Mixed Workloads in WebSphere XD V6.0 on z/OS

- OLTP and long-running J2EE executions
- Workload Management integration
- Sample scenarios

G Michael Connolly
David SK Chan
Rajesh Ramachandran
Jeff Sampson
Bart Tague

ibm.com/redbooks
International Technical Support Organization

Mixed Workloads in WebSphere XD V6.0 on z/OS

December 2006
Note: Before using this information and the product it supports, read the information in “Notices” on page vii.
Contents

Notices ................................................................. vii
Trademarks ............................................................... viii

Preface ................................................................. ix
The team that wrote this IBM Redbook ................................. ix
Become a published author ................................................ x
Comments welcome ....................................................... xi

Chapter 1. Introduction .................................................. 1
1.1 WebSphere Extended Deployment for z/OS .......................... 2
1.2 Overview .................................................................. 3

Chapter 2. Concepts ........................................................ 11
2.1 Service policies and the On Demand Router ......................... 12
  2.1.1 Service policies .................................................. 12
  2.1.2 On Demand Router ............................................