artifacts are managed by Rule Studio as individual plain-text or XML files compatible with Eclipse's standard file management, and can be stored and versioned in any SCM system. And, because Eclipse plug-ins are available for the most widely used SCM systems, developers can do integrated source management, rule project development and Java development for most projects using the same IDE.
- ▶ *Rule Team Server (RTS)* is the business rule management tool for policy managers. It combines a deep, scalable, high-performance rule repository with a thin-client rule management application designed specifically for the needs of policy managers engaged in rule authoring, management and maintenance. See Figure 20-4 on page 363 for an impression of the tool.

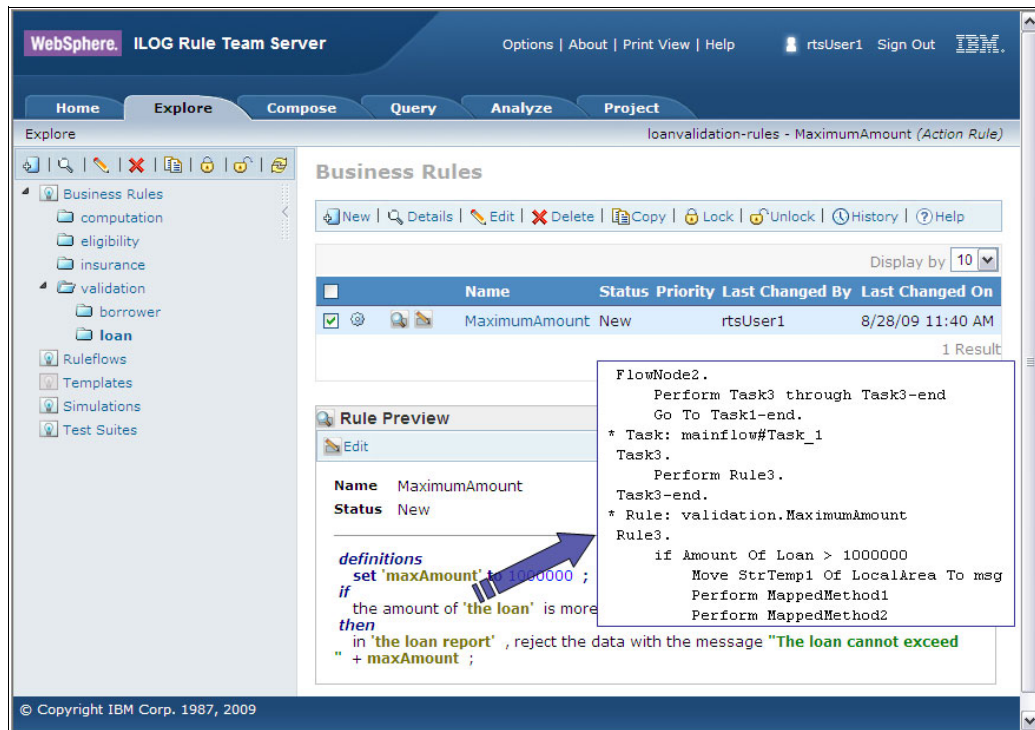


Figure 20-4 IBM WebSphere ILOG Rule Team Server (RTS)

The RTS repository addresses the specific needs of rule-based policy management with:

- Complete multi-project rule management

Projects in RTS contain the project business object model, vocabulary and all the rule artifacts, and are stored without reference to a specific execution model. The RTS repository is designed to be an enterprise repository, storing in a single place multiple independent or dependent rule projects and their histories.

- Multi user access

The repository supports automatic rule-level locking, as well as user-managed persistent locks.

- Scalability

The RTS repository scales to dozens of users working on the same or different projects, and hundreds of thousands of individual rule artifacts.

- Full version and history of rule artifacts

The RTS repository serves as a fully versioned content management system for the Business Object Model (BOM), vocabulary, and rule artifacts. As artifacts evolve with an application, the repository maintains all prior versions of each artifact in an accessible and browsable format, providing a complete audit trail of policy implementations.

- Baseline management

The repository maintains “baselines” of rule project states, allowing any previous project state for which a baseline has been created to be recalled for examination. Current project state can be “rolled back” to any previous baseline.

- RDBMS and transactional safety

The RTS repository is fully relational, and is accessed only under transactional protection, giving the repository the full benefit of content protection offered by modern RDBMSs.
- Open schema

The repository schema is documented for read-only access by the customer's own reporting and querying applications. Business rules and metadata are accessible in readable formats.
- ▶ The *Rule Execution Server* is a managed business rule execution platform that embeds the *JRules* engine. The Rule Execution Server wraps the high-performance ILOG JRules rule engine into a scalable, manageable and monitorable service that provides business rule execution for all server-based applications, service-oriented architectures and embedded rule applications. The Rule Execution Server uses the Java Connector Architecture (JCA) to provide pooled, managed access to rule-based Decision Services. Applications invoke the services through a wide choice of invocation technologies, including stateless or stateful synchronous invocation using simple Java objects (POJOs) or EJBs, or asynchronous invocation using Message-Driven Beans (MDBs). The execution unit can be deployed in leading Java EE application servers using pre-packaged resource archives or as an embedded J2SE component in non-Java EE architectures. Java EE deployments can take advantage of clustered deployments for scalability and robustness, especially when deployed to System z.
- ▶ The *Decision Validation Services (DVS)* module is for testing and simulation of business rules. It provides rule testing functionality for both developers working in the application development context, and policy managers who author and validate rules as part of the business rule life cycle.
- ▶ The *Rules for COBOL* module simplifies the inclusion of BRMS to extend existing IT investments to cost-effectively implement policy change management. The module gives access to most of the features available with ILOG JRules:
 - Define the input and output for generated code by importing COBOL data structures into Rule Studio to create a Business Object Model (BOM).
 - Create a business vocabulary for the BOM and extend the imported data-centric BOM with action phrases.
 - Create rule flows, Business Action Language (BAL) rules, decision tables and decision trees using standard ILOG JRules features.
 - Define and publish rule sets, and then generate COBOL source code from them for compilation into an executable format.

20.3 Using ILOG BRMS on System z

As mentioned previously, many large customers are managing their business critical data on the mainframe being accessed by mainframe or also distributed applications. In this context, data proximity is an important performance factor in general but also when using a BRMS. As shown in Figure 20-5 there are two options for using BRMS on System:

- ▶ Option 1: IBM WebSphere ILOG JRules on System z
- ▶ Option 2: IBM WebSphere ILOG Rules for COBOL

We describe the two options in more detail in the following sections.

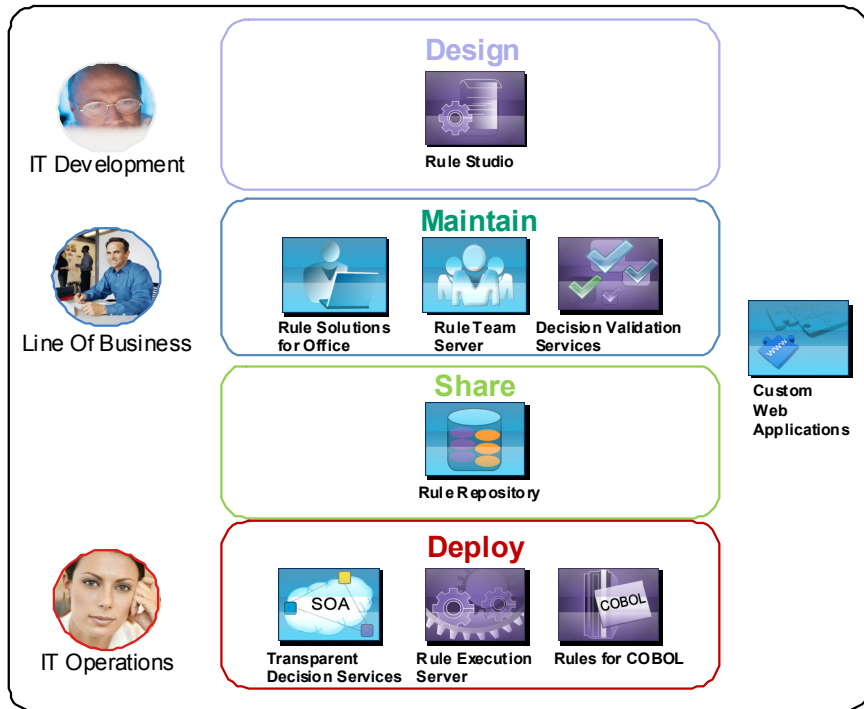


Figure 20-5 ILOG BRMS options for System z

20.3.1 Option 1: ILOG JRules on System z using Rule Execution Server

IBM WebSphere ILOG JRules on System z is the preferred option for companies who want to author and execute decision services as part of their SOA strategy on System z or are developing Java rule-based applications on System z and require a Business Rule Management System with the Java rule execution as part of the application architecture.

The automated decision will then be managed in the ILOG BRMS with the rule expressed and documented in business terms, versioned, with the ability to change it when the business needs it, and easy reuse throughout the enterprise. ILOG BRMS generates the decision services for SOA deployment.

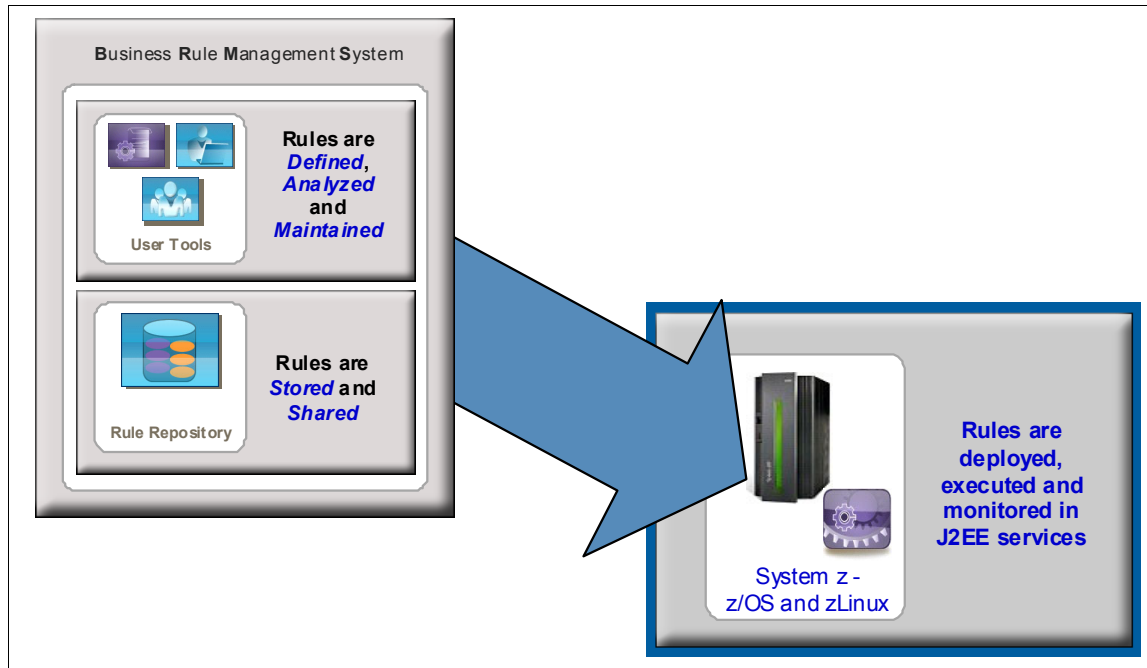


Figure 20-6 ILOG JRules on System z using the Rules Execution Server

The following components are used for development in this option:

- ▶ *Rule Studio*, the Eclipse based tool, is used by developers and technical business analysts.
- ▶ *Rule Team Server* is used by Line-Of-Business Managers and business analysts for rule authoring and management. It is also used as a repository.
- ▶ *Decision Validation Services* is used for rule testing tool by both developers and Line-Of-Business Managers

Execution and management might require RDBMS access, which can be either on or off the System z. The following components are used for execution:

- ▶ *Rule Execution Server*, which is the Java rule engine and administration and monitoring tool. It runs on WebSphere Application Server for z/OS or WebSphere Application Server for Linux for System z.
- ▶ *Rule Team Server Repository*, which is a full featured Web-based rule management tool and requires a RDBMS for the repository. In the case the repository is stored in DB2 on z/OS, ILOG JRules can use a JDBC Type 2 driver to access it.

Example

A large bank has embarked into a SOA strategy on System z. The key objective of their SOA strategy is agility and reuse. They are looking to define different types of services including decision services. Decision services perform decisioning that is unrelated to data transformation. Typically, they are passed data and return a data item that reflects a decision or an action based on the execution of business rules. Examples include services that accept credit reports and return creditworthiness scores. These services are central to the workings of the enterprise and they often represent the core functions of a business process.

The bank has identified and prioritized the business decisions that they want to begin to manage centrally. ILOG's JRules Business Rule Management System is uniquely suited for the formulation and implementation of transparent decision services:

- ▶ ILOG JRules empowers business teams to manage the business rules that automate their policies.
- ▶ ILOG JRules helps development teams to deploy these business rules as fully formed decision services and weave them into SOA platforms.
- ▶ ILOG JRules brings several business-side roles together to collaborate on transparent decision services.

Their business analysts and business policy managers can validate and simulate rules prior to deployment. And once the rules are fully tested, the administrator can hot-deploy them in minutes.

The bank begins to deploy the higher priority decision services for execution. For example, the decision service "Determine Credit Worthiness" was deployed and this service can now be used by a CICS application and batch applications. By authoring the rules once and storing them centrally in the ILOG rule repository, it eliminates the need for developers to maintain two different code sets for online and batch processing. The business can view the rules within the decision service and the IT department can monitor the decision service usage.

20.3.2 Option 2: IBM WebSphere ILOG Rules for COBOL

IBM WebSphere ILOG Rules for COBOL is an option for customers that want to realize the full benefits while retaining the existing COBOL architecture. ILOG Rules for COBOL provides the benefit of incremental application modernization by managing business logic independently of technical architecture. ILOG Rules for COBOL serves as a bridge between ILOG JRules and COBOL applications.

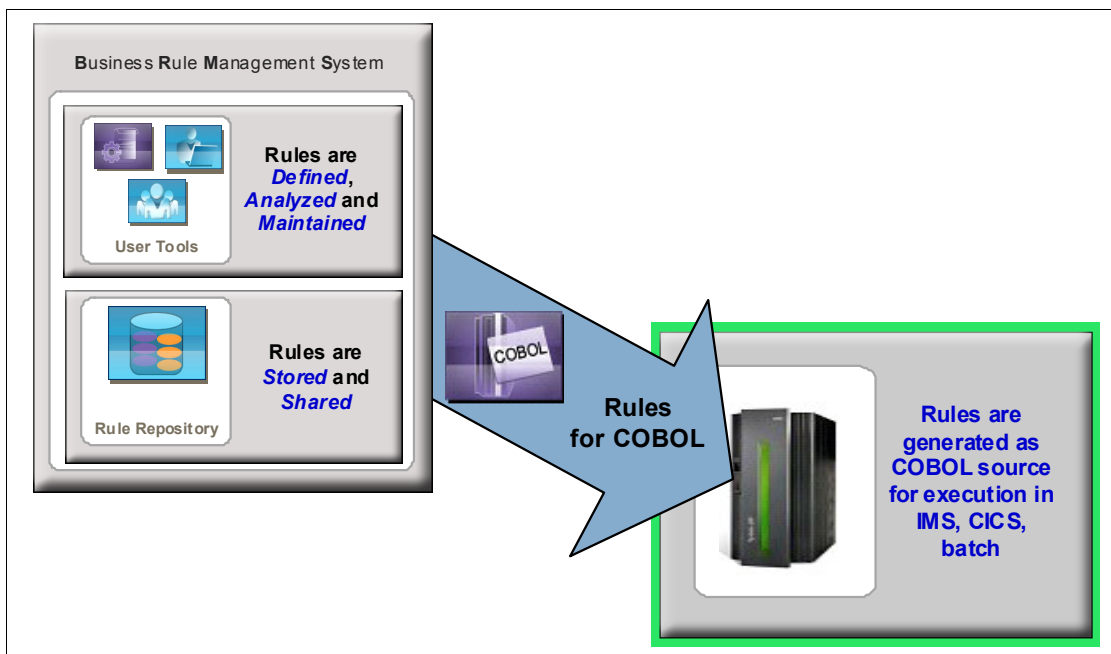


Figure 20-7 ILOG Rules for COBOL

As shown in Figure 20-7 the steps are as follows:

1. Define the input and output for generated code by importing COBOL data structures into IBM WebSphere ILOG Rule Studio to create a Business Object Model (BOM).
2. Create a business vocabulary for the BOM and extend the imported data-centric BOM with action phrases.
3. Create rule flows, Business Action Language (BAL) rules, decision tables and decision trees using standard JRules features.
4. Generate COBOL source code from defined and published rule sets for compilation into an executable format.

Using ILOG Rules for COBOL is a combination of the Business Rule Management functionality of the core ILOG and targeted for native COBOL execution:

► **Development and management:**

ILOG Rules for COBOL is installed with Rule Studio which is the rule development tool based on Eclipse. This functionality includes the capability of the import of the COBOL copybook for the creation of the Business Object Model (BOM) and the generation of the COBOL sub-program.

In addition, ILOG Rules for COBOL can also be installed in Rule Team Server (RTS) for enhanced rule authoring and management of business users. This install complements the rule authoring and management of RTS with the functionality of the generation of the COBOL sub-program.

- Execution of the rules are through a generated sub-program in Enterprise COBOL that is compiled and linked on z/OS for native COBOL execution.

Example

A large manufacturing company runs its mission critical COBOL billing system (a rule intensive application) on z/OS. This billing system is the center of a lot of other business applications and performance is critical. The Finance Department has expressed a desire to accelerate their release schedule substantially, in order to accommodate new products and services that the business wants to offer. The IT department would like to streamline their development cycle and wants to retain the COBOL execution on z/OS. In addition, the IT department knows that the new product and service rules will need to be shared with applications for order fulfillment. The manufacturing company will begin using ILOG JRules with ILOG Rules for COBOL for management of their rules used in their COBOL applications.

The Business Analysts in the Finance Department work with the IT Developers to identify the scope of data fields that will be impacted by these new rules. The IT department then identifies the COBOL copybooks using these fields. These COBOL copybooks are then used to create the Business Object Model (BOM) within ILOG JRules. Each COBOL field is given a business name to ease the authoring of the rules as well as the review and reporting of the rules.

The Business Analysts author the new product rules within Rule Team Server (the Web management tool). Each set of business rules is based on a business decision required for a business application. You can set up a rule project for each business decision that will in turn equate to one subprogram generated by ILOG Rules for COBOL.

A rule flow is also designed for the ILOG Rules for COBOL project. This rule flow will be to identify the order and dependencies in which the rule sets are needed for generation and execution in the COBOL subprogram.

Because these are new rules for a new product, the COBOL developer has identified where in the application the ILOG Rules for COBOL generated program should be called and updates the program accordingly. Then, when notified by the business analyst that the rules are ready for testing, the COBOL developer uses Rule Studio to generate the COBOL program from the rules within the ILOG JRules repository. This program is compiled and linked and ready for testing.

If these rules need to be changed, the Business Analyst would go back into RTS to make the necessary changes. RTS handles all the versioning and traceability of the rules.

Business rule mining

To begin modernizing an existing COBOL application, the current coded rules within the application must be identified. This is where a business rule mining effort is helpful.

Business rule mining is the process of extracting essential intellectual business content (business rules) from packaged or existing software, recasting them in natural language, and storing them in a source rule repository for further analysis or forward engineering. The goal is to capture these existing business rules in a way that the business can validate, control and change them over time.

The target of your business rule mining effort will be directed by the objectives of the project as well as the scope of the project phases. For example, if the first phase of a project is to implement a system based on jurisdiction (that is, state or region), then the rule mining effort would focus on identifying those rules that pertain to that jurisdiction. Other examples for rule mining efforts are by sub-process, output report/screen or fields, specific business decision. The output of the business rule mining project is the candidate business rules hopefully linked or connected together to better understand the flow as these candidate rules were found in the target program or programs as well as the related COBOL copybooks or file layouts.

The candidate rules and COBOL copybooks are used as sources for the modernization project. With WebSphere ILOG Rules for COBOL, the modernization effort is done in three steps to improve COBOL application flexibility and to rapidly create new applications with WebSphere ILOG JRules and WebSphere ILOG Rules for COBOL:

1. Import the COBOL copybook
2. Author the business rules by using the candidate rule report and redesign the coded rules within one of the rule artifacts in WebSphere ILOG JRules
3. Generate the COBOL program for native COBOL execution.

Summary


By using the BRMS with your existing COBOL applications, your organization can realize the following bottom-line benefits:

- ▶ Preserved business value of existing assets and rules managed as intellectual assets in the BRMS.
- ▶ Minimized business risk by ensuring consistency of the business rules throughout business applications.
- ▶ Improved competitive position through enhanced business service and reuse with the BRMS technology.
- ▶ Enhanced cost reduction and ROI with reduced total cost of ownership.

20.4 Using ILOG BRMS in batch

Business rules developed and deployed with ILOG BRMS can be used in different ways in a batch environment on z/OS:

- ▶ When running in the Rules Execution Server (Java EE or J2SE) on z/OS, rules can be invoked as a decision service from an outside batch program. Depending upon the batch program, the application architect needs to design how ILOG JRules will be executed.
 - Batch programs running in WebSphere XD Compute Grid will have easy access to ILOG business rules, as both environments are natural Java EE environments.
 - Batch programs running in stand-alone Java (eventually using JZOS) have easy access to the business rules too. When ILOG business rules are deployed to a J2SE environment, these can easily be reused from a stand-alone Java batch program; when ILOG business rules are deployed to a Java EE environment, a Java stand-alone batch program can perform a remote call to the business rule service in the Java EE server.
 - Traditional batch programs written in COBOL or PL/I and running with JCL, however, do not have easy access to a Java EE or J2SE environment. Options to access a service in a Java EE environment from a traditional batch program are discussed in *Enabling z/OS Applications for SOA*, SG24-7669.
- ▶ When using ILOG Rules for COBOL, business rules can be easily integrated with existing COBOL batch programs, as the business rules are generated as executable COBOL modules.



Reduce batch complexity using middleware for transformation logic

Batch processes in large companies can be quite complex. It is not unusual that those companies run thousands of different batch jobs in their environment. In most cases, each of these jobs is built of several steps. Within these steps, the data is normally processed either with standard utilities or with self-written programs.

Thus, input data often must be converted before continuing to the real business logic. In many cases this conversion is done with custom self-written programs. Depending on the transformation requirements, conversion with self-written programs can be a very expensive process. If there are changes in the transformation logic or a new transformation rule is added, a lengthy application development process must be performed.

Today, there are tools available on the market that support data transformations. By using such a tool, it is possible to react faster and in a more flexible manner regarding changes and new requirements in data transformation logic. Alternatively, it is possible to reduce the complexity of batch, because there is only one single tool in use for all types of different transformations.

The WebSphere Transformation Extender is available on the System z platform for embedding conversion logic in new or existing batch applications. You can also integrate WebSphere Transformation Extender with DB2 and participate in CICS or IMS transactions. In addition to z/OS, it also supports Linux for System z.

WebSphere Transformation Extender on System z provides the ability to quickly connect and interface existing batch applications among themselves and with industry standards. The transformation maps are developed in WebSphere Transformation Extender Design Studio on Windows, where users can build processing and integration rules, integrate data of disparate types from disparate sources, and process data objects natively without the need to program in COBOL, PL/I, or Java.

WebSphere Transformation Extender on System z provides the following advantages, among others:

- ▶ Eliminate custom integration programming for transforming data
- ▶ Avoid costly changes to applications written to older industry standards
- ▶ Improve quality assurance by validating information exchanges outside of the applications
- ▶ Reduce time and cost, maintaining interfaces to evolving standards
- ▶ Allow for fewer IT backlogs and code free maintenance

In this chapter, we create an example to show how to use WebSphere Transformation Extender to transform data. This chapter includes the following topics:

- ▶ WebSphere Transformation Extender: Enabling universal transformation
- ▶ Business value of WebSphere Transformation Extender
- ▶ Sample using WebSphere Transformation Extender in z/OS batch

21.1 WebSphere Transformation Extender: Enabling universal transformation

WebSphere Transformation Extender is a powerful, transaction-oriented universal data transformation and validation solution that delivers flexibility for IT systems, resulting in business agility. It automates the transformation of high-volume, complex transactions without the need for hand-coding, which provides enterprises with a quick return on investment.

WebSphere Transformation Extender performs transformation and routing of data in any format, including XML, non-XML, and in mixed formats, from source systems to target systems in batch and real-time environments. The sources can include files, relational databases, message-oriented middleware (MOM), packaged applications, or other external sources. After retrieving the data from its sources, WebSphere Transformation Extender transforms the data and routes it to any number of targets where it is needed, providing the appropriate content and format for each target system.

WebSphere Transformation Extender is integrated with other components of the IBM WebSphere integration software suite for complete operational and transactional data integration across the enterprise. It provides the following benefits:

- ▶ Highly automated transformation and routing of complex data across many points of integration in real time to support high message volumes
- ▶ Enterprise-wide interoperability supported by a services-oriented architecture (SOA) for seamless connectivity and interoperability across back-office systems
- ▶ Connectivity to a wide range of mainframe, existing, and enterprise applications, databases, messaging systems, and external information sources
- ▶ Support for high-performance, event-driven, transactional environments to ensure completion and validation of transactions in real time
- ▶ A comprehensive library of more than 120 pre-built functions to reduce development time and simplify specification of rules for validation, transformation, and routing
- ▶ Seamless integration across the development, data, and production layers of the enterprise, using existing IT infrastructures
- ▶ Ready-to-use solutions for industry standards and regulatory compliance for operational and transactional data integration
- ▶ Multiple execution options to support right-time, right-style transformation, whether it is batch, real-time, or embedded

WebSphere Transformation Extender exposes transformations as services inside SOAs. The WebSphere Transformation Extender family delivers editions for both batch- and event-driven systems, extending an enterprise service bus (ESB) or business process management (BPM) solution. The editions can be embedded with customer applications written in a variety of languages including C, COBOL, Java, and .Net. WebSphere Transformation Extender supports many platforms, including the System z platform, where customers can deploy WebSphere Transformation Extender in z/OS batch, CICS, IMS, UNIX System Services, and Linux on System z environments. In the remainder of this chapter we focus on using WebSphere Transformation Extender in a batch environment.

21.2 Business value of WebSphere Transformation Extender

Without a dedicated solution for transformation, you need significant programming skills to transform complex data formats, such as a Health Insurance Portability and Accountability Act (HIPAA), Electronic Data Interchange (EDI), or an SAP IDOC document to other formats. In this scenario, you need to write specialized, format-specific programs to perform the preprocessing and postprocessing tasks of checking content quality. The tasks that are associated with manually transforming formats must all be administered and maintained over time, incurring development and maintenance costs.

WebSphere Transformation Extender minimizes the impact that integration imposes on an application. WebSphere Transformation Extender maintains each transformation mapping, including content validation and format variations, in a portable form, regardless of sources, target applications, platforms, or adapter requirements, and without the user having to write code. Its powerful write-once, deploy-anywhere portable technology enables you to grow your infrastructure with your business.

New business services can be delivered quicker, with business-driven enhancements turned around in shorter cycles. Business is also better served when electronic documents that are exchanged between organizations can be validated before they are sent or received, eliminating the costly errors and corrective actions that are required to re-send the documents. Validation checking is also important for businesses that must comply with regulatory and industry requirements for business-to-business transactions. In-process validation of data before it is mapped eliminates wasted processing time, which can be expensive.

What makes WebSphere Transformation Extender so different? WebSphere Transformation Extender has unique capabilities that make it extremely powerful:

- ▶ WebSphere Transformation Extender has a unique many-to-many model of transforming and processing data. With this model, WebSphere Transformation Extender can run all transforms, lookups, and data enrichments with only one pass at the data, making it one of the best performing transformation engines on the market.
- ▶ Data is validated to content rules and context usages as part of the transformation process. It is not necessary to write separate logic or have separate executions to provide extremely rich data validation.
- ▶ There is no “language” to WebSphere Transformation Extender. The transforms and data process are all maintained within the spreadsheet-type GUI. It is not necessary to write code to handle complex transforms.
- ▶ WebSphere Transformation Extender uses self-describing data models to handle data in its native format and has a unique mechanism for describing data in its native form.

Therefore, with WebSphere Transformation Extender you can solve the following problems regarding:

- ▶ Interoperability
 - Connects applications, databases, processes within the enterprise
 - Partners processes across the enterprise
 - Powerful transformation and routing
- ▶ Speed
 - Easy design and implementation
 - Fast maintenance without coding
 - Rapid adaptation to change

- ▶ Simplified complexity
 - No coding
 - Consistent approach to multiple types of integration
 - Reusable objects
- ▶ Adaptability
 - Uses the existing IT infrastructure
 - Fully supports existing IT investments such as databases and messaging middleware
 - Non-invasive integration
 - Application and industry packs

21.3 Sample using WebSphere Transformation Extender in z/OS batch

In this section, we show how to use WebSphere Transformation Extender in a batch scenario on z/OS. We transform XML-formatted data to a COBOL copybook format without writing any specific code. All necessary activities could be performed with WebSphere Transformation Extender Design Studio using drag-and-drop functionalities.

Important: Although XML structures can be quite complex, the XML data structure in our example is relatively simple. WebSphere Transformation Extender Design Studio is prepared for complex XML structures, but you need to factor in time to obtain sufficient experience to design and build complex mappings.

Example 21-1 shows the format of the XML input data, and Example 21-2 shows the required copybook style output data format.

Example 21-1 XML input data

```
<?xml version="1.0" encoding="UTF-8"?>
<Company>
  <Name>Big data center</Name>
  <Department>
    <DepNo>1</DepNo>
    <Description>Production</Description>
  .....
```

Example 21-2 Output data

Big data center	000000001Production	000000001Paul	Smith
Big data center	000000001Production	000000002Hannah	Smith
Big data center	000000002Development	000000003Mia	Brown

Note: Unfortunately, in this broad-scoped book, it is impossible to describe all functions and possibilities of WebSphere Transformation Extender. For more information, see *IBM WebSphere Transformation Extender 8.2*, SG24-7693.

The transformation rules for this process are saved in a mapping file. To develop this file, we use the Eclipse-based WebSphere Transformation Extender Design Studio V8.2 tool. Figure 21-1 show the following relevant steps for our sample with WebSphere Transformation Extender:

1. Create transformation rules with WebSphere Transformation Extender Design Studio.
2. Save mapping file on local hard disk.
3. Test transformation rules with local files.
4. Transfer mapping file through FTP (binary) to the mainframe.
5. Run the mainframe job to process input data.

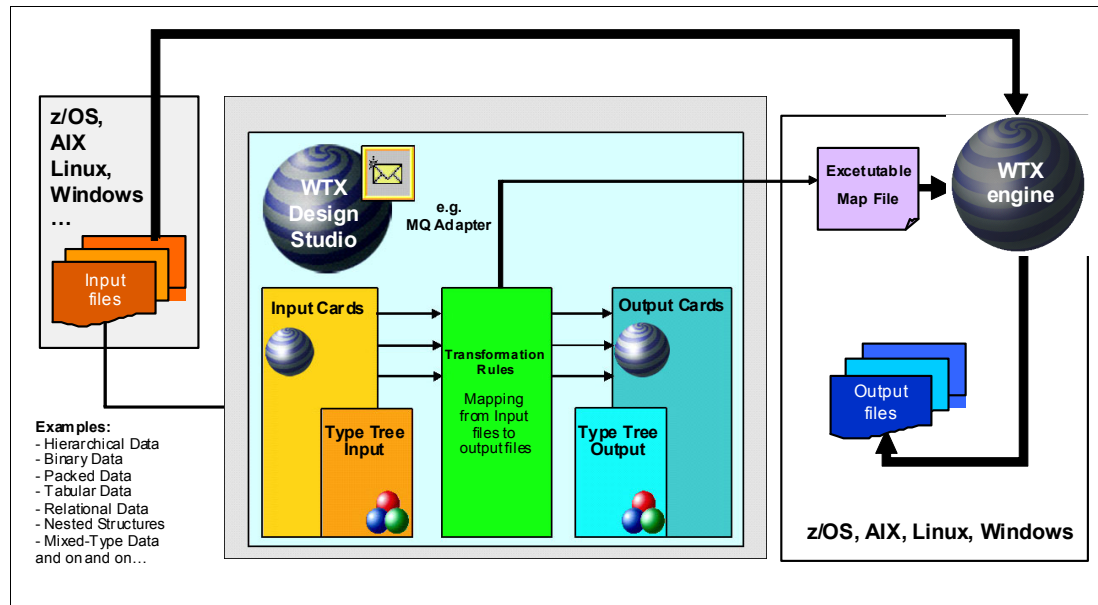


Figure 21-1 Development of WebSphere Transformation Extender maps and deployment

The WebSphere Transformation Extender Design Studio is used at design time to model, develop, and test the data transformation and application integration maps and systems. You can integrate both content transformation and content-based routing through the WebSphere Transformation Extender Design Studio graphical interface. The WebSphere Transformation Extender Design Studio version 8.2 is an Eclipse-based design environment that contains several components. For this example we use two of them:

- Type Tree Editor* Used to define data objects (including source and target data structures).
- Map Editor* Used to develop maps that define your data transformation logic.

To develop a WebSphere Transformation Extender solution, you need to complete the following steps:

1. Describe the data.

Data that is handled by WebSphere Transformation Extender is described in type trees that define the data structure and semantics. The type trees are created in the Type Tree Editor.
2. Transform the data.

Data is mapped between the source or sources and destination or destinations by using a drag-and-drop method and entering map rules. The transformation maps are created in the Map Editor.

3. Deploy the transformation.

Upon completion of the design, the map is compiled and deployed to the appropriate WebSphere Transformation Extender runtime edition.

Note: You can download the necessary material to create this example (such as input data, the WebSphere Transformation Extender Design Studio workspace, and so forth) from the Internet. See Appendix C, “Additional material” on page 453 for more information.

21.3.1 Creating the mapping file

You must create a *mapping file* for data transformation. This file contains information about the input and output format and the necessary rules for transformation. Depending on the complexity of the input and output data, creating a mapping file can be an elaborate process.

Note: On the market today, many predefined mapping files are available to support different standards. For example, to convert data based on the European Single Euro Payments Area (SEPA) standard, it is easier to buy the predefined mapping file instead of creating this file natively with WebSphere Transformation Extender.

In our example, a batch COBOL program expects data in COBOL copybook format, but the input is currently only available in XML format. Instead of changing the COBOL program and using built-in XML functions¹, we use WebSphere Transformation Extender to transform the data first and then pass it on in copybook format to the COBOL program. With this method, there is no need to change the COBOL program.

Input and output data

Our XML input data contains information about a company (see Example 21-3) and is stored in the file `c:\Redbook\WTX\input.xml`.

Example 21-3 XML input data input.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Company>
  <Name>Big data center</Name>
  <Department>
    <DepNo>1</DepNo>
    <Description>Production</Description>
    <Employee>
      <EmpNo>1</EmpNo>
      <FirstName>Paul</FirstName>
      <LastName>Smith</LastName>
    </Employee>
    <Employee>
      <EmpNo>2</EmpNo>
      <FirstName>Hannah</FirstName>
      <LastName>Smith</LastName>
    </Employee>
  </Department>
  <Department>
    <DepNo>2</DepNo>
    <Description>Development</Description>
```

¹ Refer to 4.2, “XML support in COBOL and PL/I” on page 36 for more details regarding XML support in COBOL and PL/I.

```

    <Employee>
      <EmpNo>3</EmpNo>
      <FirstName>Mia</FirstName>
      <LastName>Brown</LastName>
    </Employee>
  </Department>
</Company>

```

A corresponding XML schema definition is stored in file `c:\Redbook\WTX\input.xsd` (see Example 21-4).

Example 21-4 XML Schema Definition input.xsd

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Company" type="companyInfo" />
  <xsd:complexType name="companyInfo">
    <xsd:sequence>
      <xsd:element name="Name" type="xsd:string" minOccurs="1" maxOccurs="1" />
      <xsd:element name="Department" type="departmentInfo"
        minOccurs="1" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="departmentInfo">
    <xsd:sequence>
      <xsd:element name="DepNo" type="xsd:unsignedShort"
        minOccurs="1" maxOccurs="1" />
      <xsd:element name="Description" type="xsd:string"
        minOccurs="1" maxOccurs="1" />
      <xsd:element name="Employee" type="employeeInfo"
        minOccurs="1" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="employeeInfo">
    <xsd:sequence>
      <xsd:element name="EmpNo" type="xsd:unsignedShort"
        minOccurs="1" maxOccurs="1" />
      <xsd:element name="FirstName" type="xsd:string"
        minOccurs="1" maxOccurs="1" />
      <xsd:element name="LastName" type="xsd:string"
        minOccurs="1" maxOccurs="1" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

This data needs to be transformed to COBOL copybook format (see Example 21-5). The copybook format is stored in file `c:\Redbook\WTX\output.cpy`.

Example 21-5 COBOL copybook format output.cpy

```

01 Company.
   05 Name          PIC X(20).
   05 Department.
     10 DepNo       PIC 9(10).
     10 Description PIC X(15).
     10 Employee.

```

15 EmpNo	PIC 9(10).
15 FirstName	PIC X(10).
15 LastName	PIC X(10).

Starting WebSphere Transformation Extender Design Studio and creating a project

First, start WebSphere Transformation Extender Design Studio, and create a new Extender project called *Redbook* by selecting **File** → **New** → **Extender Project**. Figure 21-2 shows the newly created project.

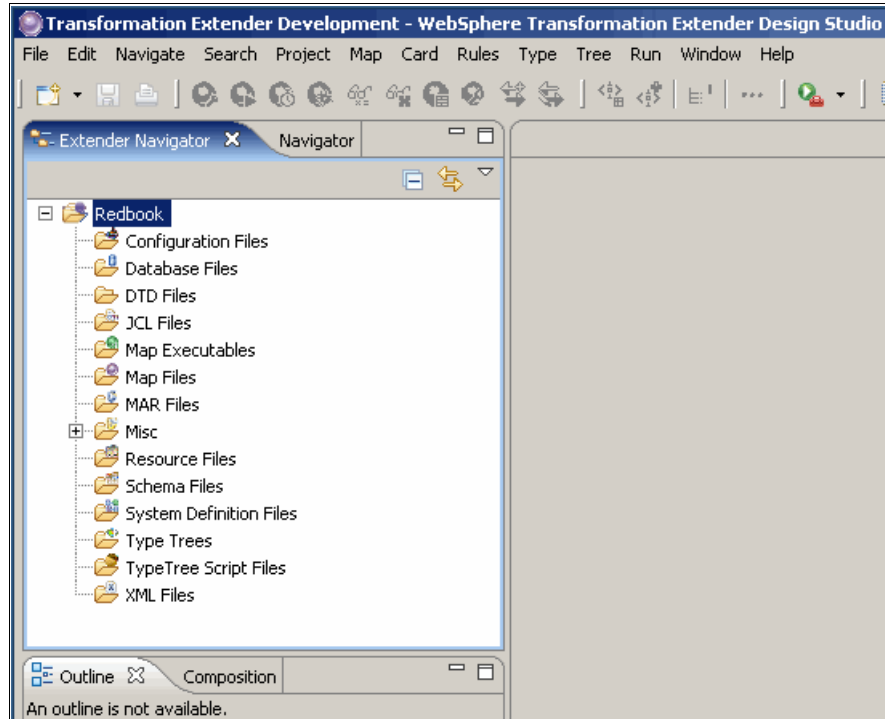


Figure 21-2 WebSphere Transformation Extender Design Studio project

Importing XML files

To import the input XML data:

1. Right-click **XML Files**, and choose **Import** → **File System**.
2. In the Import window, select the `input.xml` file. Then, click **Browse** to select the Redbook project, and click **Finish** (see Figure 21-3).

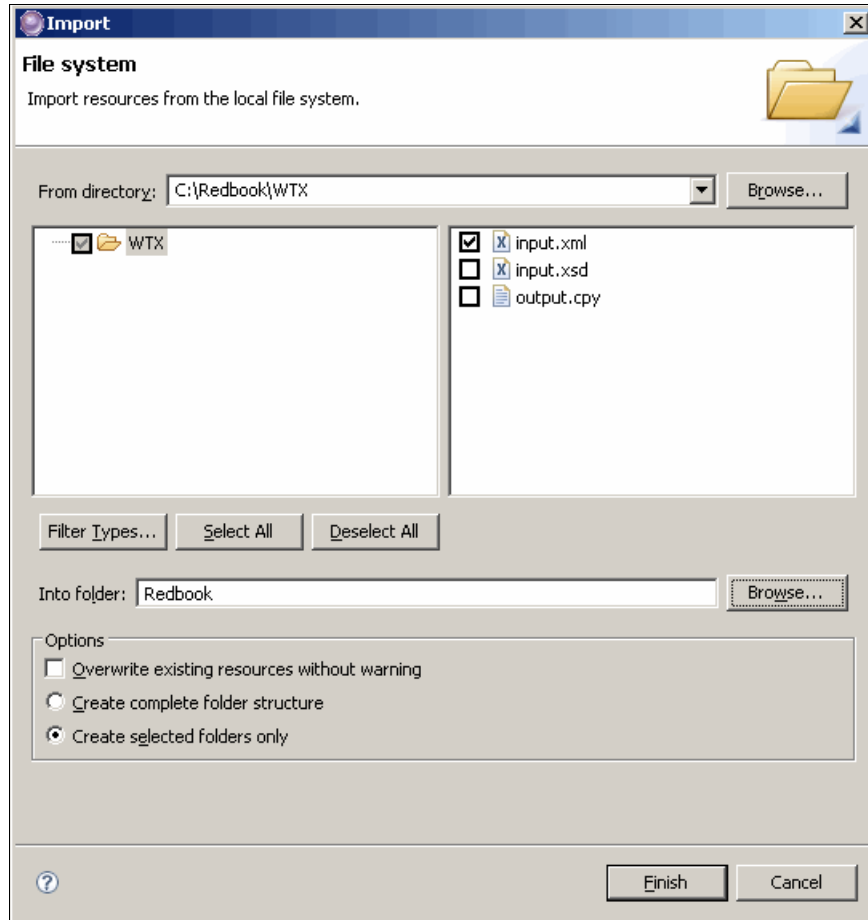


Figure 21-3 Importing XML input data

3. Next, import the XML schema definition file. Right-click **Schema Files**, and choose **Import** → **File System**.
4. In the window that opens, select the `input.xsd` file. Then, click **Browse** to select Redbook project, and click **Finish**. This method is similar to importing XML input data.

Creating the Input Type Tree using XML Schema Definition

For transformation, you need to describe input and output data. In WebSphere Transformation Extender, this description is called a *Type Tree*. You can create a Type Tree manually or use a predefined WebSphere Transformation Extender import assistant. To create the Type Tree for our input data:

1. Right-click **Type Trees**, and choose **Import** → **XML Schema** (see Figure 21-4).

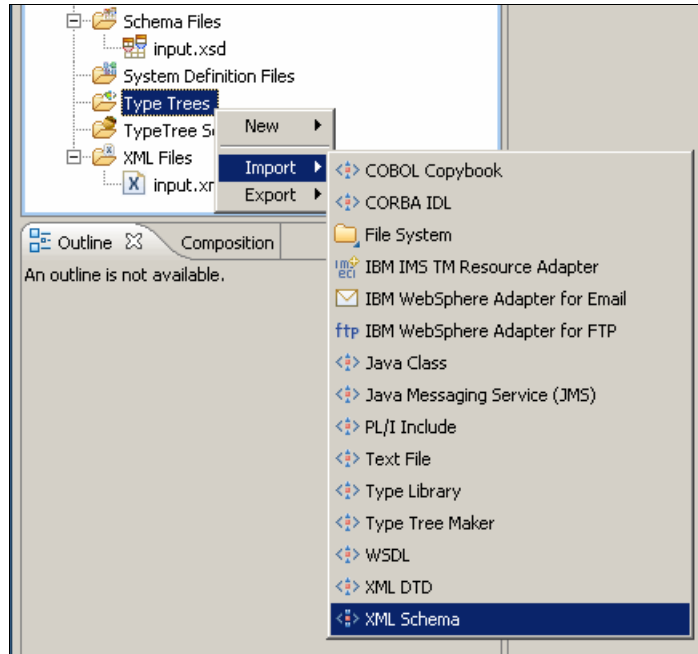


Figure 21-4 Importing XML Schema for Input Type Tree

2. In the window that opens, choose the `input.xsd` file, and **Next** twice. In the window that opens, select the Redbook project (see Figure 21-5). Click **Next**, and then click **Finish**.



Figure 21-5 Choosing Project for Input Type Tree

3. If the input file does not open automatically, open the `input.mtt` file, which is located in the Type Trees folder. Using **Tree** → **Analyze** → **Logic Only**, the input definition type tree is checked by Design Studio. The Analysis Results tab, as shown in Figure 21-6, shows the output of analysis process. On this tab, you can verify whether the Type Tree definition has any warnings or errors.

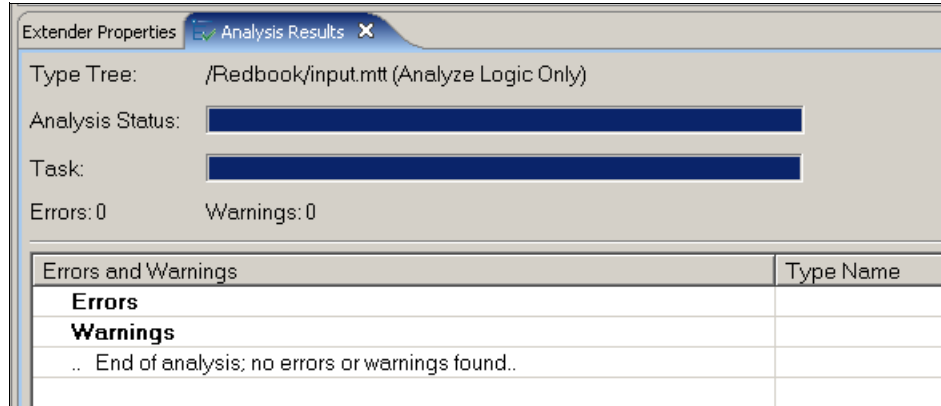


Figure 21-6 Analysis Results

4. Close the `input.mtt` file, and click **Yes** to save the changes.

Note: With the V8.2 release, maps can now use schemas directly as definitions in input or output cards. It means that you are not required to use the XML Schema Importer to create type trees, as in previous versions of WebSphere Transformation Extender. To use the schema directly in a transformation map, select an `.XSD` file instead of an `.MTT` file from within the Map Editor.

Creating the Output Type Tree using COBOL copybook

Next, you need create the Type Tree for the output data:

1. Right-click **Type Trees** in the Extender Navigator window, and choose **Import** → **COBOL Copybook**.
2. In the window that opens, select the `output.cpy` file. Then, click **Next** twice, select the Redbook project, click **Next**, and finally **Finish**.
3. If the file does not open automatically, open the `output.mtt` file, which is located in the Type Trees folder.

In the COBOL copybook, each row of the output data contains data for one employee of the company. In other words, a copybook represents one record. Because the output is a file that contains several records, you need to modify the Type Tree by creating a FILE level group to represent the entire file of records.

4. Right-click **CopyBook**, and choose **Add** to add a new type in the output Type Tree, as shown in Figure 21-7.



Figure 21-7 Add new type to output tree

- After clicking **Yes**, enter `OutputFile` as the name for this new Type Tree. Next, change the type of `OutputFile` by right-clicking **OutputFile** and selecting **Properties**. In the Extender Properties tab, change the Property Class from *Category* to *Group* (see Figure 21-8).

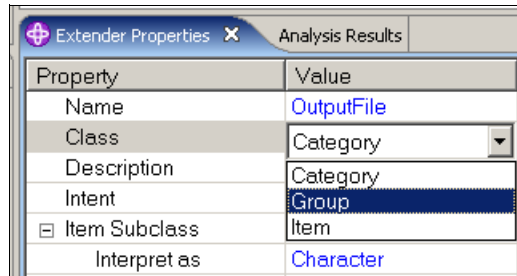


Figure 21-8 Change Property of new Type Tree

- Click **Yes** to confirm that change. Then, right-click **OutputFile**, and choose **Open**. Drag-and-drop **Company** in the first column of the Component tab, as shown in Figure 21-9.

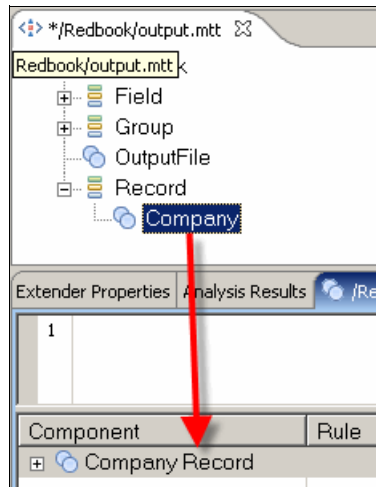


Figure 21-9 Edit "OutputFile"

- You now need to set the range of the component to specify how many occurrences of the element are expected. To set the range of the `Company Record` component, right-click the component, and select **Set Range**.
- In this example, specify an indefinite number of occurrences. So, in the Max field, enter `s`, which stands for Some. Then, click **OK**. The name changes to *Company Record(s)*, which tells WebSphere Transformation Extender to produce multiple different output rows (not only one single row).



Figure 21-10 Edit Component "Company Record"

- Next, you need to tell WebSphere Transformation Extender to write every row in a separate line. Right-click **Record - Company**, select **Properties**, and change the Terminator field from NONE to Literal. You also need to change the Value field to <LF> (for LineFeed), as shown in Figure 21-11. Click the ... button, select **LF**, click **Insert**, and then **OK**.

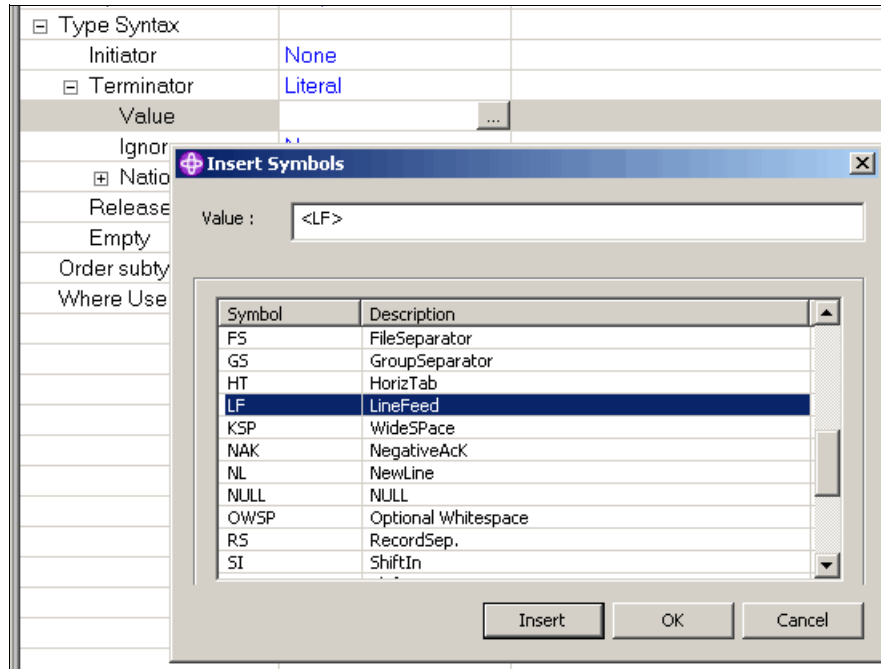


Figure 21-11 Changing value for Terminator

- Using **Tree** → **Analyze** → **Logic Only**, the output definition Type Tree is checked by WebSphere Transformation Extender Design Studio. The Analysis Results tab, shown in Figure 21-6 on page 382, again shows output of the analysis process. On this tab, you can verify whether the Type Tree definition has any warnings or errors.
- Close the output .mtt file, and save the changes.

Creating the transformation map

A transformation map contains information such as input validation, transformation, output format, and much more to produce output data. For detailed information, see *IBM WebSphere Transformation Extender 8.2*, SG24-7693.

To create a transformation map:

- In the Extender Navigator window, right-click **Map Files**, and select **New** → **Map Source** (see Figure 21-12). A map source file can contain several transformation maps.

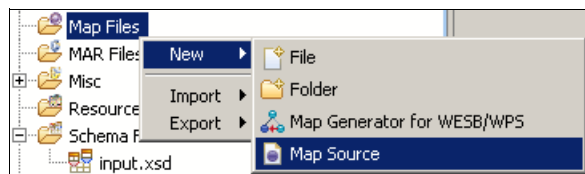


Figure 21-12 Creating the mapping file

- In the window that opens, select the Redbook project, set File Name to XML2Cobo1, and click **Finish**.
- Then, in the Outline tab, right-click **XML2Cobo1**, and select **New** (see Figure 21-13). Enter XML2Cobo1Map as the name for this map.

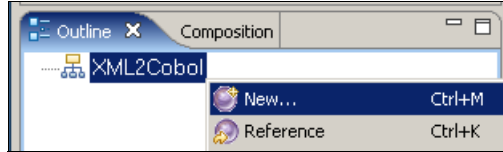


Figure 21-13 Creating a new map

Creating the Input Card

To create the Input Card:

- Right-click **Input Cards** in XML2Cobo1Map, and select **New** (see Figure 21-14). Input and Output Cards are based on Type Trees, as defined earlier.

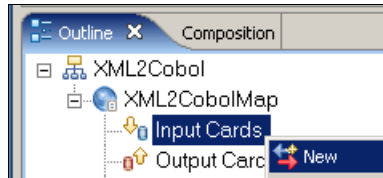


Figure 21-14 Create Input Card

- In the window that opens, set the **CardName** Property to InputXML and choose **input.mtt** from the Redbook project as the Type Tree property (see Figure 21-15).

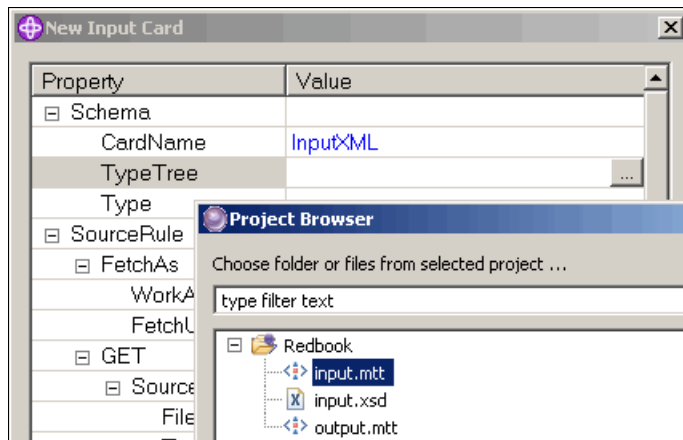


Figure 21-15 Set property Type Tree for Input Card

- For property Type, select **DOC** from the XML schema definition (see Figure 21-16).

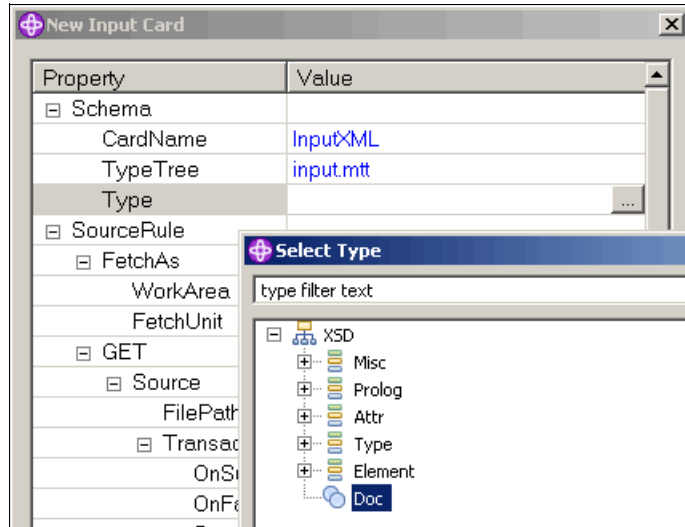


Figure 21-16 Set property Type for the Input Card

- Choose the input.xml file for the FilePath Property (see Figure 21-17), and click **OK**.

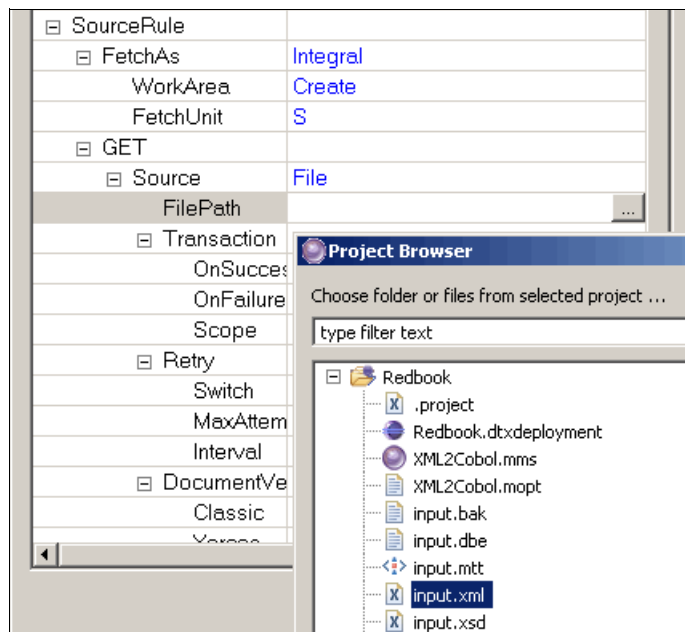


Figure 21-17 Set property FilePath for the Input Card

Creating the Output Card

Next, you need to create an Output Card definition:

1. In the Outline window under XML2CobolMap, right-click **Output Cards**, and select **New**.
2. In the window that opens, set the CardName Property to OutputCOBOL, and choose **output.mtt** from the Redbook project as the Type Tree property (see Figure 21-18).

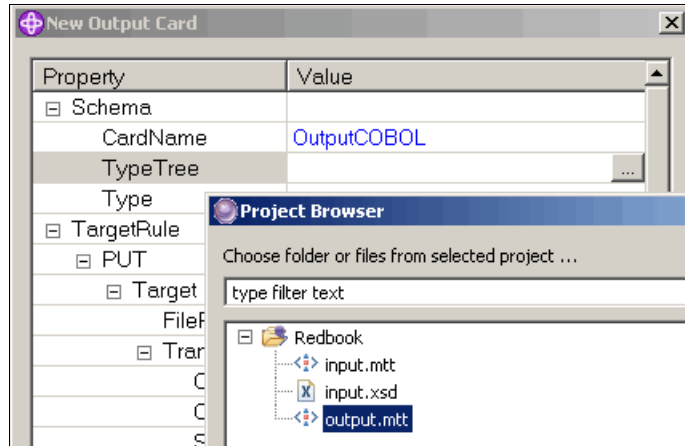


Figure 21-18 Set property “TypeTree” for Output Card

3. For the property Type, select **OutputFile** from CopyBook (see Figure 21-19).

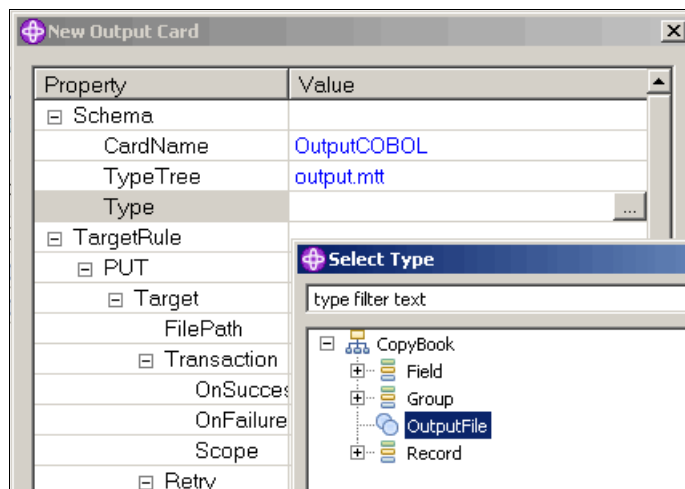


Figure 21-19 Set property Type for the Output Card

4. Finally, set `output.txt` for the `FilePath` property (see Figure 21-20), and click **OK**.

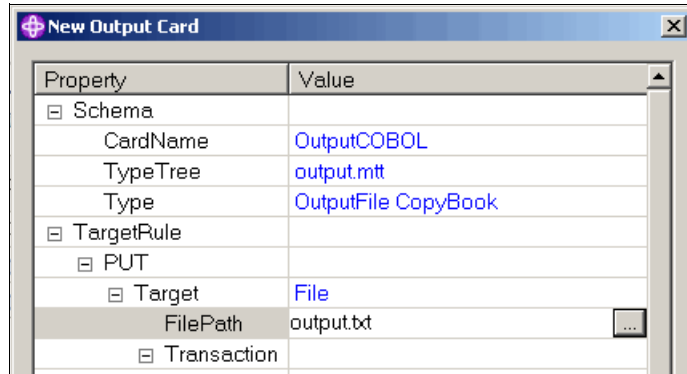


Figure 21-20 Set property `FilePath` for the Output Card

Figure 21-21 shows the content of `XML2CobolMap` with the Input Card on the left side and the Output Card on the right side.

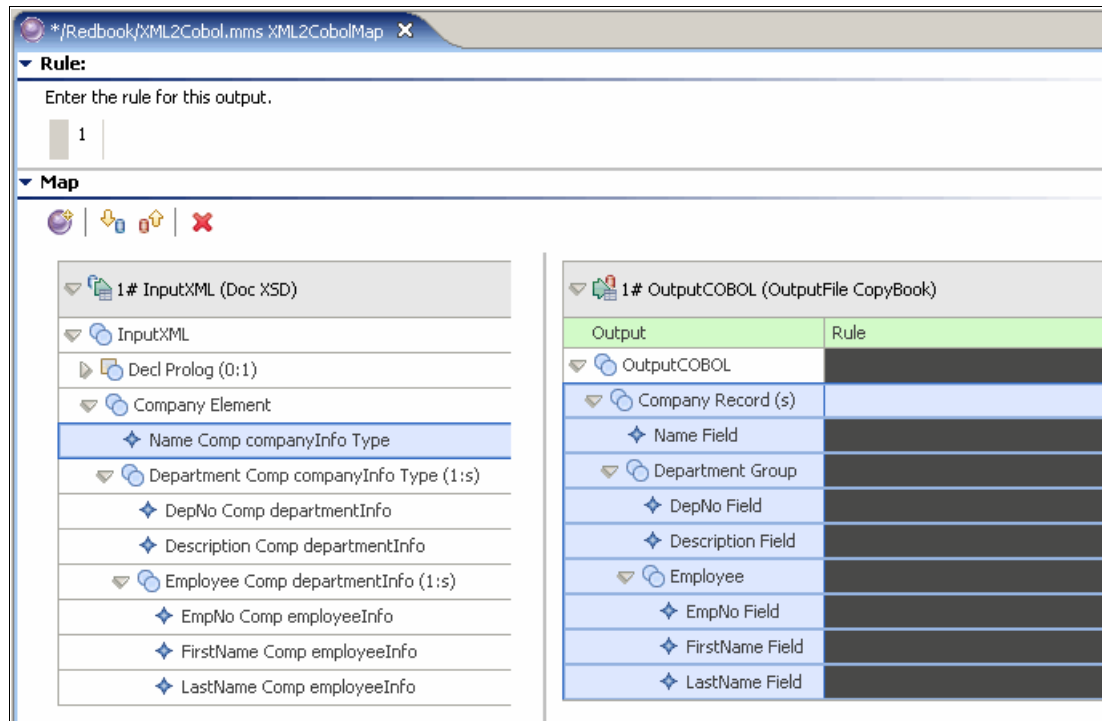


Figure 21-21 Content of `XML2CobolMap`

Creating the map

To create the map, you can drag-and-drop fields from the left to the right side to define which input field matches the corresponding output field. However, as shown in Figure 21-21, all fields except `Company Record(s)` on the right side are locked. The reason is that on the left side one single input XML data structure should map to many COBOL copybook rows.

To solve this, create a functional map, a kind of sub-routine, that is called for each element (in this case, for each record) to be created). Click **Company Record(s)** on the right side and then enter =EachCompany() in the Rule panel, as shown in Figure 21-22.

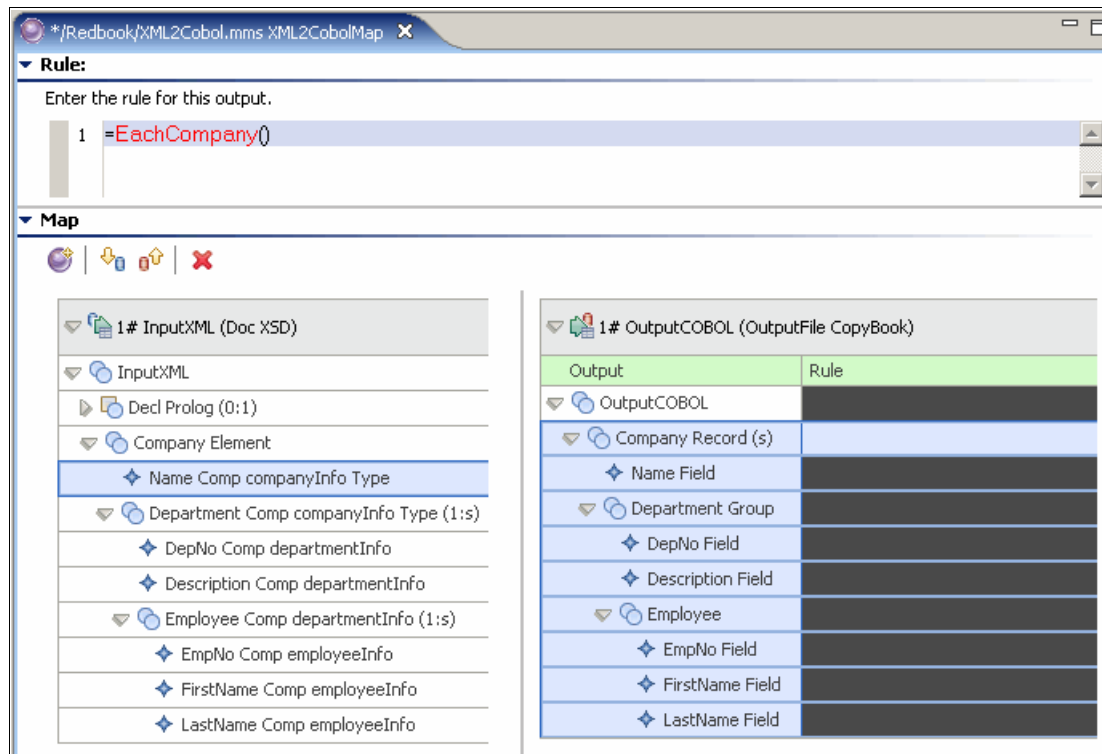


Figure 21-22 Create user-defined WebSphere Transformation Extender function EachCompany

Next, define a parameter for the WebSphere Transformation Extender function using drag-and-drop fields from the Input Card as function parameter. All different parameters must be separated by a comma sign. To define all needed parameters:

1. Drag the left side field “Name Comp companyInfo Type” and drop it as the first parameter of function EachCompany.
2. Append the comma sign followed by a blank (“, ”) after the first parameter.
3. Drag the left side group “Department Comp companyInfo Type(1:s)” and drop it as the second parameter after the comma sign (,).
4. Append the comma sign followed by a blank (“, ”) after the second parameter.
5. Drag the left side group “Employee Comp departmentInfo (1:s)” and drop it as the third parameter after the comma sign (,).
6. Press **Enter** to complete the function definition. Without pressing Enter, your function definition changes are not activated.

Example 21-6 shows the result of the function.

Example 21-6 Content of user-defined WebSphere Transformation Extender function EachCompany

```
=EachCompany(Name Comp companyInfo Type:Company Element:InputXML, Department Comp companyInfo Type:Company Element:InputXML, Employee Comp departmentInfo:Department Comp companyInfo Type:Company Element:InputXML)
```

XML2CobolMap now looks like Figure 21-23.

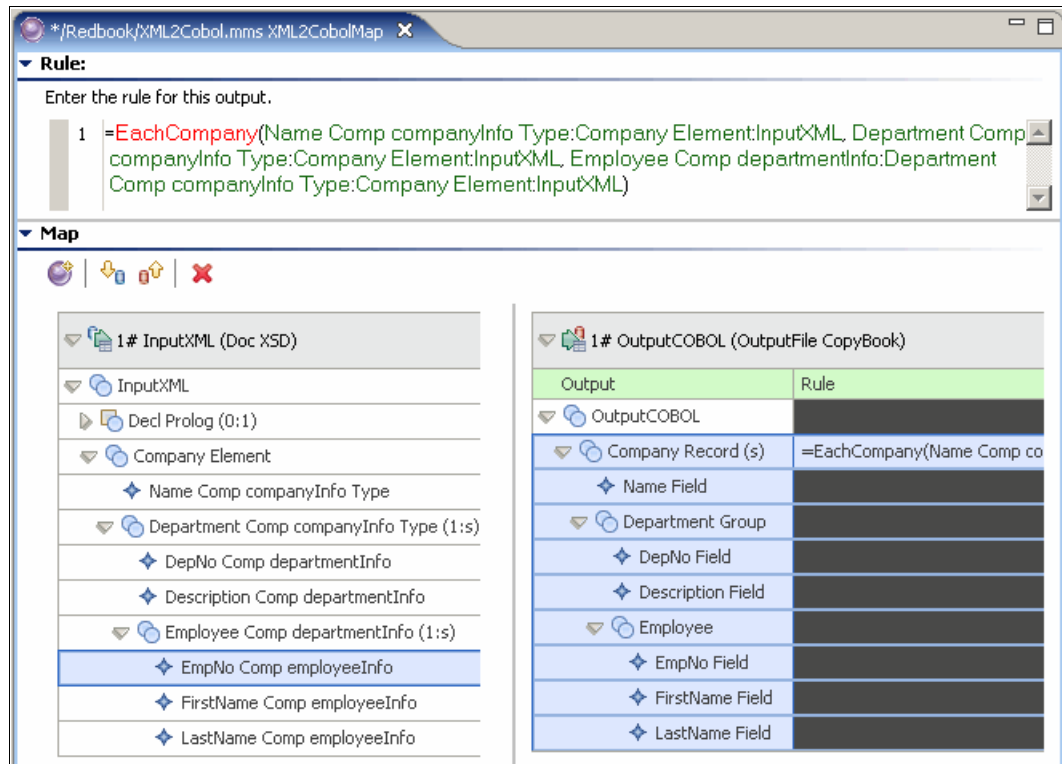


Figure 21-23 XML2CobolMap after creating user-defined function "EachCompany"

Next, create the function EachCompany by right-clicking **Company Record(s)** on the right and choosing **Functional Map Wizard** (see Figure 21-24).

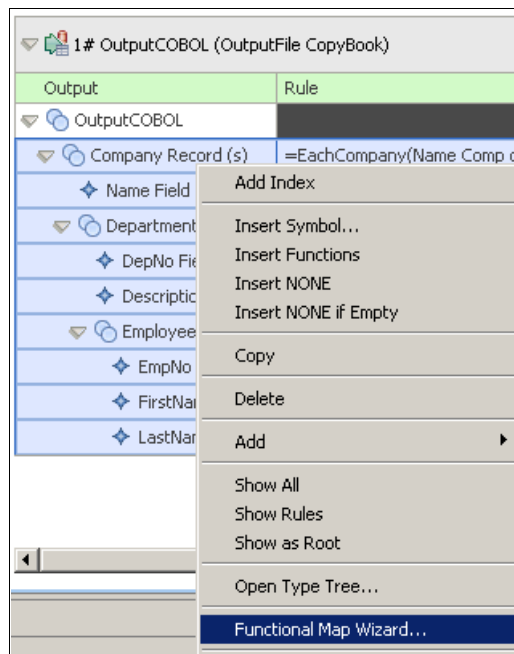


Figure 21-24 Selecting the Functional Map wizard

In the window that opens, click **Create**, and then **Close**. Now the function EachCompany is shown in the map window. Every parameter of EachCompany results in one corresponding Input Card. These cards can be sorted by dragging and dropping the cards within the current window.

Now, you can define the mapping by dragging fields from Input Cards and dropping them on the right side, as follows:

1. Drag field “In1” from input card “In1” to output card field Name Field.
2. Drag field “DepNo Comp departmentInfo” from input card “In2” to output card field DepNo Field.
3. Drag field “Description Comp departmentInfo” from input card “In2” to output card field Description Field.
4. Drag field “EmpNo Comp employeeInfo” from input card “In3” to output card field EmpNo Field.
5. Drag field “FirstName Comp employeeInfo” from input card “In3” to output card field FirstName Field.
6. Drag field “LastName Comp employeeInfo” from input card “In3” to output card field LastName Field.

Now save the map. Figure 21-25 shows the content of map XML2CobolMap.

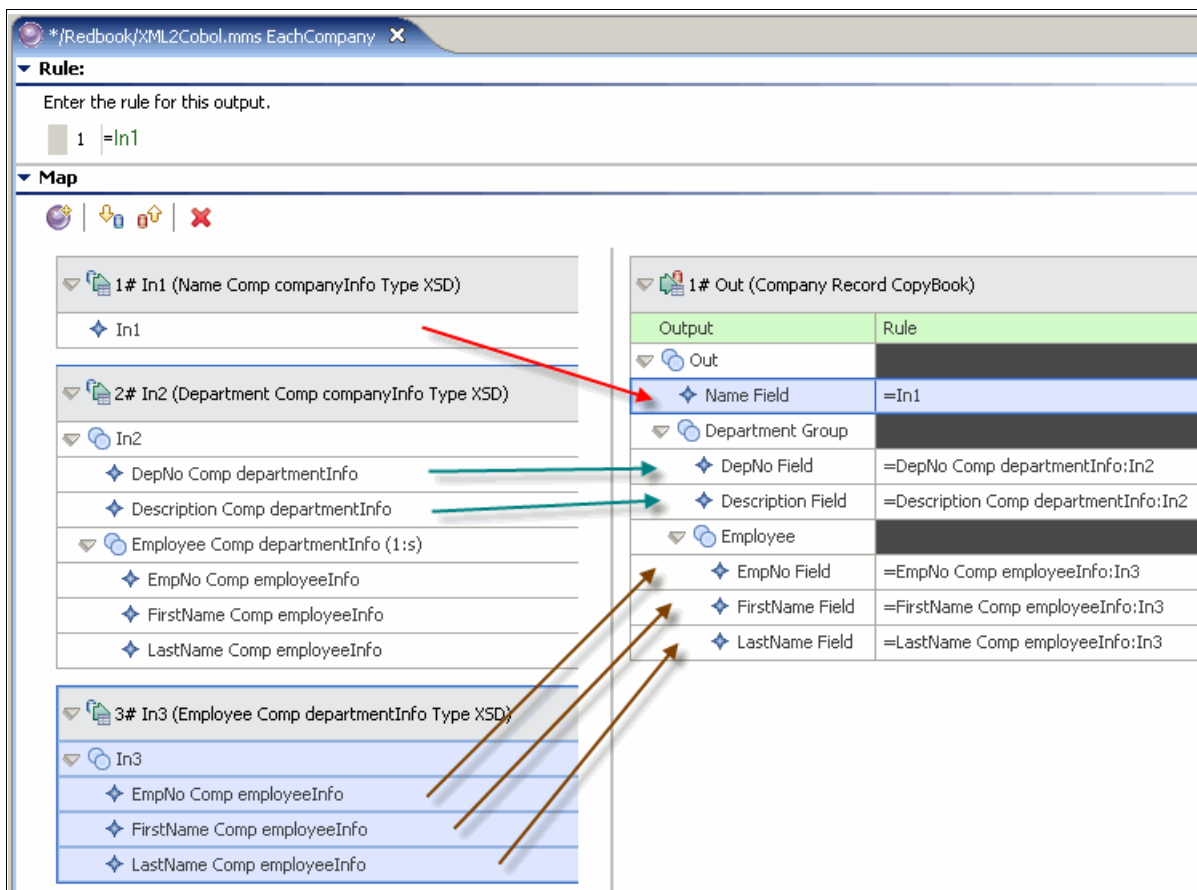


Figure 21-25 Content of XML2CobolMap

Building the map

Reopen the map by double-clicking **XML2CobolMap** in the Outline window (see Figure 21-26).

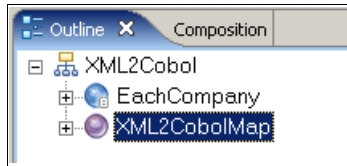


Figure 21-26 Reopen XML2CobolMap

To compile the map, choose **Map** → **Build** to check and build the map to an internal format.

Note: After changing mapping rules or input or output cards, you must call **Map** → **Build** to recompile the map.

Testing the map

To test the transformation process, select **Map** → **Run**. If everything works correctly, you see the message shown in Figure 21-27.

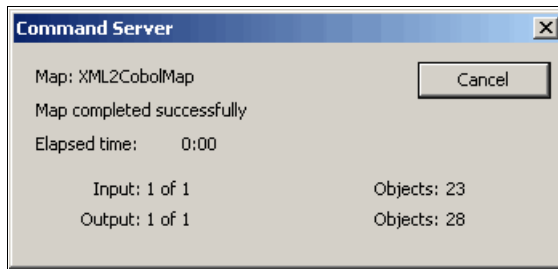


Figure 21-27 Testing the transformation process

We defined our output card to save the transformed data in file `output.txt`. This file is stored in the `Misc` folder (in the Extender Navigator window). Example 21-7 shows the content of our output file. You can view the results of the map execution from within Design Studio. Click **Map** → **View Run Results** for a prompt for the inputs and outputs that you want to view. Select or clear the boxes for the inputs and outputs that you want to review, and then click **OK**.

Example 21-7 Content of transformation output

Big data center	000000001Production	000000001Paul	Smith
Big data center	000000001Production	000000002Hannah	Smith
Big data center	000000002Development	000000003Mia	Brown

This example shows how to transform data from XML format to COBOL copybook format without writing any specific code. All necessary activities can be done by drag-and-drop using WebSphere Transformation Extender Design Studio.

In the next section, 21.3.2, “Transferring files to z/OS” on page 393, we describe how to transfer the mapping file and run the transformation process on the z/OS.

21.3.2 Transferring files to z/OS

The executable map files in internal formats have different extensions, depending on the platform on which they run:

.mmc	for Windows
.mvs	for z/OS
.hpi	for HP-UX Itanium
.hp	for HP-UX PA-RISC)
.aix	for IBM AIX
.lnx	for Linux Intel
.sun	for Sun Solaris

Creating a z/OS specific mapping file

Because our COBOL batch program is running on z/OS, we first must create a mainframe-specific executable mapping file by choosing **Map** → **Build for Specific Platform** → **IBM(TM) z/OS** (see Figure 21-28).

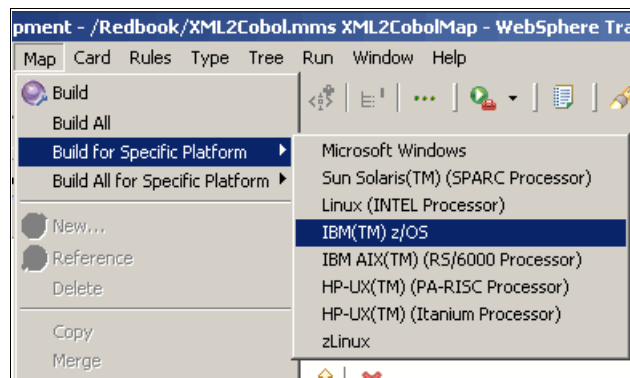


Figure 21-28 Creating z/OS specific mapping file

By using the build command for z/OS map, the compiled map is placed in the Map Executable folder, and a JCL template is generated and placed in the JCL Files folder in WebSphere Transformation Extender Design Studio. This JCL template must be modified to use the map and files that are used.

Creating a PDS data set on z/OS

In our example, the z/OS specific mapping file must be copied to a user-defined PDS data set, which in our case is called WAGNERA.WTX.MAPS. We pre-allocated this data set with the following options:

Data Set Name:	WAGNERA.WTX.MAPS
Space units:	Track
Primary quantity:	10
Secondary quantity:	10
Directory blocks:	10
Record format:	FB
Record length:	80
Block size:	16000
Data set name type:	PDS

Transferring files to a z/OS system

You have to transfer the compiled map file XML2Cobo1Map.mvs (found in the Extender Navigator window under the Map Executables folder) in binary mode to the z/OS system as a sequential

data set or member of a PDS. The record format has to Fixed Block (FB) and have the Logical Record Length LRECL) must be 80.

Furthermore, you have to transfer the input XML data `input.xml` file and XML schema definition file `input.xsd`. These files must be transferred in binary mode to the UNIX System Services file system. In our example, we transferred these files to the UNIX System Services directory `/u/wagnera`.

When transferring files, keep in mind the following aspects:

- ▶ Specify BINARY, not performing ASCII to EBCDIC translation and not eliminating carriage returns and line feeds). If you are using an FTP tool, double-check the options, because most tools will not have any pre-settings for files with file type `.mvs`, and the file transfer outcome might be a random result.
- ▶ The file transfer must not use the CRLF option, or map corruption will result. Instead, use either of the following options:
 - Use the IBM Personal Communication Facility.
 - Use standard FTP, which is available from a Windows command line.

Example 21-8 shows a proper FTP file transfer from the Windows command line.

Example 21-8 FTP commands to load the map file

```
ftp <your-host>
UserID
Password
binary
cd /u/wagnera
put C:\<WTX_workspace>\Redbook\input.xml input.xml
put C:\<WTX_workspace>\Redbook\input.xsd input.xsd
cd 'WAGNERA.WTX.MAPS'
quote site LRECL=80 RECFM=FB
put C:\<WTX_workspace>\Redbook\XML2CobolMap.mvs XML2COB
quit
```

The quote site command: By using the FTP `quote site` command, you can set the file type and allocation on the host.

21.3.3 Running the job on z/OS to transform input data

To start a transformation process, you have to submit the corresponding JCL. We describe this JCL in detail in this section. You can find the complete JCL in Example B-13 on page 448.

Note: Our JCL contains some system specific values (such XML Toolkit STEPLIB, Output data set name, and so forth). You must change these values to fit your environment.

Example 21-9 shows the job card that we used.

Example 21-9 Job card

```
//WTXBATCH JOB (999,POK),'L06R',CLASS=A,REGION=OM,
//          MSGCLASS=T,TIME=10,MSGLLEVEL=(1,1),NOTIFY=&SYSUID
```

The first step in the job deletes the output data set (in our example called WAGNERA.OUT.XML2COB0), as shown in Example 21-10.

Example 21-10 Deleting the output data set

```
//SO EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DELETE WAGNERA.OUT.XML2COB0
```

The next step starts a batch transformation process by running the program DTXCMDSV. Because we are transforming XML input data, we also need the XML Toolkit in the STEPLIB, as shown in Example 21-11.

Example 21-11 Starting the batch transformation process

```
//DTX EXEC PGM=DTXCMDSV,REGION=0M,
// PARM='XML2COB /IIM '/DD:MAPXSD''
//STEPLIB DD DISP=SHR,DSN=BB26159.SDTXLOAD
// DD DISP=SHR,DSN=IXM.SIXMLOD1
```

The program DTXCMDSV is started with two parameters:

XML2COB DD name for the executable map file

/IIM ‘’/DD:MAPXSD’’’ Because our input Type Tree was created in WebSphere Transformation Extender Design Studio by importing a XML schema definition, we need this schema definition file when running our batch transformation process. With this parameter we define the DD name for our XML schema definition file.

Example 21-12 shows a DD name pointing to the WebSphere Transformation Extender map. This is the PDS that we pre-allocated earlier and that we used to store the map. As mentioned previously, the first parameter for program DTXCMDSV defines the DD name XML2COB for the executable map file.

Example 21-12 Path to executable map file

```
//XML2COB DD DISP=SHR,DSN=WAGNERA.WTX.MAPS(XML2COB)
```

The next DD statements point to the necessary files for our batch transformation program, as shown in Example 21-13.

INPUT UNIX System Services file with input XML data. This file must be copied binary to the UNIX System Services file system and must contain the option “encoding” with the correct value.

MAPXSD The second parameter for program DTXCMDSV defines the DD name MAPXSD for the XML schema definition file. This file must also be copied binary to UNIX System Services file system.

OUTPUT For our COBOL batch program, we write the output data to an MVS data set with copybook-style records.

Example 21-13 Input and output definitions

```
//* Define the input dataset
//INPUT DD PATH='/u/wagnera/input.xml'
//*
//* Define the schema definition
```

```
//MAPXSD DD PATH='/u/wagnera/input.xsd'
//*
//* Define the output dataset
//OUTPUT DD DSN=WAGNERA.OUT.XML2COBO,
//          DCB=(RECFM=VB,LRECL=80),
//          UNIT=SYSDA,
//          SPACE=(TRK,(1,1),RLSE),
//          DISP=(NEW,CATLG,DELETE)
```

Important: The output data set must be a variable record length file, because our output Type Tree was configured to produce a line feed after every employee record. When using FB format, no line feed is produced.

The number of temporary files to create depends on the characteristics of the map itself. It is not always obvious to determine the exact number and the sizes to allocate. For complex maps or for big files, some tests for adjustment might be necessary. Allocating the temporary files is key for performance. Example 21-14 shows what we used.


Example 21-14 Temporary data sets

```
//SYSTMP01 DD DSN=&&TEMP01,
//           DISP=(NEW,DELETE,DELETE),
//           DCB=(RECFM=FBS,LRECL=32760),
//           UNIT=SYSDA,
//           SPACE=(TRK,(5,1))
//SYSTMP02 DD DSN=&&TEMP02,
//           DISP=(NEW,DELETE,DELETE),
//           DCB=(RECFM=FBS,LRECL=32760),
//           UNIT=SYSDA,
//           SPACE=(TRK,(5,1))
//SYSTMP03 DD DSN=&&TEMP03,
//           DISP=(NEW,DELETE,DELETE),
.....
```

Pre-allocating work files: Pre-allocating work files is not mandatory for small amounts of data, but we highly recommend it for performance reasons.

Use the following rules to set the number of temporary files:

- ▶ One temporary data set is allocated in all cases, which is a general use work file.
- ▶ One temporary data set is also allocated for each input.
- ▶ Depending on how the map is constructed, the map might create a temporary data set for each output to serve as its work file.
- ▶ In addition, one temporary data set is always allocated for each input or output defined with an RECFM that is not fixed unblocked (F) or fixed blocked standard (FBS).
- ▶ Because you can develop a map with no inputs, and output work files cannot be used in some situations, the minimum number of possible temporary files for a map execution is 1. The maximum possible number is $1 + (2 \times \text{number of inputs}) + (2 \times \text{number of outputs})$.



Reduce batch complexity by eliminating custom file transfer logic

In batch, you typically process lots of input and output data. Therefore, file transfer from and to other systems also plays a very important role. However, standard file transfer lacks the following functionality:

- ▶ If the destination system is suddenly down, custom built logic must be implemented to ensure that the transfer continues or restarts when the destination system is back again.
- ▶ Monitoring must be implemented on top of file transfer.
- ▶ Auditing must be implemented.

This lack of functionality forces companies to build additional logic in batch around file transfer. Therefore, batch complexity grows larger than is optimal.

In this chapter, we show how to use WebSphere MQ File Transfer Edition to address this issue.

This chapter includes the following topics:

- ▶ Using WebSphere MQ FTE to perform managed file transfers
- ▶ Initiating file transfer using a Java job
- ▶ Initiating file transfer using an Ant job
- ▶ Summary

22.1 Using WebSphere MQ FTE to perform managed file transfers

Because of custom implementations around file transfer, you can use WebSphere MQ File Transfer Edition (FTE) to remove this complexity.

Figure 22-1 illustrates how WebSphere MQ FTE works.

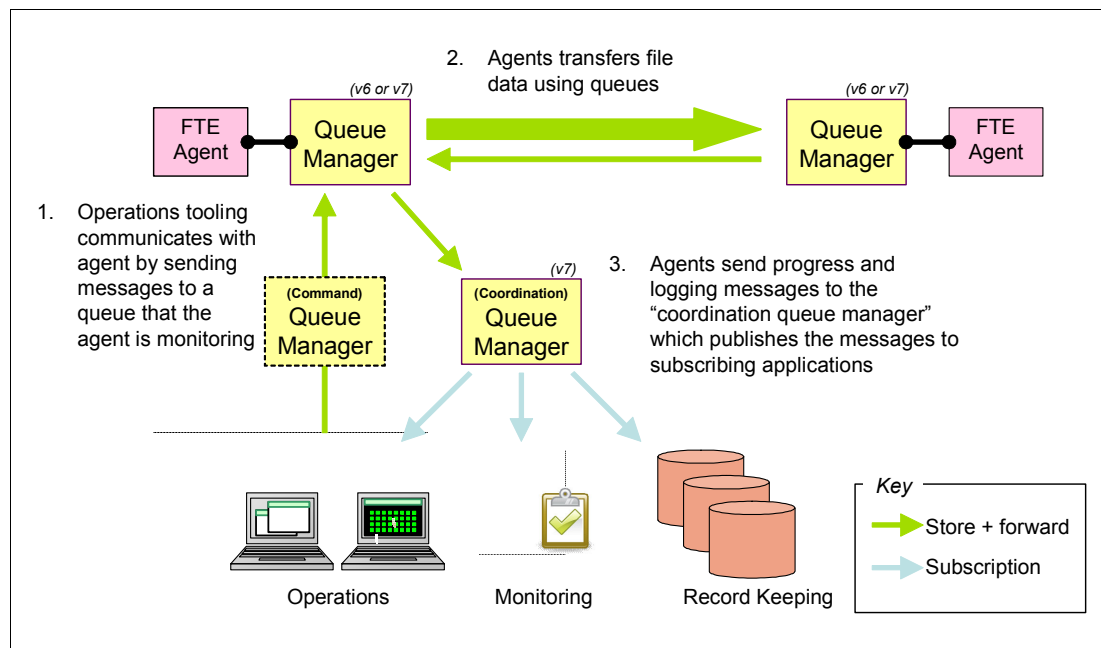


Figure 22-1 WebSphere MQ FTE overview

WebSphere MQ FTE consists of the following components:

- ▶ FTE agent

The FTE agent processes are long running Java processes that manage transferring files on and off of a system. Thus, every file transfer operation has an agent as its source and an agent as its destination.

The agent at the source end of the transfer is responsible for reading files from the source system, converting them into a stream of messages. This stream of messages is transferred, using MQ, to queues monitored by the destination agent for the transfer.

- ▶ Queue manager

WebSphere MQ provides standard queue managers that perform the message transport.

- ▶ Operational interfaces

To control the file transfer edition agents, WebSphere MQ FTE comes with operations tooling in the form of a set of command-line commands and a graphical MQ Explorer plug-in. With this operational tooling, you can define the patterns of which files are transferred where and how.

Although most of the operations functionality is common to both the command-line tools and the MQ Explorer plug-in, the command-line tools can also carry out administrative functions. Examples of these administrative functions include defining new agents, displaying all configured agents, and starting and stopping agent processes.

A typical flow of a sample file transfer might look like as follows:

1. The Operations tooling communicates with the agent on the left by sending messages to the queue that the agent is monitoring.
2. The agents transfer the file data using queues to the destination agent which stores the files.

While this process is running, the agents send progress and logging messages to the coordination queue manager, which publishes the messages to other subscribing applications.

The big advantage of WebSphere MQ FTE is that it eliminates custom logic around restart and resume in case a destination system is down because it relies on the functionalities of standard WebSphere MQ. WebSphere MQ FTE provides this functionality as it ships, thus the entire transfer is managed automatically by the infrastructure.

In addition, WebSphere MQ FTE can also eliminate custom file transfer monitoring and logging or auditing. WebSphere MQ FTE Logging is done by the transfer log entries which record transfers as they flow through the system. The log entries tell us:

- ▶ When a transfer starts
- ▶ How a transfer is progressing
- ▶ If a transfer has completed or is cancelled

The transfer log also records any malformed input message for diagnostic purposes.

There is a single transfer log for all agents in the system which resides within the coordination queue manager as topic `SYSTEM.FTE/1og`. Each message is an XML document that is defined by an XML schema.

The WebSphere MQ FTE user interface on a Windows or Linux workstation then shows a formatted representation on the transfer log view, as shown in Figure 22-2.

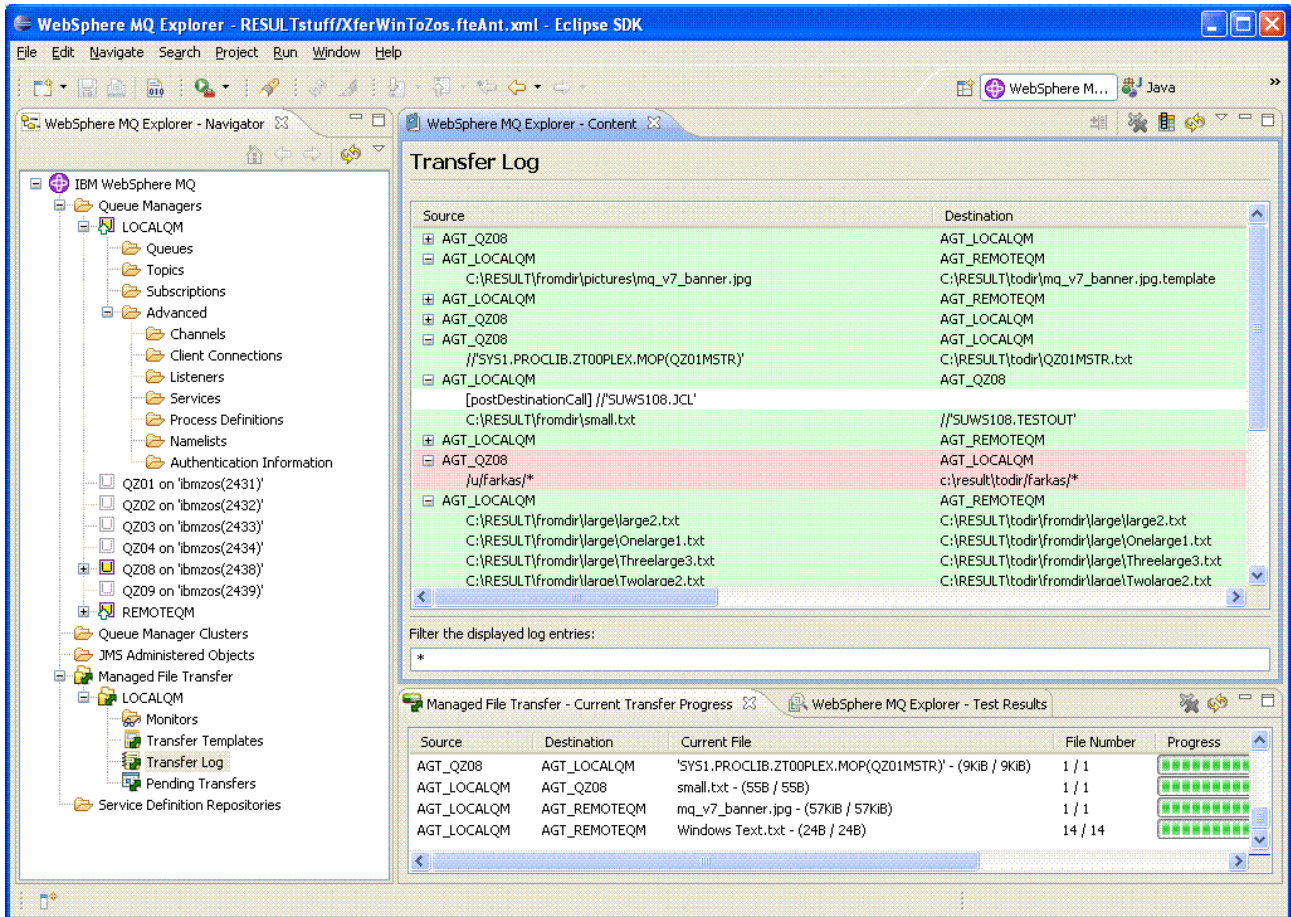


Figure 22-2 FTE GUI

Figure 22-3 show how this might look on the FTE user interface.

ID	TRANSFER_ID	FILE_MO...	SOURCE_FILENAME	SOURCE...	DESTINATION_FILENAME	SOU...
6	414d51204c4f43414c51...	text	//SYS1.PROCLIB.ZT00PLEX.MOP(QZ01MSTR)	MD5	... C:\RESULT\tdir\QZ01MST...	IBM-
9	414d51204c4f43414c51...	text	//SYS1.PROCLIB.ZT00PLEX.MOP(QZ01MSTR)	MD5	... C:\RESULT\tdir\QZ01MST...	IBM-
13	414d51204c4f43414c51...	binary	//SYS1.PROCLIB.ZT00PLEX.MOP(QZ01MSTR)	MD5	... C:\RESULT\tdir\QZ01MST...	
22	414d51204c4f43414c51...	binary	C:\RESULT\fromdir\large\large2.txt	MD5	... C:\RESULT\tdir\fromdir\larg...	
23	414d51204c4f43414c51...	binary	C:\RESULT\fromdir\large\lone\large1.txt	MD5	... C:\RESULT\tdir\fromdir\larg...	
24	414d51204c4f43414c51...	binary	C:\RESULT\fromdir\large\lthree\large3.txt	MD5	... C:\RESULT\tdir\fromdir\larg...	
25	414d51204c4f43414c51...	binary	C:\RESULT\fromdir\large\ltwo\large2.txt	MD5	... C:\RESULT\tdir\fromdir\larg...	
18	414d51204c4f43414c51...	binary	C:\RESULT\fromdir\pictures\030909_Smart_Planet_Cities_Fla...	MD5	... /u/farkas/030909_Smart_Pla...	
26	414d51204c4f43414c51...	binary	C:\RESULT\fromdir\pictures\030909_Smart_Planet_Cities_Fla...	MD5	... C:\RESULT\tdir\fromdir\pict...	
20	414d51204c4f43414c51...	binary	C:\RESULT\fromdir\pictures\mq_v7_banner_trigger_off.jpg	MD5	... /u/farkas/mq_v7_banner_trig...	
28	414d51204c4f43414c51...	binary	C:\RESULT\fromdir\pictures\mq_v7_banner_trigger_off.jpg	MD5	... C:\RESULT\tdir\fromdir\pict...	
2	414d51204c4f43414c51...	binary	C:\RESULT\fromdir\pictures\mq_v7_banner.jpg	MD5	... C:\RESULT\tdir\mq_v7_ba...	
4	414d51204c4f43414c51...	binary	C:\RESULT\fromdir\pictures\mq_v7_banner.jpg	MD5	... C:\RESULT\tdir\mq_v7_ba...	
7	414d51204c4f43414c51...	binary	C:\RESULT\fromdir\pictures\mq_v7_banner.jpg	MD5	... C:\RESULT\tdir\mq_v7_ba...	
8	414d51204c4f43414c51...	binary	C:\RESULT\fromdir\pictures\mq_v7_banner.jpg	MD5	... C:\RESULT\tdir\mq_v7_ba...	
11	414d51204c4f43414c51...	binary	C:\RESULT\fromdir\pictures\mq_v7_banner.jpg	MD5	... C:\RESULT\tdir\mq_v7_ba...	
15	414d51204c4f43414c51...	binary	C:\RESULT\fromdir\pictures\mq_v7_banner.jpg	MD5	... C:\RESULT\tdir\mq_v7_out...	
19	414d51204c4f43414c51...	binary	C:\RESULT\fromdir\pictures\mq_v7_banner.jpg	MD5	... /u/farkas/mq_v7_banner.jpg	

Figure 22-3 FTE Database Logger

Because messages are well formed, we can also write custom applications to read and process transfer log messages, for example for audit purposes. Compared to normal FTP, this kind of logging and auditing is much more effective.

For more information about WebSphere MQ FTE, see *IBM WebSphere MQ File Transfer Edition Solution Overview*, REDP-4532.

In the context of batch, it is also important to know that WebSphere MQ FTE supports z/OS UNIX System Services files as well as z/OS data sets. However, for a real batch integration, you have to trigger the transfer with a job instead of with the operational tooling. Therefore, you have the following options:

- ▶ Using a standard Java job that submits the file transfer command to WebSphere MQ FTE
- ▶ Using an Ant job

We discuss these options in the remainder of this chapter.

22.2 Initiating file transfer using a Java job

A sample file transfer initiation by a job might look like that shown in Example 22-1.

Example 22-1 Issuing a WebSphere MQ FTE transfer command

```
//STRAUERA JOB
/*JOBPARM SYSAFF=SC49
//JAVAJVM EXEC PGM=JVMLDM50,REGION=0M,PARM='+T'
//STEPLIB DD DSN=SYS1.SIEALNKE,DISP=SHR
// DD DSN=MQ700.V090210.SCSQAUTH,DISP=SHR
// DD DSN=MQ700.V090210.SCSQANLE,DISP=SHR
```

```

//          DD DSN=MQ700.V090210.SCSQLOAD,DISP=SHR
//SYSPRINT DD SYSOUT=*          < System stdout
//SYSOUT   DD SYSOUT=*          < System stderr
//STDOUT   DD SYSOUT=*          < Java System.out
//STDERR   DD SYSOUT=*          < Java System.err
//CEEDUMP  DD SYSOUT=*
//ABNLIGNR DD DUMMY
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//STDENV   DD *
# This is a shell script that configures
# any environment variables for the Java JVM.
# Variables must be exported to be seen by the launcher.
# Use PARM='+T' and set -x to debug environment script problems
set x
. /etc/profile
#
# Java configuration (including MQ Java interface)
export JAVA_HOME=/usr/lpp/java/J5.0
export PATH=/bin:${JAVA_HOME}/bin
LIBPATH="${LIBPATH}:${JAVA_HOME}/bin
LIBPATH="${LIBPATH}:${JAVA_HOME}/bin/classic
LIBPATH="${LIBPATH}:/pp/mqmv7/V090210/java/lib/
export LIBPATH
#
# FTE configuration
export FTE_PROD=/usr/lpp/mqmfte/V7ROM1/mqhg
export FTE_CONFIG=/var/mqmfte
export _BPXK_AUTOCVT=ON
#
# Select function to be executed (script names without fte prefix)
#
. ${FTE_PROD}/bin/fteBatch CreateTransfer
#
# Set JZOS parameters to FTE values
export IBM_JAVA_OPTIONS="${FTE_JAVA_OPTIONS}"
export JZOS_MAIN_ARGS="${FTE_MAIN_ARGS}"
//MAINARGS DD *
-da CHICAGO.AGENT2 -dm MQH2 -sa CHICAGO.AGENT -sm MQH1 -t text -w
-ds "'/STRAUER.SPUFI.OUT'" -de overwrite
"'/STRAUER.SPUFI.OUT2'"
//

```

This sample calls the JZOS batch launcher (see 7.2, “Running Java with JZOS” on page 79 for more details) and invokes a WebSphere MQ FTE command. The WebSphere MQ FTE command is passed by the MAINARGS DD statement and transfers a single data set from agent CHICAGO.AGENT to agent CHICAGO.AGENT2. The -w parameter is included so that the step waits for the transfer command to complete. The step return code contains the return code from the invoked function.

You need to adjust the following parameters in the JCL:

- ▶ STEPLIB to include the JZOS and MQ load modules
- ▶ JAVA_HOME to point to the Java home directory in UNIX System Services
- ▶ LIBPATH to include the MQ Java libraries

- ▶ FTE_PROD to point to the FTE production directory in UNIX System Services
- ▶ FTE_CONFIG to point the WebSphere MQ FTE config directory under UNIX System Services

After submitting the job, you can read the file transfer results in the job output.

Note: Because the command is submitted using a Java job, it requires a JVM startup. See 18.2, “Stand-alone Java batch” on page 339 for performance aspects of JVM startup.

22.3 Initiating file transfer using an Ant job

Another option to initiate a WebSphere MQ FTE command in batch is to use Ant in combination with BPXBATCH.

In addition to the standard Ant tasks, such as copy, delete, or mkdir, WebSphere MQ FTE offers extensions to Ant. These extensions, in combination with the standard Ant tasks, allow you to send commands to WebSphere MQ FTE agents using WebSphere MQ. Those extensions include, for example:

```
fte:copy
fte:move
fte:invoke
```

Example 22-2 shows a basic Ant script.

Example 22-2 A sample Ant script, antTest.xml

```
<?xml version="1.0"?>
<project name="Project1" default="main" basedir="/u/strauer/ant"
xmlns:fte="antlib:com.ibm.wmqfte.ant.taskdefs">
  <target name="init">
    <property name="srcfile" value="file1.txt"/>
    <property name="dstfile" value="file1_new.txt"/>
    <property name="Name" value="Demo System"/>
  </target>
  <target name="step1" depends="init">
    <echo>Starting build of ${Name}</echo>
  </target>
  <target name="step2" depends="step1">
    <fte:filecopy src="CHICAGO.AGENT@MQH1" dst="CHICAGO.AGENT2@MQH2"
rcproperty="copy.result">
    <fte:filespec srcfilespec="${srcfile}" dstfile="${dstfile}"/>
    </fte:filecopy>
  </target>
  <target name="step3" depends="step2">
    <echo>Finished build of ${Name}</echo>
  </target>
  <target name="main" description="runs this script">
    <antcall target="step3"/>
  </target>
</project>
```

This script performs the following steps:

1. Printing out that the script is starting.
2. Copying a file from /u/strauer/file1.txt to /u/strauer/file1_new.txt.
3. Printing out that the script has finished.

All these steps depend on each other.

Compared to the job in 22.2, “Initiating file transfer using a Java job” on page 401, Ant allows you to easily create very sophisticated file transfer scripts that include dependencies. Furthermore, Ant is a platform-independent, open standard compared to JCL.

To launch the Ant script in batch, you can use BPXBATCH. First, ensure that the environment variables FTE_JAVA_HOME and PATH are set correct in the .profile file, as follows:

```
export PATH=$PATH:/pp/mqmfte71/UK00000/bin
export FTE_JAVA_HOME=/usr/lpp/java/J5.0
```

Now, you can call the **fteAnt** command under <FTE_HOME>/bin using a BPXBATCH job, as shown in Example 22-3.

Note: The **fteAnt** command is tagged ASCII. Therefore, you need to make sure AUTOCVT is set to ON.

Example 22-3 Launching Ant with BPXBATCH

```
//STRAUERB JOB
/*JOBPARM SYSAFF=SC49
//STEP1 EXEC PGM=BPXBATCH,REGION=0M,
// PARM='SH fteAnt -f /u/strauer/ant/antTest.xml'
//STDOUT DD SYSOUT=*
//STDERR DD SYSOUT=*
//*
```

This job calls the Ant script in a batch job.

Note: Using Ant and BPXBATCH also requires a JVM startup. See 18.2, “Stand-alone Java batch” on page 339 for performance aspects of JVM startup.

22.4 Summary

WebSphere MQ FTE allows you to easily perform a file transfer that is completely managed. For example, if a system in fails, WebSphere MQ FTE makes sure that the transfer is completed later. Thus, you can reduce batch complexity by eliminating custom written file transfer management logic. In addition, you can integrate WebSphere MQ FTE in batch using a Java job or Ant.



Reduce complexity by exploiting DFSORT / ICETOOL

DFSORT has evolved far beyond simple sorting. This chapter demonstrates why you might consider DFSORT functions to replace batch steps that you have written yourself to perform tasks that can be done by standard functions and features provided by the z/OS environment. You might want to consider DFSORT for the following reasons:

- ▶ DFSORT statements are less complex to code and maintain than self-written code.
- ▶ DFSORT functions have the potential to be more computationally efficient than self-written code.

In this chapter, we discuss how you can build applications from the standard DFSORT and ICETOOL capabilities.

Note: The functions for DFSORT have been extended greatly over the last few years through formal releases and new-function APARs, which are usually released annually. This chapter assumes that you have the latest functions available.

Note: In this chapter the term *OUTFIL* means the *OUTFIL* statement that is used with the DFSORT program. The term *DFSORT OUTFIL* is a more formal, equivalent term.

This chapter includes the following topics:

- ▶ Invoking DFSORT functions
- ▶ Data that DFSORT can process
- ▶ Beyond sorting
- ▶ Symbols
- ▶ Invoking DFSORT from Java with JZOS

23.1 Invoking DFSORT functions

You can invoke DFSORT in a number of different ways:

- ▶ As direct program statements, using SYSIN
- ▶ Using ICETOOL and its own program statements
- ▶ Using REXX
- ▶ Using Assembler
- ▶ Using compiled high-level languages such as COBOL
- ▶ Using JZOS
- ▶ As an automatically-invoked replacement for IEBGENER (using ICEGENER)

23.2 Data that DFSORT can process

DFSORT can read from and write to data sets of the following types:

- ▶ Sequential data sets, whether on disk or tape (including a member of a PDS or PDSE)
- ▶ BatchPipes/MVS pipes
- ▶ VSAM files
- ▶ SPOOL files

Data can be *Fixed*, *Variable*, or *Variable Spanned* (except some types do not support Variable Spanned data).

DFSORT has a good reputation for supporting new device types and data set configurations, such as DFSMS Extended Format Sequential (which supports Striping and Compression) for input and output data sets.

DFSORT cannot directly process database data, such as DB2 or IMS. Such data has to be extracted to flat files first or written from flat files afterwards.

23.3 Beyond sorting

You can use DFSORT for other functions beyond sorting, merging, and copying data. The following resources provide details:

- ▶ The article *Beyond Sorting*, which is available at:
<http://www.ibm.com/support/docview.wss?rs=114&uid=isg3T7000091>
- ▶ The article *Smart DFSORT Tricks*, which is available at:
<http://www.ibm.com/support/docview.wss?rs=114&uid=isg3T7000094>

Note: You can invoke many of DFSORT capabilities using DFSORT directly. You can invoke some using ICETOOL, which is part of the DFSORT program product, to invoke DFSORT transparently.

Fundamentally, DFSORT takes records from a data set (or a concatenation of data sets) and processes them and writes the results, whether that processing involves a sort or is more akin to copying. (The MERGE operation takes multiple data sets and merges them, post-processing the merged records and writing the results.)

Where speed, simplicity, or maintainability are important, both new and existing application components that meet this pattern should be examined to see whether DFSORT functions are a good match.

In this section, we describe some of the more uncommon functions of DFSORT.

23.3.1 Record selection

DFSORT has very sophisticated record selection capabilities, whether before any sort (INCLUDE / OMIT) or after the sort (OUTFIL INCLUDE / OMIT).

With OUTFIL INCLUDE / OMIT, you can use different criteria to send records to multiple output destinations, as Figure 23-1 shows. In this example, records are selected before the actual sort with INCLUDE and then reformatted with INREC. Then, the sort data is written to the following destinations:

- ▶ SORTOUT
- ▶ OUTFIL INCLUDE 1
- ▶ OUTFIL INCLUDE 2

The data written in each case is different, whether because different records are selected to be written or different there is reformatting for the same record.

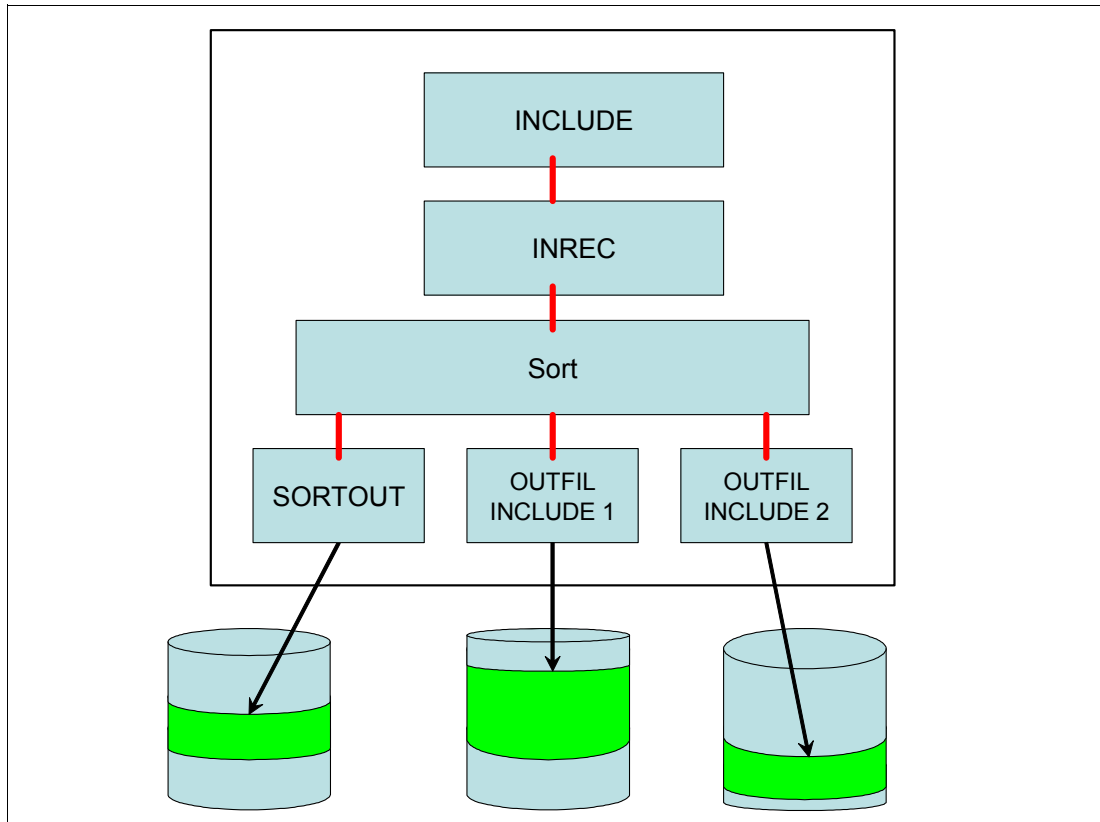


Figure 23-1 OUTFIL allows additional data outputs

The following selection criteria can be used individually or together, using AND or OR logic.

- ▶ Fields can be compared to other fields or to constant values, whether numeric or character data.
- ▶ Fields can be compared to dates and times, including using simple logic such as comparing to yesterday's date.
- ▶ Substring comparisons can be made, for when the exact position of a match isn't known in a string.
- ▶ Bit-level comparisons can be made.
- ▶ Records can be selected based on whether they have valid EBCDIC or Packed-Decimal numerics in a specific field, which can be especially helpful for checking whether data has broken numerics in it, a common source of batch job failures.
- ▶ The ICETOOL SELECT operator writes records that meet (or fail to meet) specific criteria as a group of records, such as "all records with duplicate values for a certain field occur in more than one record" (ALLDUPS). You can reformat the selected records.
- ▶ With IFTHEN processing a flag can be conditionally injected into the records that can then be used to keep or delete them.

23.3.2 Record reformatting

DFSORT has versatile record editing capabilities, whether using INREC on input or OUTREC or OUTFIL on output. Unless you are writing to multiple destinations or need some specific OUTFIL feature, it is recommended that you use the OUTREC statement rather than the OUTFIL statement.

Here are some examples of reformatting capabilities:

- ▶ Overlaying fields
- ▶ Adding fields at the end of records
- ▶ Moving fields
- ▶ Finding and replacing strings, both as substrings and as direct lookups
- ▶ Manipulate timestamps and add the current time and date into records
- ▶ Reformat fields, such as converting to and from hexadecimal, translating from lower case to upper case, and number reformatting
- ▶ Perform arithmetic operations, such as adding two numeric fields together to create a new field
- ▶ Inserting sequence numbers
- ▶ Left- and right-justifying fields
- ▶ Converting between variable-length and fixed-length records and vice versa

With IFTHEN, you can selectively reformat records, based on whether the record meets specific criteria.

Originally, DFSORT only reformatted records by explicitly specifying all the fields. With OVERLAY, you can now specify just the fields to be changed, which simplifies many reformatting applications.

23.3.3 Parsing

Traditionally, DFSORT and ICETOOL worked only with fields in fixed positions with fixed lengths in the record. With PARSE and IFTHEN PARSE, you can extract up to 100 delimited fields and use them wherever you previously used fixed fields.

A delimited field is one that begins somewhere after one delimiting character and ends somewhere before another delimiting character.

Figure 23-2 shows how to use PARSE to move variable-position and variable-length fields into fixed positions and edited to yield fixed-length fields.

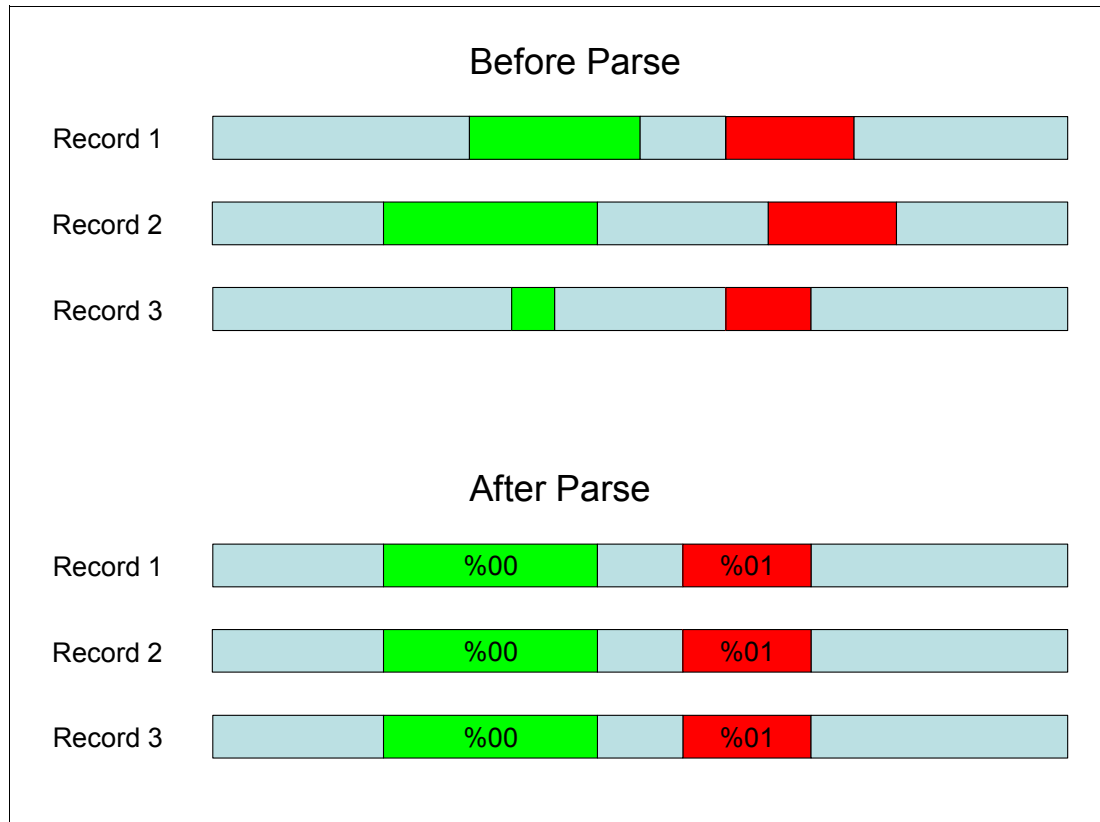


Figure 23-2 How PARSE moves and reformats fields to fixed positions

The following example, from the article *DFSORT: Beyond Sorting*, shows parsing and reformatting comma separated values (CSV):

```
%01=(ENDBEFR=C',' ,FIXLEN=8),
%=(ENDBEFR=C',' ,'),
%03=(ENDBEFR=C',' ,FIXLEN=5),
%04=(FIXLEN=12)),
BUILD=(%01,14:%03,TRAN=LTOU,25:%04,UFF,T0=FS,LENGTH=15)
```

The PARSE operand defines the rules for extracting the delimited fields from each record into %nn fixed parsed fields as follows:

- ▶ The first CSV field is extracted into the 8-byte %01 field. If the extracted data is less than 8 bytes, it is padded on the right with blanks.
- ▶ The second CSV field is ignored.

- ▶ The third CSV field is extracted into the 5-byte %03 field. If the extracted data is less than 5 bytes, it is padded on the right with blanks.
- ▶ The fourth CSV field is extracted into the 12-byte %04 field. If the extracted data is less than 12 bytes, it is padded on the right with blanks.

The BUILD operand reformats the records using the %nn fields as follows:

- ▶ Output positions 1-8 contains the data in the %01 parsed field.
- ▶ Output positions 14-18 contains the data in the %03 parsed field translated from lowercase to uppercase.
- ▶ Output positions 25-39 contains the numeric digits from the %04 parsed field converted to FS format.

23.3.4 Report writing

Both ICETOOL and OUTFIL can be used to write reports. ICETOOL is simple to use and is recommended for reports where the sophistication of OUTFIL is not required. OUTFIL can produce highly detailed reports that contain the following levels:

- ▶ Report-level, with report elements appearing once per report
- ▶ Page-level, with report elements appearing on each page
- ▶ Section-level, with report elements appearing once per section

A section break occurs when the value of specified fields changes, for example for a new value of the Month field.

Both trailers and headers can be created, with or without detail lines in between. Detail lines come from individual records, including those which did not cause the section break.

Headers can contain information such as the date and time the report was created, the new value for the field that caused the section break, and the page number.

Trailers can contain the same information as headers but also information that can only be computed at the end of a section such as averages, totals, and counts.

23.3.5 Record combining

Most DFSORT and ICETOOL operations treat each record individually. There are, however, some functions which combine records:

- ▶ OUTFIL, as previously noted, allows computations such as averages, totals and counts to be performed. These can operate on all records in the input data set (with TRAILER1), or on all records on a page (with TRAILER2), or on all records in a section (with SECTIONS and TRAILER3).
- ▶ SORT with SUM can be used to create totals and count records, and hence averages.
- ▶ IFTHEN WHEN=GROUP can process records as groups.
- ▶ ICETOOL OCCUR prints each unique value for specified numeric and character fields and the number of times it occurs.
- ▶ ICETOOL RANGE prints a message containing the count of values in a numeric range.
- ▶ ICETOOL SELECT selects records for an output data set based on meeting criteria such as duplicates.

- ▶ ICETOOL SPLICE splices together fields from records that have duplicate numeric or character field values. You can use this for situations where you need to join data sets or look up values from another data set.
- ▶ ICETOOL STATS prints messages containing the minimum, maximum, average and the total of all the values for a specific numeric field.
- ▶ ICETOOL UNIQUE prints a message containing the count of unique values in a numeric or character field.

23.3.6 Checking data

Many batch job failures occur because of errors in the input data, for example invalid numeric data (such as when character data is written to a numeric field). ICETOOL can help you avoid these errors as follows:

- ▶ ICETOOL VERIFY can be used to check a data set for errors in signed packed decimal and signed zoned decimal fields. For example, if one of the fields in one of the records has invalid numeric data ICETOOL will terminate with return code 12.

You can print the record number and hexadecimal field value for each record and field in error. (You can specify up to 10 fields to check.)

You can limit the number of errors printed before ICETOOL terminates with the LIMIT parameter.

- ▶ Logical errors and character field problems can be checked with ICETOOL COUNT to set return code of 4 or 12 based on how many records meet error criteria.

Example 23-1 sets return code 4 if any record in the IN data set contains the string ERROR anywhere in positions 1 to 80. Not specifying RC4 will generate a return code 12 instead.

This example is a very simple error condition. More complex error conditions that could be checked for include whether the record as a whole has logical consistency. For example the city being Poughkeepsie but the state being other than New York or Arkansas.

Example 23-1 Using ICETOOL COUNT to detect data errors

```
//ICET1 EXEC PGM=ICETOOL
//DFSMSG DD SYSOUT=*
//TOOLMSG DD SYSOUT=*
//IN DD *
ABCD
DEFG
THIS RECORD IS IN ERROR
/*
//OUT DD SYSOUT=*
//TOOLIN DD *
COUNT FROM(IN) USING(COUN) NOTEMPTY RC4
/*
//COUNCNTL DD *
INCLUDE COND=(1,80,SS,EQ,C'ERROR')
/*
```

These techniques set a non-zero return code. An installation can automate based on such a return code:

- ▶ The same job can behave differently depending on the return code.
- ▶ Automation can detect the return code and cause manual or automatic intervention if it is non-zero.

23.3.7 IFTHEN conditional processing

IFTHEN processing allows different records in a data set to be processed differently, depending on programmer-specified criteria. Further, a series of steps can be performed against the data without intermediate data sets being written.

IFTHEN can be specified on INREC, OUTREC and OUTFIL statements. Indeed a DFSORT step can include IFTHEN processing on as many INREC, OUTREC and OUTFIL statements as you like. For example, you might include IFTHEN on an INREC statement and three OUTFIL statements.

You can specify the following five types of IFTHEN clause:

- ▶ WHEN=INIT, which are processed before all other IFTHEN clauses. The BUILD, OVERLAY, or PARSE subclause is applied to all records.
- ▶ WHEN=(*logexp*), where *logexp* is a logical expression. The BUILD, OVERLAY, or PARSE subclause is applied if the logical expression evaluates to true.
- ▶ WHEN=ANY, where the clause's BUILD, OVERLAY, or PARSE subclause is applied if any of the preceding WHEN=(*logexp*) clauses evaluates to true.
- ▶ WHEN=GROUP allows you to do various types of operations involving groups of records, such as propagating fields, identifiers and sequence numbers within groups. It further facilitates other types of group operations such as sorting by groups, and including or omitting records by groups.

Groups are identified by RECORDS=*n*, BEGIN=(*logexp*), or END=(*logexp*), or any combination of the three.

- ▶ WHEN=NONE, where the BUILD, OVERLAY, or PARSE clause is applied only if none of the preceding WHEN=(*logexp*) expressions are true.

Clauses are processed in the order they're specified, except that WHEN=INIT and WHEN=GROUP clauses are always processed first and WHEN=NONE clauses last.

The following sections provide two simple examples that illustrate the power of IFTHEN.

Treating records differently

Example 23-2 shows a JCL sample.

Example 23-2 Sample JCL

```
//COPY1 EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//SORTIN DD *
AXLE
BOX
AERIAL
DEPOT
ASPIC
CARROT
```



```

/*
//SORTOUT DD SYSOUT=*
//SYSIN DD *
  OPTION COPY
INREC IFTHEN=(WHEN=(1,1,CH,EQ,C'A'),OVERLAY=(8:C'YES')),
  IFTHEN=(WHEN=NONE,OVERLAY=(8:C'NO'))
/*

```

This sample treats records with the character *A* in position 1 differently from those with some other character in the first position. The records with *A* have YES placed in position 8, and the records without *A* have NO placed in position 8. This sample produces the following output:

```

AXLE  YES
BOX   NO
AERIAL YES
DEPOT NO
ASPIC YES
CARROT NO

```

This example, of course, is a very simple IFTHEN example.

Handling groups

Example 23-3 shows a JCL sample.

Example 23-3 Sample JCL

```

//COPY2 EXEC PGM=SORT
//SYSOUT DD SYSOUT=*
//SORTIN DD *
<GROUP>
AXLE
BOX
</GROUP>
<GROUP>
AERIAL
DEPOT
ASPIC
CARROT
</GROUP>
/*
//SORTOUT DD SYSOUT=*
//SYSIN DD *
  OPTION COPY
  INREC IFTHEN=(WHEN=GROUP,
    BEGIN=(1,7,CH,EQ,C'<GROUP>'),
    END=(1,8,CH,EQ,C'</GROUP>'),
    PUSH=(20:SEQ=3,25:ID=3))
  OUTFIL OMIT=(1,7,CH,EQ,C'<GROUP>',OR,
    1,8,CH,EQ,C'</GROUP>'),
    OVERLAY=(20:20,3,ZD,SUB,+1,EDIT=(IIT),25:25,3,ZD,EDIT=(IIT))

```

This example is a more complex example that illustrates the use of WHEN=GROUP with additional post processing. This sample produces the following output:

AXLE	1	1
BOX	2	1
AERIAL	1	2
DEPOT	2	2
ASPIC	3	2
CARROT	4	2

In this output, the first column of numbers is the record number within a group. The second is the group's identifier.

The input data in this example is a pair of groups of records. Each group is bracketed with <GROUP> and </GROUP>.

The WHEN=GROUP clause recognizes the start and end of each group - with the strings specified by BEGIN= and END=.

When a new group is encountered a new record sequence number within the group is placed in positions 20 to 22 (specified by 20:SEQ=3). This sequence number is incremented by 1 for each record in the group.

Specifying 25:ID=3 means the group number (starting at 1) is placed in positions 25 to 27. The group number is incremented by 1 when a new group is encountered.

The OUTFIL OMIT statement removes the group start and end records, and reformats the remaining data records:

- ▶ The sequence number within group is decremented by 1 and reformatted to blank out leading zeroes. (It has to be decremented by 1 because the <GROUP> record had sequence number 1 and the first data record had sequence number 2.
- ▶ The group identifier number is reformatted, suppressing its leading zeroes.

This example, admittedly, is a more complex example. In fact most of the complexity is in the formatting in the OUTFIL statement.

23.4 Symbols

Symbols can make DFSORT and ICETOOL invocations more readable and maintainable. Consider the following example:

```
SORT FIELDS=(12,8,CH,A,4,8,CH,A)
```

This code fragment is difficult to maintain if the input data changes format. This is particularly the case when the same data set could be processed by more than one job.

Now consider the same example rewritten using Symbols. First, here is the SORT invocation again:

```
SORT FIELDS=(SURNAME,A,FIRSTNAME,A)
```

So, DFSORT will understand the SURNAME and FIRSTNAME symbols and define them in a SYMNames DD of the form:

```
//SYMNames DD *
FIRSTNAME,4,8,CH
SURNAME,12,8,CH
/*
```

This code is much more maintainable and the SORT statement is more comprehensible. Rather than using SYMNames DD * for the SYMNames file consider using a permanent data set (such as a PDS member). If more than one DFSORT invocation uses the same input file, a change in file format requires that you change only one part—the SYMNames permanent data set.

Note: Symbols are not attached to the data (as a variable would be). A symbol relates to the same offset and length in the record throughout. So, for example, if you refer to a symbol in a reformatting INREC statement it is likely the reformatting will move the field to a different position in the record and the symbol will *not* adjust its position to match.

A useful technique is to define one set of symbols for each stage, perhaps prefixing each set with additional underscore characters to keep the set unique.

A field can be mapped more than once. Consider the following example:

```
//SYMNames DD *
DATE,1,10,CH
YEAR,=,4,CH
SKIP,1
MONTH,*,2,CH
SKIP,1
DAY,*,2,CH
/*
```

The symbol DATE maps all 10 characters of a date field in the form *YYYY-MM-DD*.

The symbol YEAR maps the *YYYY* portion.

Each SKIP moves the cursor past a “-” portion of the field.

MONTH and DAY map the *MM* and *DD* portions of the field.

The equal sign character (=) in this example says “start the YEAR symbol at the same position as the previous symbol” (in this case DATE).

The star character (*) in the definitions of MONTH and DAY means “start the symbol just after the previous symbol or SKIP.”

23.4.1 Converting COBOL copybooks to DFSORT symbols

The article *Smart DFSORT Tricks* includes a sample REXX EXEC COBDFSYM that allows you to convert a COBOL copybook to a DFSORT Symbols file using the following two-step process:

1. Wrap the COBOL copybook in a dummy program, and compile the program to get a compiled listing.
2. Use the COBDFSYM REXX program to process the compiled listing, and create a DFSORT symbols data set.

This process seems convoluted, but it uses the COBOL compiler to do the hardest work, which makes the REXX EXEC relatively lightweight and maintainable. In reality, installations using this technique can combine the two steps in single a batch job for ease of use. The article includes a sample two-step program that you can adapt.

Example 23-4 shows a sample COBOL copybook.

Example 23-4 Sample COBOL copybook

```
01 PACKAGE-RECORD.
  5 PACKAGE-HEADER.
    10 PACKAGE-HEADER-1 PIC X(13).
    10 FILLER PIC X.
    10 PACKAGE-HEADER-2 PIC X(1).
    10 FILLER PIC X.
    10 PACKAGE-SEQUENCE PIC 9(8) COMP-3.
  5 CUSTOMER-GROUP.
    10 CG-NAME PIC X(3).
    10 CG-COUNT PIC 9(10) COMP-3.
    10 CG-DATE PIC 9(6) COMP.
    10 CG-TIME PIC 9(8) COMP.
    10 CG-TYPE PIC S9(2) COMP.
    10 CG-LIMIT PIC S9(7).
    10 CG-STATUS PIC X(8).
      88 APPROVED VALUE 'APPROVED'.
      88 DENIED VALUE 'DENIED '.
      88 PENDING VALUE SPACES.
    10 CG-COUNTY-NO PIC 99.
      88 DUTCHESS VALUE 14.
      88 KINGS VALUE 24.
      88 NOCOUNTY VALUE ZERO.
```

Example 23-5 shows resulting DFSORT Symbols deck.

Example 23-5 Resulting DFSORT Symbols deck

```
POSITION,1
PACKAGE-RECORD,=,57,CH @1
PACKAGE-HEADER,=,21,CH @1
PACKAGE-HEADER-1,=,13,CH @1
SKIP,1 @14
PACKAGE-HEADER-2,*,1,CH @15
SKIP,1 @16
PACKAGE-SEQUENCE,*,5,PD @17
CUSTOMER-GROUP,*,36,CH @22
CG-NAME,=,3,CH @22
```

```

CG-COUNT,*,6,PD @25
CG-DATE,*,4,BI @31
CG-TIME,*,4,BI @35
CG-TYPE,*,2,FI @39
CG-LIMIT,*,7,ZD @41
CG-STATUS,*,8,CH @48
  APPROVED,'APPROVED'
  DENIED,'DENIED '
  PENDING,' '
CG-COUNTY-NO,*,2,ZD @56
  DUTCHESS,14
  KINGS,24
  NOCOUNTY,0

```

In this example, the @56 and similar comments denote the positions in the record to which the symbol maps.

23.5 Invoking DFSORT from Java with JZOS

JZOS includes the DfSort class, which enables programmers to invoke DFSORT from Java in a straightforward manner. Java is an ideal language for scripting DFSORT because:

- ▶ Sophisticated operations can be performed inside a single program.
- ▶ Encapsulating these sophisticated operations inside a Java class makes them reusable.
- ▶ This approach retains the speed and linguistic simplicity of DFSORT.
- ▶ DFSORT is driven by text statements and Java has very strong text processing capabilities.
- ▶ JZOS allows records from arbitrary sources to be sorted by DFSORT and written to arbitrary destinations, not just standard data sets.

23.5.1 JZOS invoking DFSORT sample

Consider a case where we have records with 4-character names. We want to find the most popular names in the records, which is termed *ranking by frequency*. We use the following sample input data:

```

ABCD
EFGH
ABCD
ABCD
EFGH
IJKL

```

The desired output is:

```

ABCD    3
EFGH    2
IJKL    1

```

To produce this output, we need to perform the following DFSORT operations:

1. Count the records for each value in the field, and produce one record for each field value with a count in it.

2. Sort the *count* records by count, and reformat the output to produce the report.

The output of the first operation is the input to the second, so we pass it by a transient data set. This data set contains records each with two fields:

- ▶ The 4-character (EBCDIC) string
- ▶ A 4-byte integer count (binary)

Example 23-6 shows the JZOS DfSort class used to perform two DFSORT invocations within the same program.

Example 23-6 Sample JZOS program to drive DFSORT: Ranking by frequency

```
// This program performs a "rank" operation
//
// In this case it counts the number of records with a give value in
// columns 1 to 4 and sorts by count descending.
//
// It invokes DFSORT twice to do it.
//
import com.ibm.jzos.DfSort;
import java.util.Random;

class SortRank
{
    public static void main(String args[])
    {

        // Generate a transient data set name based on userid and a
        // random integer rendered as a 7-digit number.
        Random generator=new Random();
        int r=generator.nextInt(9999999);
        String transDS="TEMPFILE.T"+Integer.toString(r);

        // Invoke DFSORT once - to create counts for each value
        DfSort dfSort1 = new DfSort();
        dfSort1.addControlStatement("INREC OVERLAY=(5:X'00000001')");
        dfSort1.addControlStatement("SORT FIELDS=(1,4,CH,A)");
        dfSort1.addControlStatement("SUM FIELDS=(5,4,BI)");
        dfSort1.addAllocation("alloc fi(SORTIN) da('MPACKER.INPUT1') reuse"+
            " shr msg(2)");
        dfSort1.addAllocation("alloc fi(SORTOUT)" +
            " da("+transDS+") reuse catalog"+
            " new cyl blksize(0) lrecl(80) recfm(fb) space(1,1) msg(2)");
        dfSort1.setSameAddressSpace(true);
        dfSort1.execute();

        dfSort1.addAllocation("free fi(SORTOUT)");

        // Invoke DFSORT a second time - to sort based on counts
        // NOTE: Transient data set is deleted at end of this sort.
        DfSort dfSort2 = new DfSort();
        dfSort2.addAllocation("alloc fi(SORTIN) da("+transDS+") reuse"+
            " delete shr msg(2)");
        dfSort2.addAllocation("alloc fi(SORTOUT)" +
            " da('MPACKER.OUTPUT1') reuse catalog"+
            " new cyl blksize(0) lrecl(80) recfm(fb) space(1,1) msg(2)");
```

```

dfSort2.addControlStatement("SORT FIELDS=(5,4,BI,D)");
dfSort2.addControlStatement("OUTREC "+
"FIELDS=(1,4,X,5,4,BI,EDIT=(IIIT))");
dfSort2.setSameAddressSpace(true);
dfSort2.execute();
}
}

```

Here are the steps involved in this program:

1. Generate a transient data set name.

In this case, we generate an unqualified data set name using a random number. (It would best to create a temporary data set, but JZOS, which uses the BPXWDYN service, is unable to create temporary data sets.)

2. Perform the first DFSORT invocation.

We create object `dfSort1` of class `DfSort` to control the sort.

We add control statements using the `addControlStatement()` method and allocate SORTIN and SORTOUT data sets using the `addAllocation()` method.

We ensure that DFSORT runs in the same address space, and we run DFSORT using the `execute()` method.

Note: The DFSORT statements in the first invocation are as follows:

```

INREC OVERLAY=(5:X'00000001') which places a binary '1' in positions 5 to 8
SORT FIELDS=(1,4,CH,A) which sorts on the character field we're counting
SUM FIELDS=(5,4,BI) which counts by summing the '1' values from INREC

```

3. Free the SORTOUT data set so we can reallocate it to the SORTIN DD in the second DFSORT invocation.
4. Invoke DFSORT a second time.

This time, the SORTIN is the output of the first sort and the SORTOUT is our final report.

Note: The DFSORT statements in the second invocation are as follows:

```

SORT FIELDS=(5,4,BI,D) which sorts on the frequency field
OUTREC FIELDS=(1,4,X,5,4,BI,EDIT=(IIIT)) which formats the output

```

This example is fairly complex, but it does include two DFSORT invocations and lots of data set management. To be fair, the JCL is simpler because there are no DD statements for the data or the DFSORT control statements.

In both DFSORT invocations the program was instructed to run in the same address space. This simplifies retrieving the messages DFSORT produces.

You can retrieve the DFSORT return code with the `getReturnCode()` method. In our example, we did not do so.

Further integration between DFSORT and JZOS is possible. You can pass the `getChildStdinStream()` and `getChildStdoutStream()` methods records to and from DFSORT. These methods connect DFSORT to arbitrary sources of data and destinations, using the standard Java `BufferedInputStream` and `BufferedOutputStream` classes.

Note: It is not possible to use a `getChildStdoutStream()` as an OUTFIL destination. Further, because `getChildStdinStream()` and `getChildStdoutStream()` use E15 and E35 exits, respectively it is not possible to use these Assembler exits when the streams are used. It is, however, possible to use these exits in a JZOS DFSORT step when the streams are not used.



Part 6

Appendixes



DB2 configuration

In this appendix, we describe the DB2 configuration for the samples using DB2. This appendix includes the following topics:

- ▶ Data Definition Language for Java stored procedure
- ▶ Data Definition Language for stand-alone Java application
- ▶ DB2 SQL Insert statements for sample data

Data Definition Language for Java stored procedure

The following examples show the DDL that we use for the Java Stored procedure described in 6.3, “Java in DB2 for z/OS” on page 64.

Example A-1 DDL for table BATCH.XML2PDF

```
CREATE TABLE BATCH.XML2PDF
  (ID          INTEGER      NOT NULL
  ,XML_DATA    XML
  ,PDF_DATA    BLOB(1M)
  ,PRIMARY KEY (ID)
  )
;
GRANT SELECT, INSERT, UPDATE, DELETE
ON TABLE BATCH.XML2PDF TO PUBLIC;
```

Example A-2 DDL for Java Stored procedure

```
CREATE PROCEDURE BATCH.JAVASTP
  (IN ACTION  VARCHAR(15)
  ,IN PDF_DIR VARCHAR(100))
EXTERNAL NAME 'com.ibm.itso.sample.GenPdf.runGenerate'
LANGUAGE JAVA
PARAMETER STYLE JAVA
NOT DETERMINISTIC
FENCED
CALLED ON NULL INPUT
MODIFIES SQL DATA
NO DBINFO
NO COLLID
WLM ENVIRONMENT D9GGWLMJ
ASUTIME NO LIMIT
STAY RESIDENT YES
PROGRAM TYPE SUB
SECURITY DB2
INHERIT SPECIAL REGISTERS
STOP AFTER SYSTEM DEFAULT FAILURES
COMMIT ON RETURN NO ;

GRANT EXECUTE ON PROCEDURE BATCH.JAVASTP TO PUBLIC;
```

Data Definition Language for stand-alone Java application

Example A-3 is the DDL that we use for the sample application described in 7.5, “Sample stand-alone Java batch application” on page 82.

Example A-3 DDL for Java stand-alone application

```
SET CURRENT SCHEMA = 'BATCH';

-- CREATE TABLE CUSTOMER
CREATE TABLE CUSTOMER
  (CUST_NO      INTEGER      NOT NULL
```

```

, FIRST_NAME  VARCHAR(20) NOT NULL
, LAST_NAME   VARCHAR(30) NOT NULL
, STREET_NAME VARCHAR(30) NOT NULL
, STREET_NO   SMALLINT   NOT NULL
, ZIP         SMALLINT   NOT NULL
, CITY        VARCHAR(20) NOT NULL
, STATE       CHAR(2)     NOT NULL
, COUNTRY     VARCHAR(20) NOT NULL
, PRIMARY KEY (CUST_NO)
)
;

-- CREATE TABLE ITEM
CREATE TABLE ITEMS
(ITEM_NO      INTEGER      NOT NULL
, DESCRIPTION VARCHAR(100) NOT NULL
, SINGLE_PRICE DECIMAL(10, 2) NOT NULL
, PRIMARY KEY (ITEM_NO)
)
;

-- CREATE TABLE ORDER
CREATE TABLE ORDER
(ORDER_NO     INTEGER      NOT NULL
, POS_NO      SMALLINT    NOT NULL
, CUST_NO     INTEGER      NOT NULL
, ITEM_NO     INTEGER      NOT NULL
, QUANTITY    SMALLINT    NOT NULL
, PRIMARY KEY (ORDER_NO, POS_NO)
)
;

-- ADD FOREIGN KEYS
ALTER TABLE ORDER
  ADD CONSTRAINT FK_ORDER_CUST
    FOREIGN KEY (CUST_NO)
    REFERENCES CUSTOMER
    ON DELETE CASCADE
;
ALTER TABLE ORDER
  ADD CONSTRAINT FK_ORDER_ITEM
    FOREIGN KEY (ITEM_NO)
    REFERENCES ITEMS
    ON DELETE RESTRICT
;

-- GRANTS
GRANT SELECT, INSERT, UPDATE, DELETE
ON TABLE CUSTOMER
, ITEMS
, ORDER
TO PUBLIC
;

```

DB2 SQL Insert statements for sample data

To work with the DB2 tables defined before, we execute the SQL statements shown in the following examples to insert some sample data.

Example A-4 SQL statements to insert sample data for Java Stored procedure

```
-- Insert first company
INSERT INTO BATCH.XML2PDF (ID, XML_DATA)
VALUES (1,
'<?xml version="1.0"?>
<Company>
  <Name>Big data center</Name>
  <Department>
    <DepNo>1</DepNo>
    <Description>Production</Description>
    <Employee>
      <EmpNo>1</EmpNo>
      <FirstName>Paul</FirstName>
      <LastName>Smith</LastName>
    </Employee>
    <Employee>
      <EmpNo>2</EmpNo>
      <FirstName>Hannah</FirstName>
      <LastName>Smith</LastName>
    </Employee>
  </Department>
  <Department>
    <DepNo>2</DepNo>
    <Description>Development</Description>
    <Employee>
      <EmpNo>3</EmpNo>
      <FirstName>Mia</FirstName>
      <LastName>Brown</LastName>
    </Employee>
  </Department>
</Company>
')
;
-- Insert second company
INSERT INTO BATCH.XML2PDF (ID, XML_DATA)
VALUES (2,
'<?xml version="1.0"?>
<Company>
  <Name>Small data center</Name>
  <Department>
    <DepNo>11</DepNo>
    <Description>Production</Description>
    <Employee>
      <EmpNo>1</EmpNo>
      <FirstName>Henry</FirstName>
      <LastName>Tramp</LastName>
    </Employee>
  </Department>
<Department>
```

```

    <DepNo>22</DepNo>
    <Description>Development</Description>
    <Employee>
      <EmpNo>2</EmpNo>
      <FirstName>Mike</FirstName>
      <LastName>Rocket</LastName>
    </Employee>
  </Department>
</Company>
')
;

```

Example A-5 SQL statements to insert sample data for stand-alone Java application

```

-- SET CURRENT SCHEMA
SET CURRENT SCHEMA = 'BATCH';

-- INSERT CUSTOMER
INSERT INTO CUSTOMER
  (CUST_NO, FIRST_NAME, LAST_NAME, STREET_NAME, STREET_NO
  ,ZIP, CITY, STATE, COUNTRY)
VALUES (1, 'Bob', 'Smith', 'Bestplace', 12,
        12345, 'New York City', 'NY', 'USA');

INSERT INTO CUSTOMER
  (CUST_NO, FIRST_NAME, LAST_NAME, STREET_NAME, STREET_NO
  ,ZIP, CITY, STATE, COUNTRY)
VALUES (2, 'Mary', 'Smith', 'Bestplace', 12,
        12345, 'New York City', 'NY', 'USA');

INSERT INTO CUSTOMER
  (CUST_NO, FIRST_NAME, LAST_NAME, STREET_NAME, STREET_NO
  ,ZIP, CITY, STATE, COUNTRY)
VALUES (3, 'John', 'Fellow', 'Nicestreet', 345,
        23456, 'Boston', 'MA', 'USA');

INSERT INTO CUSTOMER
  (CUST_NO, FIRST_NAME, LAST_NAME, STREET_NAME, STREET_NO
  ,ZIP, CITY, STATE, COUNTRY)
VALUES (4, 'Bart', 'Brown', 'Route', 66,
        10815, 'Cubero', 'NM', 'USA');

-- INSERT ITEMS
INSERT INTO ITEMS
  (ITEM_NO, SINGLE_PRICE, DESCRIPTION)
VALUES (1, 82.50, 'Sandwich Maker');

INSERT INTO ITEMS
  (ITEM_NO, SINGLE_PRICE, DESCRIPTION)
VALUES (2, 35.00, 'Indoor Grill');

INSERT INTO ITEMS
  (ITEM_NO, SINGLE_PRICE, DESCRIPTION)
VALUES (3, 632.30, 'Espresso Machine');

```

```
INSERT INTO ITEMS
  (ITEM_NO, SINGLE_PRICE, DESCRIPTION)
VALUES (4, 52.40, 'Hand Mixer');
```

```
INSERT INTO ITEMS
  (ITEM_NO, SINGLE_PRICE, DESCRIPTION)
VALUES (5, 72.30, 'Ice Crusher');
```

```
INSERT INTO ITEMS
  (ITEM_NO, SINGLE_PRICE, DESCRIPTION)
VALUES (6, 18.00, '12-Cup Coffee Maker');
```

```
INSERT INTO ITEMS
  (ITEM_NO, SINGLE_PRICE, DESCRIPTION)
VALUES (7, 138.00, '950 Watts Microwave Oven');
```



B

Source code

This appendix contains source code to which we refer in the chapters of this book. You can find all source code for this book in the additional materials. For information about downloading the additional materials, refer to Appendix C, “Additional material” on page 453.

Java stored procedure to generate PDF files

In this section, we provide the following source codes.

- ▶ Java Stored procedure that queries XML data from a DB2 z/OS database and creates PDF files
- ▶ Content of the files you need to deploy the Stored procedure
- ▶ Java application that refresh the Stored procedure application environment
- ▶ Java application to test the Stored procedure

Example B-1 GenPdf.java - Java Stored procedure

```
package com.ibm.itso.sample;

import java.sql.SQLException;

public class GenPdf {

    public static void runGenerate(String action, String pdfDir)
        throws SQLException {
        PdfCreator pdfc = new PdfCreator();
        pdfc.generatePDF(action, pdfDir);
    }
}
```

Example B-2 PdfCreator.java - Java Stored procedure

```
package com.ibm.itso.sample;

import java.awt.Color;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import com.lowagie.text.Chunk;
import com.lowagie.text.Document;
import com.lowagie.text.DocumentException;
import com.lowagie.text.Element;
import com.lowagie.text.Font;
import com.lowagie.text.FontFactory;
import com.lowagie.text.Paragraph;
import com.lowagie.text.pdf.PdfWriter;
import com.lowagie.text.pdf.draw.LineSeparator;
import com.lowagie.text.pdf.draw.VerticalPositionMark;

public class PdfCreator {
```

```

public void generatePDF(String action, String pdfDir)
    throws SQLException {
    try {
        // check value of parameter "action"
        if (!action.equalsIgnoreCase("store_fs")
            && !action.equalsIgnoreCase("store_fs_db2")
            && !action.equalsIgnoreCase("store_db2")) {
            throw new SQLException("wrong value for parameter 'action'");
        }

        // Get connection to the database
        Connection con = DriverManager.getConnection("jdbc:default:connection");

        // Select company data from XML column
        String strSQL = "SELECT ID, EMP.* FROM BATCH.XML2PDF, XMLTABLE("
            + "'$X/Company/Department/Employee' PASSING XML_DATA AS \"X\" "
            + "COLUMNS COMPANY    VARCHAR(50) PATH '.././Name' "
            + "    , DEPT_NO    INTEGER    PATH '../DepNo' "
            + "    , DEPT_DESCR VARCHAR(20) PATH '../Description' "
            + "    , EMP_NO    INTEGER    PATH 'EmpNo' "
            + "    , FIRST_NAME VARCHAR(20) PATH 'FirstName' "
            + "    , LAST_NAME  VARCHAR(20) PATH 'LastName' "
            + ") EMP ORDER BY ID, EMP.DEPT_NO, EMP_NO";

        // Start generating PDF document(s)
        PreparedStatement ps = con.prepareStatement(strSQL);
        ResultSet rs = ps.executeQuery();

        Document document = null;
        Chunk c = null;
        VerticalPositionMark separator = new LineSeparator(1, 100,
            Color.BLUE, Element.ALIGN_RIGHT, -2);
        ByteArrayOutputStream out = null;
        boolean printHeading = true;
        int oldDeptNo = 0;
        int curId = 0;
        int oldId = 0;
        while (rs.next()) {
            curId = rs.getInt("ID");
            if (curId != oldId) {
                // Company has changed
                if (oldId != 0) {
                    // save PDF in DB2 and/or File system
                    document.close();
                    this.savePdf(con, action, pdfDir, oldId, out);
                }

                // generate new PDF file in memory for next company
                document = new Document();
                out = new ByteArrayOutputStream();
                PdfWriter.getInstance(document, out);

                document.open();
                printHeading = true;
                oldDeptNo = 0;
            }
        }
    }
}

```

```

        oldId = curId;
    }

    String CompanyName = rs.getString("COMPANY");
    int curDeptNo = rs.getInt("DEPT_NO");
    String DeptDesc = rs.getString("DEPT_DESCR");
    int EmpNo = rs.getInt("EMP_NO");
    String EmpFirstName = rs.getString("FIRST_NAME");
    String EmpLastName = rs.getString("LAST_NAME");

    // Heading
    if (printHeading) {
        document.add(separator);
        Paragraph heading = new Paragraph();
        heading.setAlignment(Element.ALIGN_LEFT);
        heading.setSpacingAfter(3);
        c = new Chunk("All Employees for company '" + CompanyName
            + "'\n", FontFactory.getFont(FontFactory.HELVETICA,
            12, Font.NORMAL, new Color(0, 0, 0)));
        heading.add(new Chunk(c));
        document.add(heading);
        document.add(separator);

        printHeading = false;
    }

    // print Employees
    if (curDeptNo != oldDeptNo) {
        // new Department
        Paragraph department = new Paragraph();
        department.setAlignment(Element.ALIGN_LEFT);
        department.setSpacingBefore(15);
        c = new Chunk("Department " + curDeptNo + ": " + DeptDesc
            + "\n", FontFactory.getFont(FontFactory.HELVETICA,
            12, Font.NORMAL, new Color(0, 0, 0)));
        department.add(new Chunk(c));
        document.add(department);

        oldDeptNo = curDeptNo;
    }

    Paragraph employee = new Paragraph();
    employee.setAlignment(Element.ALIGN_LEFT);
    employee.setIndentationLeft(30);
    c = new Chunk("EmpNo: " + EmpNo + ", " + EmpFirstName + " "
        + EmpLastName + "\n", FontFactory.getFont(
        FontFactory.HELVETICA, 12, Font.NORMAL, new Color(0, 0,
        0)));
    employee.add(new Chunk(c));
    document.add(employee);
}
rs.close();
ps.close();

// save last PDF in DB2 and/or File system

```

```

        document.close();
        this.savePdf(con, action, pdfDir, oldId, out);
    } catch (DocumentException de) {
        throw new SQLException("DocumentException: " + de.getMessage());
    } catch (IOException ioe) {
        throw new SQLException("IOException: " + ioe.getMessage());
    }
}

private void savePdf(Connection con, String action, String pdfDir,
    int oldId, ByteArrayOutputStream out) throws IOException, SQLException {
    // handle PDF file depending on parameter 'action'
    if (action.equalsIgnoreCase("store_fs")
        || action.equalsIgnoreCase("store_fs_db2")) {
        // write PDF to file system
        FileOutputStream fOut = new FileOutputStream(pdfDir
            + File.separator + "Report" + oldId + ".pdf");
        out.writeTo(fOut);
        fOut.close();
    }

    if (action.equalsIgnoreCase("store_db2")
        || action.equalsIgnoreCase("store_fs_db2")) {
        // update current DB2 row and save PDF file as BLOB
        String strSQL = "UPDATE BATCH.XML2PDF SET PDF_DATA = ? WHERE ID = ?";
        PreparedStatement psUpd = con.prepareStatement(strSQL);
        InputStream is = new ByteArrayInputStream(out.toByteArray());
        psUpd.setBinaryStream(1, is, -1);
        psUpd.setInt(2, oldId);
        psUpd.execute();
        psUpd.close();
    }

    // Close the document
    out.close();
}
}
}

```

Example B-3 deploy.xml

```

<project name="deploy" default="all">
  <!-- This ant script deploys a batch Java applications to z/OS
    using ftp.

    To run this script, YOU need to customize a zos.properties file
    in one of the following places (in order of highest precedence):
    - the user's home directory ("Documents and Settings\<name>" in Windows)
    - the eclipse workspace base directory (the parent of this directory)
    - the same directory as this script

    See the sample zos.properties files for required values
  -->
  <property file="${user.home}/zos.properties"/>
  <property file="./zos.properties" />
  <property file="./zos.properties" />

```

```

<target name="all" depends="compile, deployJar" />

<!-- for compiling if not done automatically by IDE (Eclipse) -->
<target name="compile">
  <mkdir dir="bin"/>
  <javac destdir="bin"
    debug="on">
    <src path="src"/>
  </javac>
</target>

<target name="buildJar">
  <mkdir dir="deploy"/>
  <jar destfile="deploy/${jarname}">
    <fileset dir="bin">
      <include name="**/*"/>
    </fileset>
  </jar>
</target>

<target name="deployJar" depends="buildJar" >
  <echo message="Copying files to ${server}:${appl.home}..."/>
  <ftp server="${server}"
    userid="${userid}"
    password="${password}"
    remotedir="${appl.home}"
    depends="no"
    binary="yes"
    verbose="yes" >
    <fileset dir="deploy" includes="" casesensitive="yes" />
  </ftp>
</target>
</project>

```

Example B-4 zos.properties

```

# Customize this file or move a customized copy of this file up to
# either the Eclipse workspace directory or your "home directory"
server = zos.server.dns.name
userid = uid
password = pw

# the home directory for deploying the application jar
appl.home = /u/wagnera/javastp

# jar name
jarname = PdfGenerate.jar

# time to wait for output in seconds
waittime = 0

# remote debugging
debug= no

```

Example B-5 RefreshWLM.java

```
package com.ibm.itso.sample;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Types;

public class RefreshWLM {

    /**
     * Start to refresh Java Stored Procedure after changing Code
     *
     * @throws Exception
     */
    public static void main(String[] args) throws Exception {
        try {
            if (args.length != 3) {
                System.err.println("Input parameter is missing!");
                System.err.println
                ("Para1: JDBC URL (e.g. 'jdbc:db2://<server>:<port>/<DB2Location>')");
                System.err.println("Para2: DB User");
                System.err.println("Para3: DB Password");

                throw new Exception("Input parameter is missing!!!!");
            }

            String URL = args[0];
            String User = args[1];
            String Pw = args[2];

            // connect to DB2
            Class.forName("com.ibm.db2.jcc.DB2Driver");
            System.out.println("Connection.....");
            Connection con = DriverManager.getConnection(URL, User, Pw);
            System.out.println("connected.");

            System.out.println("Call STP.....");
            CallableStatement cs = con
                .prepareCall("CALL SYSPROC.WLM_REFRESH(?, ?, ?, ?)");
            cs.setString(1, "D9GGWLMJ");
            cs.setString(2, "D9G1");
            cs.registerOutParameter(3, Types.VARCHAR);
            cs.registerOutParameter(4, Types.INTEGER);
            cs.execute();
            System.out.println("WLM_REFRESH ended with: RC=" + cs.getInt(4)
                + ", Text: " + cs.getString(3));
            cs.close();

            con.close();
            System.out.println("Connection closed.");
        } catch (ClassNotFoundException cnfe) {
            System.err.println(cnfe.getMessage());
            cnfe.printStackTrace();
        }
    }
}
```

```

    } catch (SQLException sqle) {
        System.err.println(sqle.getMessage());
        sqle.printStackTrace();
    }
}
}
}

```

Example B-6 TestSTP.java

```

package com.ibm.itso.sample;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class TestSTP {

    /**
     * Start to test Java Stored Procedure
     */
    public static void main(String[] args) throws Exception {

        try {
            if (args.length != 3) {
                System.err.println("Input parameter is missing!");
                System.err.println
                    ("Para1: JDBC URL (e.g. 'jdbc:db2://<server>:<port>/<DB2Location>')");
                System.err.println("Para2: DB User");
                System.err.println("Para3: DB Password");

                throw new Exception("Input parameter is missing!!!!");
            }

            String URL = args[0];
            String User = args[1];
            String Pw = args[2];

            // connect to DB2
            Class.forName("com.ibm.db2.jcc.DB2Driver");
            System.out.println("Connection.....");
            Connection con = DriverManager.getConnection(URL, User, Pw);
            System.out.println("connected.");

            System.out.println("Call STP.....");
            CallableStatement cs = con.prepareCall("CALL BATCH.JAVASTP(?, ?)");
            cs.setString(1, "store_fs_db2");
            cs.setString(2, "/u/wagnera/javastp/pdfs");
            cs.execute();
            cs.close();
            System.out.println("Call STP ended.");

            con.close();
            System.out.println("Connection closed.");
        } catch (ClassNotFoundException cnfe) {

```



```

        System.err.println(cnfe.getMessage());
        cnfe.printStackTrace();
    } catch (SQLException sqle) {
        System.err.println(sqle.getMessage());
        sqle.printStackTrace();
    }
}
}
}

```

Java PDF creator

In this section, we provide the source code for the Java application that queries a DB2 z/OS database and creates a PDF invoice based on the DB2 data.

Example B-7 Invoice Creator.java

```

package com.ibm.itso.sample;

public class InvoiceCreator {

    public static void main(String[] args) {
        PdfCreator pdfc = new PdfCreator();
        pdfc.createInvoice(args[0]);
    }
}

```

Example B-8 PdfCreator.java

```

package com.ibm.itso.sample;

import java.awt.Color;
import java.io.FileOutputStream;
import java.io.IOException;
import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;

import com.lowagie.text.Chunk;
import com.lowagie.text.Document;
import com.lowagie.text.DocumentException;
import com.lowagie.text.Element;
import com.lowagie.text.Font;
import com.lowagie.text.FontFactory;
import com.lowagie.text.Paragraph;
import com.lowagie.text.pdf.PdfPCe11;
import com.lowagie.text.pdf.PdfPTable;
import com.lowagie.text.pdf.PdfWriter;
import com.lowagie.text.pdf.draw.LineSeparator;
import com.lowagie.text.pdf.draw.VerticalPositionMark;

```

```

public class PdfCreator {
    PdfCreator()
    {

    }

    public void createInvoice(String pdfDir)
    {
        try {
            //Connect to DB2 and retrieve all orders
            System.out.println("Connecting to DB2");

            Class.forName("com.ibm.db2.jcc.DB2Driver");
            Connection con=DriverManager.getConnection("jdbc:db2:DB9G");
            System.out.println("... connected.");

            ArrayList orderNos = new ArrayList();

            String strSQL = "SELECT DISTINCT Order_No FROM BATCH.ORDER";
            PreparedStatement ps = con.prepareStatement(strSQL);
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                int orderNo=rs.getInt(1);
                orderNos.add(orderNo);
            }
            rs.close();
            ps.close();

            //Create PDF invoices for each order number
            for (int i=0; i<orderNos.size(); i++)
            {
                int currentOrderNo=(Integer)orderNos.get(i);
                System.out.println("Starting to create pdf file
                "+pdfDir+"/Invoice_No_"+String.valueOf(currentOrderNo)+".pdf...");

                FileOutputStream out = new
                FileOutputStream(pdfDir+"/Invoice_No_"+String.valueOf(currentOrderNo)+".pdf");
                Document document = new Document();
                PdfWriter.getInstance(document, out);
                document.open();
                Chunk c;

                // Add sender information
                Paragraph sender= new Paragraph();
                sender.setAlignment(Element.ALIGN_RIGHT);
                c=new Chunk("Bob's Batch Store\n",
                FontFactory.getFont(FontFactory.HELVETICA, 12, Font.NORMAL, new Color(0, 0, 0)));
                sender.add(new Chunk(c));
                c=new Chunk("JES Street 1111\n",
                FontFactory.getFont(FontFactory.HELVETICA, 12, Font.NORMAL, new Color(0, 0, 0)));
                sender.add(new Chunk(c));
                c=new Chunk("12601 Poughkeepsie, NY\n",
                FontFactory.getFont(FontFactory.HELVETICA, 12, Font.NORMAL, new Color(0, 0, 0)));
                sender.add(new Chunk(c));
            }
        }
    }
}

```

```

        c=new Chunk("USA\n", FontFactory.getFont(FontFactory.HELVETICA, 12,
Font.NORMAL, new Color(0, 0, 0)));
        sender.add(new Chunk(c));
        document.add(sender);

        //Add recipient information from database
        strSQL = "SELECT DISTINCT BATCH.CUSTOMER.* FROM BATCH.CUSTOMER,
BATCH.ORDER WHERE BATCH.CUSTOMER.CUST_NO = BATCH.ORDER.CUST_NO AND
BATCH.ORDER.ORDER_NO = "+ String.valueOf(currentOrderNo);
        ps = con.prepareStatement(strSQL);
        rs = ps.executeQuery();
        while (rs.next()) {
            String name=rs.getString("FIRST_NAME")+
"+rs.getString("LAST_NAME");
            String street=rs.getString("STREET_NAME")+
"+rs.getInt("STREET_NO");
            String town=rs.getInt("ZIP")+ " "+rs.getString("CITY")+
"+rs.getString("STATE");
            String country=rs.getString("COUNTRY");

            Paragraph recipient= new Paragraph();
            recipient.setAlignment(Element.ALIGN_LEFT);
            c=new Chunk(name+"\n", FontFactory.getFont(FontFactory.HELVETICA,
12, Font.NORMAL, new Color(0, 0, 0)));
            recipient.add(new Chunk(c));
            c=new Chunk(street+"\n", FontFactory.getFont(FontFactory.HELVETICA,
12, Font.NORMAL, new Color(0, 0, 0)));
            recipient.add(new Chunk(c));
            c=new Chunk(town+"\n", FontFactory.getFont(FontFactory.HELVETICA,
12, Font.NORMAL, new Color(0, 0, 0)));
            recipient.add(new Chunk(c));
            c=new Chunk(country+"\n",
FontFactory.getFont(FontFactory.HELVETICA, 12, Font.NORMAL, new Color(0, 0, 0)));
            recipient.add(new Chunk(c));
            recipient.add("\n");
            document.add(recipient);
        }
        rs.close();
        ps.close();

        //Add vertical line
        VerticalPositionMark separator = new LineSeparator(1, 100, Color.BLUE,
Element.ALIGN_RIGHT, -2);
        document.add(separator);
        document.add(new Chunk("\n"));

        //Add table header data
        float[] widths = {0.08f, 0.15f, 0.57f, 0.20f, 0.20f};
        PdfPTable table = new PdfPTable(widths);
        table.setWidthPercentage(100);
        PdfPCell cell = new PdfPCell();

        c=new Chunk("Pos", FontFactory.getFont(FontFactory.HELVETICA, 12,
Font.BOLD, new Color(0, 0, 0)));
        cell = new PdfPCell(new Paragraph(c));

```

```

        cell.setMinimumHeight(25);
        cell.setHorizontalAlignment(Element.ALIGN_CENTER);
        cell.setVerticalAlignment(Element.ALIGN_MIDDLE);
        table.addCell(cell);
        c=new Chunk("Quantity", FontFactory.getFont(FontFactory.HELVETICA, 12,
Font.BOLD, new Color(0, 0, 0)));
        cell = new PdfPCell(new Paragraph(c));
        cell.setHorizontalAlignment(Element.ALIGN_CENTER);
        cell.setVerticalAlignment(Element.ALIGN_MIDDLE);
        table.addCell(cell);
        c=new Chunk("Description", FontFactory.getFont(FontFactory.HELVETICA,
12, Font.BOLD, new Color(0, 0, 0)));
        cell = new PdfPCell(new Paragraph(c));
        cell.setHorizontalAlignment(Element.ALIGN_CENTER);
        cell.setVerticalAlignment(Element.ALIGN_MIDDLE);
        table.addCell(cell);
        c=new Chunk("Single price", FontFactory.getFont(FontFactory.HELVETICA,
12, Font.BOLD, new Color(0, 0, 0)));
        cell = new PdfPCell(new Paragraph(c));
        cell.setHorizontalAlignment(Element.ALIGN_CENTER);
        cell.setVerticalAlignment(Element.ALIGN_MIDDLE);
        table.addCell(cell);
        c=new Chunk("Total price", FontFactory.getFont(FontFactory.HELVETICA,
12, Font.BOLD, new Color(0, 0, 0)));
        cell = new PdfPCell(new Paragraph(c));
        cell.setHorizontalAlignment(Element.ALIGN_CENTER);
        cell.setVerticalAlignment(Element.ALIGN_MIDDLE);
        table.addCell(cell);

        BigDecimal total=new BigDecimal(0);

        //Add order data from database
        strSQL = "SELECT * FROM BATCH.ITEMS, BATCH.ORDER WHERE
BATCH.ITEMS.ITEM_NO = BATCH.ORDER.ITEM_NO AND BATCH.ORDER.ORDER_NO = "+
String.valueOf(currentOrderNo)+" ORDER BY BATCH.ORDER.POS_NO";
        ps = con.prepareStatement(strSQL);
        rs = ps.executeQuery();
        while (rs.next()) {
            int pos=rs.getInt("POS_NO");
            int quantity=rs.getInt("QUANTITY");
            String description=rs.getString("DESCRIPTION");
            BigDecimal singlePrice=rs.getBigDecimal("SINGLE_PRICE");
            BigDecimal totalPrice=new BigDecimal("0");

            c=new Chunk(String.valueOf(pos),
FontFactory.getFont(FontFactory.HELVETICA, 12, Font.NORMAL, new Color(0, 0, 0)));
            cell = new PdfPCell(new Paragraph(c));
            table.addCell(cell);
            c=new Chunk(String.valueOf(quantity),
FontFactory.getFont(FontFactory.HELVETICA, 12, Font.NORMAL, new Color(0, 0, 0)));
            cell = new PdfPCell(new Paragraph(c));
            table.addCell(cell);
            c=new Chunk(description, FontFactory.getFont(FontFactory.HELVETICA,
12, Font.NORMAL, new Color(0, 0, 0)));
            cell = new PdfPCell(new Paragraph(c));

```

```

        table.addCell(cell);
        c=new Chunk(singlePrice.toString()+" USD",
FontFactory.getFont(FontFactory.HELVETICA, 12, Font.NORMAL, new Color(0, 0, 0)));
        cell = new PdfPCell(new Paragraph(c));
        cell.setHorizontalAlignment(Element.ALIGN_RIGHT);
        table.addCell(cell);
        totalPrice=singlePrice.multiply(new BigDecimal(quantity));
        c=new Chunk(totalPrice.toString()+" USD",
FontFactory.getFont(FontFactory.HELVETICA, 12, Font.NORMAL, new Color(0, 0, 0)));
        cell = new PdfPCell(new Paragraph(c));
        cell.setHorizontalAlignment(Element.ALIGN_RIGHT);
        table.addCell(cell);
        total=total.add(totalPrice);
    }
    rs.close();
    ps.close();

    //Add table footer data
    c=new Chunk("Total", FontFactory.getFont(FontFactory.HELVETICA, 12,
Font.BOLD, new Color(0, 0, 0)));
    cell = new PdfPCell(new Paragraph(c));
    cell.setVerticalAlignment(Element.ALIGN_MIDDLE);
    cell.setMinimumHeight(25);
    table.addCell(cell);
    cell = new PdfPCell(new Paragraph(""));
    cell.setColspan(3);
    table.addCell(cell);
    c=new Chunk(total.toString()+" USD",
FontFactory.getFont(FontFactory.HELVETICA, 12, Font.BOLD, new Color(0, 0, 0)));
    cell = new PdfPCell(new Paragraph(c));
    cell.setHorizontalAlignment(Element.ALIGN_RIGHT);
    cell.setVerticalAlignment(Element.ALIGN_MIDDLE);
    table.addCell(cell);
    document.add(table);

    //Add vertical line
    document.add(new Chunk("\n"));
    document.add(new Chunk("\n"));
    document.add(separator);

    c=new Chunk("Please read our terms and conditions.",
FontFactory.getFont(FontFactory.HELVETICA, 10, Font.ITALIC, new Color(0, 0, 0)));
    document.add(new Paragraph (c));

    //Close the document
    document.close();
    System.out.println("... invoice pdf file created.");
}
con.close();

} catch (DocumentException de) {
    System.err.println(de.getMessage());
    de.printStackTrace();
} catch (IOException ioe) {
    System.err.println(ioe.getMessage());
}

```

```

        ioe.printStackTrace();
    } catch (SQLException sqle) {
        System.err.println(sqle.getMessage());
        sqle.printStackTrace();
    } catch (ClassNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
}
}

```

PHP PDF creator

In this section, we provide the source code for the PHP application that queries a DB2 z/OS database and creates a PDF invoice based on the DB2 data.

Example B-9 DbConnect.php

```

<?php
function getDb2Connection()
{
    $dsn='odbc:DB9G';
    $username='USER';
    $passwd='PASSWORD';
    $dbh = new PDO($dsn, $username, $passwd, array('PDO_ATTR_PERSISTENT' =>
false));
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING);
    return $dbh;
}
?>

```

Example B-10 InvoiceCreator.php

```

<?php
require('../lib/fpdf.php');
require ('DbConnect.php');

try
{
    //Connect to DB2 and retrieve all orders
    echo "Connecting to DB2 \n";
    $con=getDb2Connection();
    echo "... Connected \n";

    $orderNos = array();
    $i=0;
    $sql = "SELECT DISTINCT Order_No FROM BATCH.ORDER";
    foreach ($con->query($sql) as $row)
    {
        $orderNos[$i] = ($row[0]);
        $i++;
    }
    $sql=NULL;
}

```

```

//Create PDF invoices for each order number
for ( $orderCount = 0; $orderCount < count($orderNos); $orderCount++) {
    $currentOrderNo = $orderNos[$orderCount];
    echo "Starting to create pdf file
".$argv[1]."/Invoice_No_". $currentOrderNo.".pdf...\n";

    $pdf=new FPDF();
    $pdf->AddPage();
    $pdf->SetFont('Courier','',12);

    //Add sender information
    $text="Bobs Batch Store\n";
    $text=$text."JES Street 1111\n";
    $text=$text."12601 Poughkeepsie, NY\n";
    $text=$text."USA\n";
    $pdf->MultiCell(0,5,$text,0,R);

    //Add recipient information from database
    $sql = "SELECT DISTINCT BATCH.CUSTOMER.* FROM BATCH.CUSTOMER, BATCH.ORDER
WHERE BATCH.CUSTOMER.CUST_NO = BATCH.ORDER.CUST_NO AND BATCH.ORDER.ORDER_NO = " .
$currentOrderNo;
    foreach ($con->query($sql) as $row)
    {
        $text=$row["FIRST_NAME"]." ".$row["LAST_NAME"]."\n";
        $text=$text.$row["STREET_NAME"]." ".$row["STREET_NO"]."\n";
        $text=$text.$row["ZIP"]." ".$row["CITY"].", ".$row["STATE"]."\n";
        $text=$text.$row["COUNTRY"]."\n";
        $text=$text."\n\n\n";
        $pdf->MultiCell(0,5,$text,0,L);
        $i++;
    }
    $sql=NULL;

    //Add vertical line
    $pdf->SetDrawColor(0,0,255);
    $pdf->SetLineWidth(0.3);
    $pdf->Line(10,58,200,58);

    //Add table header data
    $pdf->SetDrawColor(0,0,0);
    $pdf->SetLineWidth(0.2);
    $pdf->SetFont('Times','B',12);
    $header=array('Pos','Quantity','Description','Single price','Total price');
    $w=array(13,20,87,35,35);
    for($i=0;$i<count($header);$i++)
        $pdf->Cell($w[$i],7,$header[$i],1,0,'L');
    $pdf->Ln();
    $linecount=0;
    $totalPrice=0;

    //Add order data from database
    $sql = "SELECT * FROM BATCH.ITEMS, BATCH.ORDER WHERE BATCH.ITEMS.ITEM_NO =
BATCH.ORDER.ITEM_NO AND BATCH.ORDER.ORDER_NO = ".$currentOrderNo." ORDER BY
BATCH.ORDER.POS_NO";
    foreach ($con->query($sql) as $row)

```

```

    {
        $pdf->SetFont('Times','I',12);
        $totalItem=$row["SINGLE_PRICE"]*$row[QUANTITY];
        $pdf->Cell(13,7,$row["POS_NO"],1,0,'L');
        $pdf->Cell(20,7,$row["QUANTITY"],1,0,'L');
        $pdf->Cell(87,7,$row["DESCRIPTION"],1,0,'L');
        $pdf->Cell(35,7,number_format($row["SINGLE_PRICE"],2).' USD ',1,0,'R');
        $pdf->Cell(35,7,number_format($totalItem,2).' USD ',1,0,'R');
        $pdf->Ln();
        $linecount++;
        $totalPrice=$totalPrice+$totalItem;
    }
    $sql=NULL;

    //Add table footer data
    $pdf->SetFont('Times','B',12);
    $footer=array('Total',' ',number_format($totalPrice,2).' USD ');
    $w=array(13,142,35);
    for($i=0;$i<count($header);$i++)
    $pdf->Cell($w[$i],7,$footer[$i],1,0,'R');
    $pdf->Ln();

    //Add vertical line
    $pdf->SetDrawColor(0,0,255);
    $pdf->SetLineWidth(0.3);
    $pos=88+$linecount*7;
    $pdf->Line(10,$pos,200,$pos);

    $pdf->SetFont('Times','I',10);
    $text="\n\nPlease read our terms and conditions.";
    $pdf->MultiCell(0,6,$text,0,L);
    $pdf->Output($argv[1].'Invoice_No_'. $currentOrderNo.'.pdf', F);
}

$con=NULL;
}
catch(Exception $e)
{
    echo "Failed: " . $e->getMessage();
    exit(0);
}
?>

```

Dynamic batch Web application

In this section, we provide the source code of a simple servlet that calls a session bean named `TriggerBatchBean`. The `TriggerBatchBean` is intended as an empty template to call z/OS batch jobs in different ways.

Example B-11 BatchServlet.java

```
package com.ibm.itso.sample;

import java.io.IOException;
import java.io.PrintWriter;

import javax.ejb.CreateException;
import javax.naming.InitialContext;
import javax.rmi.PortableRemoteObject;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class BatchServlet extends javax.servlet.http.HttpServlet implements
javax.servlet.Servlet {
    /**
     *
     */
    private TriggerBatchHome batchHome;

    private static final long serialVersionUID = 1L;

    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServlet#HttpServlet()
     */
    public BatchServlet() {
        super();
    }

    public void init(ServletConfig config) throws ServletException{
        //Look up home interface
        try {
            InitialContext ctx = new InitialContext();
            Object objref =
ctx.lookup("ejb/com/ibm/itso/sample/TriggerBatchHome");
            batchHome = (TriggerBatchHome)PortableRemoteObject.narrow(objref,
TriggerBatchHome.class);
        } catch (Exception NamingException) {
            NamingException.printStackTrace();
        }
    }

    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServlet#doGet(HttpServletRequest request,
HttpServletResponse response)
```

```

    */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        PrintWriter writer = response.getWriter();

        String parameter1 = request.getParameter("Parameter1");
        String parameter2 = request.getParameter("Parameter2");

        if (parameter1 == null && parameter2==null) {
            writer.println("<html>");
            writer.println("<title>Dynamic job submission on z/OS</title>");
            writer.println("<body>");
            writer.println("<h1>Welcome to the z/OS batch submitter
application</h1>");
            writer.println("<p>Please enter all required parameters.</p>");
            writer.println("<form method=\"get\">");
            writer.println("<p>Parameter 1: <input name=\"Parameter1\"
type=\"text\" size=\"80\" maxlength=\"100\"></p>");
            writer.println("<p>Parameter 2: <input name=\"Parameter2\"
type=\"text\" size=\"80\" maxlength=\"100\"></p>");
            writer.println("<input type=\"submit\" value=\" Submit\">");
            writer.println("<input type=\"reset\" value=\" Cancel\">");
            writer.println("</form>");
            writer.println("</body>");
            writer.println("</html>");
        } else {
            writer.println("<html>");
            writer.println("<title>Dynamic job submission on z/OS</title>");
            writer.println("<body>");
            writer.println("<h1>z/OS batch submitter application - Results</h1>");
            String returnMessage="";
            try {
                TriggerBatch triggerBatch=batchHome.create();
                String[] paramters= {parameter1, parameter2};
                returnMessage=triggerBatch.execute(paramters);
                if (returnMessage.indexOf("Error")==-1)
                {
                    writer.println("<p>"+returnMessage+"</p>");
                }
                else
                {
                    writer.println("<p><font
color=\"#FF0000\">"+returnMessage+"</font></p>");
                }
            } catch (CreateException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            writer.println("<p><a HREF=\"javascript:history.back()\">
Back</a></p>");
            writer.println("</body>");
            writer.println("</html>");
        }
        writer.close();
    }
}

```

```

    /* (non-Java-doc)
    * @see javax.servlet.http.HttpServlet#doPost(HttpServletRequest request,
    HttpServletResponse response)
    */
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        // TODO Auto-generated method stub
    }
}

```

Example B-12 TriggerBatchBean.java

```

package com.ibm.itso.sample;

/**
 * Bean implementation class for Enterprise Bean: TriggerBatch
 */
public class TriggerBatchBean implements javax.ejb.SessionBean {

    static final long serialVersionUID = 3206093459760846163L;
    private javax.ejb.SessionContext mySessionCtx;
    /**
     * getSessionContext
     */
    public javax.ejb.SessionContext getSessionContext() {
        return mySessionCtx;
    }
    /**
     * setSessionContext
     */
    public void setSessionContext(javax.ejb.SessionContext ctx) {
        mySessionCtx = ctx;
    }
    /**
     * ejbCreate
     */
    public void ejbCreate() throws javax.ejb.CreateException {
    }
    /**
     * ejbActivate
     */
    public void ejbActivate() {
    }
    /**
     * ejbPassivate
     */
    public void ejbPassivate() {
    }
    /**
     * ejbRemove
     */
    public void ejbRemove() {
    }
}

```

```

public String execute(String[] parameters)
{
    String message="You entered the following parameters: ";
    for (int i=0; i<parameters.length; i++)
    {
        message=message+" "+parameters[i];
    }
    return message;
}
}

```

JCL for running WebSphere Transformation Extender transformation in batch mode

In this section, we provide the JCL to run XML transformation to COBOL CopyBook format on z/OS.

Example B-13 JCL for running WebSphere Transformation Extender transformation in batch mode

```

//WTXBATCH JOB (999,POK),'L06R',CLASS=A,REGION=OM,
//          MSGCLASS=T,TIME=10,MSGLEVEL=(1,1),NOTIFY=&SYSUID
//*
//*****
/* Sample JCL to perform the XML2COB map for the IBM Websphere *
/* Transformation Extender for z/OS(MVS) utilizing BURST MODE *
/* processing. *
//*****
/* ----- Clean Output file
//SO EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DELETE WAGNERA.OUT.XML2COBO
/* NOTE: If the LE runtime library is not in the MVS LINKLIST,
/* the LE runtime library must be added to the STEPLIB
/* concatenation.
/* This job will allocate an output dataset and several
/* temporary workfiles. If you require a VOLSER specification,
/* the vol=ser= parameter must be added to the jcl.
//*****
//DTX EXEC PGM=DTXCMSDV,REGION=OM,
// PARM='XML2COB /IIM '/DD:MAPXSD''
//STEPLIB DD DISP=SHR,DSN=BB26159.SDTXLOAD
// DD DISP=SHR,DSN=IXM.SIXMLOD1
//*
/* The map ddname specified in the JCL must match the map name specified
/* on the command line defined in the PARM statement
//*
/* A map can be coded in JCL in any of the
/* following three ways:
//*
//* //XML2COB DD DSN=DTX.SDTXSAMP,DISP=SHR
//* The command line identifies the map by its ddname:
//* e.g. PARM='XML2COB <option> <option> etc.'

```

```

/**
/** //XML2COB DD DSN=DTX.SDTXSAMP(DTXBMMVS),DISP=SHR
/** The DD statement identifies the map as a member of
/** a PDS and the command line identifies the map by its ddname:
/** e.g. PARM='XML2COB <option> <option> etc.'
/**
/** //MAPLIB DD DSN=DTX.SDTXSAMP,DISP=SHR
/** The DD statement identifies the PDS only and the member
/** name is identified on the command line in parentheses
/** following the ddname that identifies the PDS:
/** e.g. PARM='MAPLIB(DTXBMMVS)'
/**
/**XML2COB DD DISP=SHR,DSN=WAGNERA.WTX.MAPS(XML2COB)
/**
/** Sysout datasets. SYSPRINT, SYSOUT and CEEDUMP are required by
/** Language Environment.
/**
/**DTXLOG DD SYSOUT=* Execution log
/**DTXAUD DD SYSOUT=* Audit file
/**DTXTRCE DD SYSOUT=* Trace file
/**SYSPRINT DD SYSOUT=*
/**SYSOUT DD SYSOUT=*
/**CEEDUMP DD SYSOUT=*
/**
/** Define the input dataset
/**INPUT DD PATH='/u/wagnera/input.xml'
/**
/** Define the schema definition
/**MAPXSD DD PATH='/u/wagnera/input.xsd'
/**
/** Define the output dataset
/**OUTPUT DD DSN=WAGNERA.OUT.XML2COB0,
/** DCB=(RECFM=VB,LRECL=80),
/** UNIT=SYSDA,
/** SPACE=(TRK,(1,1),RLSE),
/** DISP=(NEW,CATLG,DELETE)
/** Static temporary file allocations (if you want temporary files to be
/** dynamically allocated, remove or comment out the following statements).
/** The maximum number of temporary files used by a map is (2 * number of
/** inputs) + (2 * number of outputs) + 1. The minimum is equal to the number
/** of inputs + the number of outputs + 1. If static temporary allocations
/** are to be used they must be defined temporary datasets with RECFM=FBS;
/** The record length should be as large as convenient but not larger than
/** 32K. The BUFNO parameter should not be used. The amount of space
/** used by temporary files varies greatly from map to map depending on
/** map complexity, the amount of data being processed and the paging
/** options chosen.
/**
/**SYSTMP01 DD DSN=&&TEMP01,
/** DISP=(NEW,DELETE,DELETE),
/** DCB=(RECFM=FBS,LRECL=32760),
/** UNIT=SYSDA,
/** SPACE=(TRK,(5,1))
/**SYSTMP02 DD DSN=&&TEMP02,
/** DISP=(NEW,DELETE,DELETE),

```

```

//          DCB=(RECFM=FBS,LRECL=32760),
//          UNIT=SYSDA,
//          SPACE=(TRK,(5,1))
//SYSTMP03 DD DSN=&&TEMPO3,
//          DISP=(NEW,DELETE,DELETE),
//          DCB=(RECFM=FBS,LRECL=32760),
//          UNIT=SYSDA,
//          SPACE=(TRK,(5,1))
//SYSTMP04 DD DSN=&&TEMPO4,
//          DISP=(NEW,DELETE,DELETE),
//          DCB=(RECFM=FBS,LRECL=32760),
//          UNIT=SYSDA,
//          SPACE=(TRK,(5,1))
//SYSTMP05 DD DSN=&&TEMPO5,
//          DISP=(NEW,DELETE,DELETE),
//          DCB=(RECFM=FBS,LRECL=32760),
//          UNIT=SYSDA,
//          SPACE=(TRK,(5,1))

```

Triggering Java Stored procedure to generate PDF files

In this section, we provide the Java code for the GenPdf and TestSTP classes.

Example B-14 GenPdf.java - Trigger Java Stored procedure

```

package com.ibm.itso.sample;

import java.sql.SQLException;

public class GenPdf {

    public static void triggeredByMQ(String inMsg, String[] outMsg)
        throws SQLException {

        String temp = inMsg;
        String jobID = temp.substring(0, temp.indexOf(":"));
        temp = temp.substring(temp.indexOf(":") + 1);
        String action = temp.substring(0, temp.indexOf(":"));
        String pdfDir = temp.substring(temp.indexOf(":") + 1);

        PdfCreator pdfc = new PdfCreator();
        pdfc.generatePDF(action, pdfDir);

        outMsg[0] = new String("PDF(s) fuer JobID '" + jobID + "' successfully
        created.");
    }

    public static void runGenerate(String action, String pdfDir)
        throws SQLException {
        PdfCreator pdfc = new PdfCreator();
        pdfc.generatePDF(action, pdfDir);
    }
}

```

```
package com.ibm.itso.sample;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class TestSTP {

    /**
     * Start to test Java Stored Procedure
     */
    public static void main(String[] args) throws Exception {

        try {
            if (args.length != 3) {
                System.err.println("Input parameter is missing!");
                System.err.println("Para1: JDBC URL (e.g." +
                    "'jdbc:db2://<server>:<port>/<DB2Location>')");
                System.err.println("Para2: DB User");
                System.err.println("Para3: DB Password");

                throw new Exception("Input parameter is missing!!!!");
            }

            String URL = args[0];
            String User = args[1];
            String Pw = args[2];

            // connect to DB2
            Class.forName("com.ibm.db2.jcc.DB2Driver");
            System.out.println("Connection.....");
            Connection con = DriverManager.getConnection(URL, User, Pw);
            System.out.println("connected.");

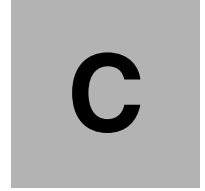
            String inMsg = "TestLocal:store_fs_db2:/u/wagnera/javastp/pdfs";

            System.out.println("Call STP.....");
            CallableStatement cs =
                con.prepareCall("CALL BATCH.TRIGGER_STP_WITH_MQ(?, ?)");
            cs.setString(1, inMsg);
            cs.registerOutParameter(2, java.sql.Types.VARCHAR);
            cs.execute();
            System.out.println("Call STP ended: " + cs.getString(2));

            cs.close();

            con.close();
            System.out.println("Connection closed.");
        } catch (ClassNotFoundException cnfe) {
            System.err.println(cnfe.getMessage());
            cnfe.printStackTrace();
        } catch (SQLException sqle) {
            System.err.println(sqle.getMessage());
        }
    }
}
```

```
        sqlc.printStackTrace();  
    }  
}  
}
```



Additional material

This book refers to additional material that you can download from the Internet as described in this appendix.

Locating the Web material

The Web material that is 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/SG247779>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG247779.

Using the Web material

The additional Web material that accompanies this book includes the following files:

<i>File name</i>	<i>Description</i>
COBOL_data_generator.zip	COBOL job as described in 7.5, “Sample stand-alone Java batch application” on page 82
Java_STP_pdf_generator.zip	Java DB2 stored procedure as described in 6.3, “Java in DB2 for z/OS” on page 64
Java_pdf_generator.zip	Java job as described in 7.5, “Sample stand-alone Java batch application” on page 82
PHP_pdf_generator.zip	PHP job as described in 9.4, “Sample application for using PHP in stand-alone batch” on page 161

DynamicBatchRADProject.zip	The Dynamic Batch application as Rational Application Developer project as described in 14.1, “Job submission with native Java technology” on page 224.
NativeJavaEjb.zip	The EJB that calls a z/OS batch job with native Java technology from 14.1, “Job submission with native Java technology” on page 224.
WTX_sample.zip	WebSphere Transformation Extender example as described in 21.3, “Sample using WebSphere Transformation Extender in z/OS batch” on page 375.
WXDCG_sample.zip	WebSphere XD Compute Grid sample workspace as described in 8.7, “Developing a WebSphere batch application” on page 109.
SchedulerWrapperProject.zip	The Scheduler Wrapper Test using the EJB interface for scheduling WebSphere XD Compute Grid jobs as Eclipse project as described in 14.2.4, “Web services and EJB interfaces for the Job Scheduler” on page 249 and 14.1.1, “Developing the code” on page 225
STP_TriggeredByMQ.zip	Trigger Stored procedure by using MQ as described in “Creating the DB2 stored procedure to process MQ message” on page 271
STP_TriggeredByMQ_EJB.zip	The EJB that calls send a message to MQ to trigger DB2 Stored procedure by MQListener as described in “Creating the DB2 stored procedure to process MQ message” on page 271
STP_TriggeredByWebService.zip	Trigger a stored procedure by using Web Services as described in 14.4.2, “DB2 as a Web service provider” on page 279
Batch_Design_Principles.zip	Using the BDS Framework in WebSphere Compute Grid, including a shared service example, as described in Chapter 17, “Batch application design and patterns in WebSphere XD Compute Grid” on page 319

System requirements for downloading the Web material

The following system configuration is recommended for downloading and unpacking the provided .zip files:

Hard disk space: 40 MB minimum to download and 100 MB minimum to extract
Operating System: Windows

How to use the Web material

Create a subdirectory (folder) on your workstation, and decompress the contents of the Web material .zip files into this folder. Follow the instructions in the chapter that describe the sample to get started with the additional material.

Related publications

We consider the publications that we list in this section particularly suitable for a more detailed discussion of the topics that we cover in this book.

IBM Redbooks publications

For information about ordering these publications, see “How to get IBM Redbooks publications” on page 457. Note that some of the documents referenced here might be available in softcopy only.

- ▶ *DB2 for z/OS Stored Procedures: Through the CALL and Beyond*, SG24-7083
- ▶ *DB2 UDB for z/OS: Design Guidelines for High Performance and Availability*, SG24-7134
- ▶ *DB2 9 for z/OS Performance Topics*, SG24-7473
- ▶ *DB2 for z/OS and WebSphere: The Perfect Couple*, SG24-6319
- ▶ *DB2 for z/OS and OS/390: Ready for Java*, SG24-643535
- ▶ *EEnabling z/OS Applications for SOA*, SG24-7669
- ▶ *IBM WebSphere Transformation Extender 8.2*, SG24-7693
- ▶ *Java Stand-alone Applications on z/OS, Volume I*, SG24-7177
- ▶ *Java Stand-alone Applications on z/OS Volume II*, SG24-7291
- ▶ *Java Application Development for CICS*, SG24-5275
- ▶ *Topics on Version 7 of IBM Rational Developer for System z and IBM WebSphere Developer for System z*, SG24-7482
- ▶ *IBM WebSphere MQ File Transfer Edition Solution Overview*, REDP-4532

Other publications

These publications are also relevant as further information sources:

- ▶ *Application Programming Guide and Reference for Java*, SC18-9842
- ▶ *XML Guide*, SC18-9858
- ▶ *DB2 9 Performance Monitoring and Tuning Guide*, SC18-9851
- ▶ *WebSphere z/OS - The Value of Co-Location*, WP101476
- ▶ *Enterprise COBOL for z/OS Programming Guide Version 4 Release 1*, SC23-8529
- ▶ *Enterprise PL/I for z/OS Programming Guide Version 3 Release 8*, SC27-1457
- ▶ *Pattern-Oriented Software Architecture: A System Of Patterns*. West Sussex, England, ii. Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, John Wiley & Sons Ltd., 1996

Online resources

These Web sites are also relevant as further information sources:

- ▶ IBM z/OS Java
<http://www.ibm.com/servers/eserver/zseries/software/java/>
- ▶ JZOS
<http://www.ibm.com/servers/eserver/zseries/software/java/products/jzos/overview.html>
- ▶ JZOS Toolkit javadoc
<http://www.ibm.com/developerworks/java/zos/javadoc/jzos/index.html>
- ▶ JZOS Sample applications
<http://www.ibm.com/servers/eserver/zseries/software/java/products/jzos/jzossamp.html>
- ▶ JZOS Cookbook
<http://www.alphaworks.ibm.com/tech/zosjavabatchtk/download>
- ▶ z/OS Java 6.0 User Guide
<http://download.boulder.ibm.com/ibmdl/pub/software/dw/jdk/diagnosis/diag60.pdf>
- ▶ z/OS Java 6.0 Diagnostics Guide
<http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/topic/com.ibm.java.doc.us.er.zos.60/collection-welcome.html>
- ▶ JinsightLive for IBM System z
<http://www.alphaworks.ibm.com/tech/jinsightlive>
- ▶ iText Java PDF library
<http://www.lowagie.com/iText/>
- ▶ FTP Ant task
<http://ant.apache.org/manual/OptionalTasks/ftp.html>
- ▶ z/OS UNIX System Services Tools and toys
<http://www.ibm.com/servers/eserver/zseries/zos/unix/tools/>
- ▶ PHP on z/OS
<http://www.ibm.com/servers/eserver/zseries/zos/unix/ported/php/index.html>
- ▶ Eclipse PDT
<http://www.eclipse.org/pdt/downloads/>
- ▶ FPDF PHP library
<http://www.fpdf.org>
- ▶ WebSphere MQ File Transfer Edition Information Center
http://publib.boulder.ibm.com/infocenter/wmqfte/v7r0/index.jsp?topic=/com.ibm.wmqfte.home.doc/help_home_wmqfte.htm
- ▶ Managing Batch process in an SOA: Application Development Trends, The Lee, 2008
<http://adtmag.com/Articles/2008/02/25/Managing-Batch-Processing-in-an-SOA.aspx?Page=4>

How to get IBM Redbooks publications

You can search for, view, or download IBM Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

A

ADD CLONE 213
ALTER TABLE 213
Ant libraries, in Eclipse 72

B

Batch

tuning

DFSMSHsm 352
IEFACTRT 353
Life Of A Data Set (LOADS) 347
OPEN and CLOSE of data sets 346
OPEN and CLOSE, non-VSAM data sets 346
OPEN and CLOSE, VSAM data sets 346
SMF 30-4 record 346
SMF 30-5 record 345
System Log (SYSLOG) 354
Tivoli Workload Scheduler 352

Batch window 11

Batch window, optimizing

Critical Path Analysis 304
data in memory (DIM) techniques 305
DB2 buffer pools 305
dependency diagram 303
DFSORT 305
Gantt chart 302
I/O 305
parallelism 305
Queued Sequential Access Method (QSAM), buffering 305
seven approaches 304
Virtual I/O (VIO) 305
VSAM Local Shared Resources, buffering 305

BatchPipes

advantages 311
BatchPipePlex 314
BatchPipeWorks 314, 316
connectors 314
CICS 314
DB2 Template Utility 314
FTP 314
data sets supported 311
fittings pipeline 316
half pipe 315
pipeline 315
setting up 313
SMF records 351
use ID access 313
using, blocksize 313
using, DD statement 313
Version 2 314

BatchPipes, and DFSORT 316

BatchPipeWorks 309

Binary (stream) mode 52

Binder 34

BPXBATCH 78

BPXBATSL 78

Bulk processing reference architecture

Analytics 27
Bulk application container 22–23
Bulk application development 26
Data access management services 22
Infrastructure services 22
Invocation and scheduling services 22
Invocation services 24
System management and operations 25

Business Action Language (BAL) 368

Business Object Model (BOM) 363

Business policy 360

Business Process Execution Language (BPEL) 17

Business Rule Management 360

Business Rule Management System (BRMS) 359–360

Business rule mining 369

Business rules 360

C

Cache

removing data from 209
Climb Strategy 209
First In First Out (FIFO) 209
Least Frequently Used (LFU) 209
Least Recently Used (LRU) 209
Random 209

Caching 209

hardware 209

Caching, in DB2

dynamic statement cache 209

Caching, in WebSphere Application Server 209

Change Capture Agent 196

Checkpoint 24

Checkpoint interval 24

Classic Data Architect 196

Cleansing 191

CLOB (Character Large Object) 46

COBOL

XML support, built-in 36
ENCODING phrase 37
NAMESPACE-DECLARATION 37
NAMESPACE-DECLARATION XML 37
processing input 36
processing output 37
registers 37
XML GENERATE statement 37
XML PARSE statement 36–37
XML-CODE 37
XML-EVENT 37
XML-NAMESPACE 37
XML-NAMESPACE-PREFIX 37

- XML-NNAMESPACE 37
- XML-NNAMESPACE-PREFIX 37
- XML-NTEXT 37
- XMLPARSE compiler option 36
- XMLPARSE(COMPAT) compiler option 37
- XMLPARSE(XMLSS) compiler option 36
- XML-TEXT 37
- COBOL and PL/I
 - C++ interoperability 35
 - Java interoperability 35
 - Unicode support 35
 - XML support 35
- Codepage, UNIX System Services 54
- Coupling Facility Resource Management (CFRM) 313

D

- Data Mart 190
- Data sources 195
- Data Storage Area (DSA) 198
- Data Warehouse 190
 - workloads, supported 190
- Data Warehousing 189
- Data Web Services (DWS) 280
- Data, cleanse 194
- Data, deliver 194
- Data, extraction stage 193
- Data, federation 195
- Data, publication 196
- Data, replication 196
- Data, transform 194
- Data, understand 194
- DataStage 194
- DataStage ETL Accelerator 202
- DB2
 - explain, for access paths 212
 - multi-row processing 212
- DB2 for z/OS
 - triggering a stored procedure
 - MQListener 270
 - Java code 272
 - JDBC drivers 276
 - remove tasks 275
 - starting tasks 274
 - stop/restart tasks 274
 - stored procedure, defining 272
 - stored procedure, testing locally 273
 - tasks, configuring 273
- DB2 on z/OS
 - buffer pool, maximum size, in DB2 V9 218
 - buffer pool, size 218
 - checkpointing 221
 - clone tables 213
 - Data Set VSAM Control Interface (CI) size 219
 - DSNZPARMs 219
 - fast table append 217
 - index, when using expressions 217
 - Isolation Level 215
 - Cursor Stability (CS) 215
 - JDBC 216
 - Read Stability (RS) 215

- Repeatable Read (RR) 215
- Uncommitted Read (UR) 215
- webSphereDefaultIsolationLevel 216
- Isolation Level, setting 216
 - at DB2 package/plan level 216
 - at SQL statement level 216
 - in a Java EE application 216
 - in a Java native application 216
- JDBC, optimizing 219
- lock avoidance 216
- locking
 - intent modes 213
- locks 213
- locks, compatibility for page and row locks 214
- locks, compatibility for table spaces 215
- Real Time Statistics (RTS) 217
- REBIND 218
- REORG 218
- RUNSTATS 218
- SEQPRES 219
- tuning 343
 - Accounting Trace 344, 348
 - DB2 Catalog 344
 - Explain facility 350
 - Performance Trace 350
 - PLAN_TABLE 350
 - SMF, 42-6 records 344
 - Statistics 345
 - Statistics Trace 343–344
 - Statistics Trace, areas of activity instrumented 344
 - SYSIBM.SYSCOPY 352
 - SYSIBM.SYSPACKSTMT 350
 - SYSIBM.SYSSTATEMENT 350
 - Web service provider 279
- DB2 tables, joining 210
- db2mqln1 274
- db2mqln2 274
- DFSMSshsm
 - SMF 352
- DFSORT 405
 - combining records 410
 - data supported 406
 - IFTHEN 412
 - IFTHEN PARSE 409
 - IFTHEN, types 412
 - invoking functions 406
 - OVERLAY 408
 - PARSE 409
 - parsing 409
 - record selection 407
 - reformatting records 408
 - SMF records 351
- DFSORT, invoking with JZOS
 - DfSort class 417
- DL/I (Data Language/I) 59
- DLIModelUtility plug-in 59
- DOM (Document Object Model) 39
- DRDA and zAAP eligibility 46
- DRDA and zIIP eligibility 46

E

- EBCDIC, using PDF files in 171
- Electronic Data Interchange (EDI) 374
- ETL Accelerator 201
- European Single Euro Payments Area (SEPA) 377
- extract, transform, and load (ETL) 20, 189, 191, 198
 - cleansing 191
 - data refresh 192
 - data replication 192
 - extracting data faster 193
 - extraction process 191
 - file locking 198
 - load process 191
 - tools, on System z 193
 - transformation process 191
 - trickle feed 192

H

- Health Insurance Portability and Accountability Act (HIPAA) 374
- Hibernate 60, 325
- HiperSockets 201

I

- IBM InfoSphere 194
- IBM InfoSphere Datastage 189
- IBM InfoSphere Information Server suite 194
- IBM Rational Developer for System z 81
- IBM WebSphere ILOG BRMS 360
- IBM WebSphere ILOG BRMS (ILOG BRMS) 360
- ICETOOL
 - checking data 411
 - OCCUR 410
 - RANGE 410
 - SELECT 410
 - SPLICE 411
 - STATS 411
 - UNIQUE 411
- ILOG BRMS 360
 - Business Action Language (BAL) rules 364
 - Decision Services 364
 - Decision Validation Services (DVS) 364
 - JRules 364
 - Rule Execution Server 364
 - Rule Studio 362
 - Rule Team Server (RTS) 362
 - Rules for COBOL 364
- IMS Database (IMS DB) 58
- IMSDatabaseView object 59
- Information Server Client Package interface 201
- InfoSphere Change Data Capture 197–198, 202
 - Q replication 199
 - SQL replication 199
- InfoSphere Classic Data Event Publisher 196
- InfoSphere Classic Federation 196
- InfoSphere Classic Federation Data Server 196
- InfoSphere Data Event Publisher 196
- InfoSphere DataStage 200
 - parallel processing 204

- partition parallelism 204
- pipeline parallelism 204

- stage 203
- stages 203
- InfoSphere DataStage Designer 203
- InfoSphere DataStage Enterprise for z/OS 201
- InfoSphere DataStage MVS Edition 201
- InfoSphere DataStage Transformation Extender 203
- InfoSphere Replication Server 197
- In-memory data grid (IMDG) 23
- In-memory database (IMDB) 23
- Inter Language Communication (ICL) 47
- Inter Language Communication (ICL), and Java 47
- iText-2.1.5.jar 70

J

- Java EE batch application 100
- Java EE environment, on z/OS 95
- Java EE standard 94
- Java Native Interface (JNI) 55
- Java on z/OS
 - advantages 50
 - APIs, specific to z/OS 51
 - CICS 58
 - encoding 55
 - IMS 58
 - interoperability, with Enterprise COBOL 55
 - interoperability, with Enterprise PL/I 56
 - interoperability, with other languages 55
 - run times 51
- Java on z/OS, stand-alone
 - invocation 78
 - tooling 80
- Java Persistence API (JPA) 60
- Java Persistence Architecture (JPA) 325
- Java Record I/O library (JRIO) 52
- Java Stored Procedure, in DB2
 - calling 64
 - configuring 65
 - creating 72
 - JAVAENV file 66
 - sample application 67
 - testing 75
 - WLM Application Environment 65
 - WLM Application Environment, refreshing 74
 - WLM environment startup procedure 65
- Java stored procedure, in DB2 64
- Java, in IMS 58
 - data access 58
 - development 63
 - interoperability, with other languages 63
- Java Batch Processing (JBP) 60
 - checkpoints 62
 - example 62
 - restarting 62
 - starting 61
- Java DataBase Connectivity (JDBC) 59
- Java Message Processing (JMP) 60
- Java, performance 338
 - Ahead-of-time (AOT) compilation 340

- garbage collection 338
 - gencon policy 338
- JinsightLive 338
- Just In Time compiler (JIT) 339
- JVM, startup 339
 - profiler 338
 - shared classes 339
- Job Control Language (JCL) 99
- Job Entry Subsystem (JES) 99
- Jobs, triggering
 - using native Java 224
 - using Tivoli Workload Scheduler 263
 - using WebSphere XD Compute Grid 234
 - command line interface 240
 - command line interface, example 241
 - lrcmd.sh script 240
 - using base scheduler 249
 - Web Services and EJB interfaces 249
 - Web Services interface 261
 - Web Services interface, tools 262
 - WSGrid utility 242
- JZOS batch launcher 53, 79
 - supported SDK version 79
- JZOS Launcher 101
- JZOS toolkit 52
 - MVS data sets supported 52

L

- Language Environment 47
- Load module 34
- LOAD utility 202

M

- Metadata Server 194
- Modified Indirect Data Address Word (MIDAW) 218

N

- Non-window batch 306

O

- Object code 34
- Object-relational mapper 60
- online transaction processing (OLTP) 20
- online transaction processing (OLTP) system 191
- OUTFIL
 - combining records 410
 - report writing 410

P

- Performance instrumentation 341
 - SMF 342
- PHP 158
- PHP on z/OS
 - batch launcher options 159
 - DB2 on z/OS, access to 158
 - DB2, access to 168
 - development

- Eclipse, setup 162
- PHP Development Tools (PDT), download 162
- development tools 160
- Eclipse 160
 - in batch 158
 - interoperability, with other languages 160
 - introduction 158
 - php command 158
 - PHP Development Tools (PDT) 160
 - PHP interpreter 158
 - PHP libraries 158
 - sample, steps 160
- PHP, what is it? 158
- Pipes, in UNIX 317
- PL/I
 - XML support, built-in 38
 - PLISAXA 38
 - PLISAXB 38
 - PLISAXC 38
 - PLISAXx subroutines 38
 - XMLCHAR function 38
 - zAAP eligibility 39
 - PL/I Simple API for XML (SAX) parser 38
 - pureXML 45

Q

- QualityStage 194

R

- Record mode 52
- Redbooks Web site 457
- Redbooks website
 - Contact us xv
- Replication, modes 194
 - continuous mirroring 194
 - periodic mirroring 194
 - refresh 194
- Resource broker 25
- Rule Team Server (RTS), repository 363

S

- SAP IDOC 374
- SAX2 (Simple API for XML) 39
- Service-Level Agreement (SLA) 11
- SMF record types 342
 - 101 348–349
 - 14 346
 - 15 346
 - 16 351
 - 241 352
 - 30-4 346
 - 30-5 345
 - 42-6 344, 346
 - 62 346
 - 64 346
 - 91 351
 - subtype 11 351
 - subtype 12 351

- subtype 13 351
- subtype 14 351
- subtype 15 351
- Staging tables 198
- straight through processing (STP) 20
- Stream mode 52
- Symbols, in DFSORT and ICETOOL 414
- System Authorization Facility (SAF) 104

T

- Text (stream) mode 52
- Tivoli Dynamic Workload Broker (TDWB) 26
- Tivoli Dynamic Workload Broker agent 264
- Tivoli Workload Automation 264
- Tivoli Workload Scheduler 11
 - Application Description (AD) 353
 - automatic job rerun 265
 - calendar and event based scheduling 263
 - Critical Path 265
 - fault tolerance 264
 - file transfer, integrated 266
 - high availability 264
 - integration
 - AF Operator 265
 - Tivoli System Automation for z/OS 265
 - interfaces 266
 - Query Current Plan (QCP) 353
 - reporting 266
 - scalability 264
 - Tivoli Enterprise Portal, integration 264
 - Workload Manager, integration 265
- Tivoli Workload Scheduler for z/OS
 - strengths 263
- Tools, for z/OS UNIX Systems Services 80
- Trickle feed 192

W

- Web 2.0 17
- Web Services 17
- WebSphere Application Server for z/OS 95
- WebSphere Data Event Publisher 202
- WebSphere Extended Deployment (XD) Compute Grid for z/OS 93
 - Batch container 100
 - Batch Data Stream (BDS) 103
 - Batch Packager 119
 - BDS framework
 - ByteReaderPattern 115
 - ByteWriterPattern 115
 - FileReaderPattern 115
 - FileWriterPattern 115
 - JDBC, supported classes 114
 - JDBCReaderPattern 114
 - JDBCWriterPattern 114
 - JPARReaderPattern 115
 - JPAWriterPattern 115
 - RecordOrientedDatasetReaderPattern 115
 - RecordOrientedDatasetWriterPattern 115
 - components 99

- Compute Grid Test Server 129
- Compute Grid UTE 129
 - download 129
 - setting up 129
- development
 - Batch Data Stream (BDS) Framework 111
 - Batch Packager 111
 - Batch Simulator 111
 - BDS framework 114
 - IDE 111
 - job submission, options 132
 - life cycle 112
 - packaging properties, generating 118
 - Unit Test Environment (UTE) 111
- GenericXDBatchStep 114
- Grid Execution Endpoint (GEE) 99–100
- high availability 104
- job definition, options 117
- Job Management Console (JMC) 99, 137
- Job Scheduler (JS) 99–100
- Job Scheduler (JS), access methods 105
- Ircmd 99
- On-Demand Router (ODR) 105
- overview 97
- Parallel Job Manager (PJM) 99–100, 105
- programming 106
- programming model
 - batch data stream 108
 - batch job step 108
 - BatchDataStream interface 109
 - BatchJobStepInterface interface 109
 - checkpoint algorithm 109
 - CheckpointPolicyAlgorithm interface 109
 - principal interfaces 109
 - results algorithm 109
 - ResultsAlgorithm interface 109
- programming models 107
 - compute-intensive 107
 - transactional batch 107
- Quality of Service (QoS) 103
- security 104
- security roles 104
 - assignment 104
 - lradmin 104
 - lrsubmitter 104
- testing, options 124
- testing, using Batch Simulator in Eclipse 124
- testing, using Batch Simulator on z/OS 138
- tools 103
- why to use 103
- workload types 97
- xJCL 99
- WebSphere ILOG Rules for COBOL 367
 - Business Object Model (BOM) 368
- WebSphere MQ File Transfer Edition (FTE) 398
 - FTE Agent 398
 - logging 399
 - operational interfaces 398
 - Queue Manager 398
 - transfer, triggering

- using Ant 403
 - using JZOS 401
- WebSphere Transformation Extender 371, 373–374
 - advantages 372
 - benefits 373
 - mapping file 377
 - Type Tree 381
- WebSphere Transformation Extender Design Studio 371, 376
 - components 376
 - Map Editor 376
 - Type Tree Editor 376
 - development
 - steps 376
 - map, building 392
 - map, creating 388
 - map, testing 392
 - mapping file, creating 384
 - mapping file, creating a z/OS version 393
 - starting 379
 - transferring files to z/OS 393
 - XML files, importing 380
- WebSphere XD Compute Grid
 - classification rules 235
 - job class 235
 - job log 235
 - Job Management Console (JMC) 235
 - access 235
 - job schedules, managing 235
 - Job scheduler (JS) 100
 - scheduling jobs with an external scheduler 266
 - Service Integration Bus (SIBus) 268
 - WSGrid utility
 - invoking 246
 - xJCL 100
- WebSphere XD Operations Optimization feature 105

- z/OS XML System Services 36, 39
 - specialty engines, using 36

X

- xJCL 99
- XL/XP XML parser 42
- XML Extender 46
- XML in DB2
 - creating a table with XML 68
 - XMLTABLE function 71
- XML Parser, C++ Edition 39
- XML parsing on z/OS, with DB2 V9 44
- XML parsing, issues 41
- XML processing options on z/OS 36
- XML support, in COBOL and PL/I 35
- XML System Services 16
- XML Toolkit for z/OS 39, 41–42
- XQuery 16

Z

- z/OS BatchPipes 201
- z/OS C/C++ library, I/O 52
- z/OS XML parser 40
 - character encodings 40
 - specialty engine eligibility 40



Redbooks

Batch Modernization on z/OS

(0.5" spine)
0.475" x 0.873"
250 x 459 pages



Redbooks®

Batch Modernization on z/OS

Provides an overview of current batch processing technology

Discusses how to use, simplify, and improve batch processing

Includes example scenarios that use batch processing

Mainframe computers play a central role in the daily operations of many of the world's largest corporations, and batch processing is a fundamental part of the workloads that run on the mainframe. A large portion of the workload on IBM z/OS systems is processed in batch mode. Although several IBM Redbooks publications discuss application modernization on the IBM z/OS platform, this book specifically addresses batch processing in detail.

Many different technologies are available in a batch environment on z/OS systems. This book demonstrates these technologies and shows how the z/OS system offers a sophisticated environment for batch. In this practical book, we discuss a variety of themes that are of importance for batch workloads on z/OS systems and offer examples that you can try on your own system. The book also includes a chapter on future developments in batch processing.

The audience for this book includes IT architects and application developers, with a focus on batch processing on the z/OS platform.

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

SG24-7779-01

ISBN 0738436968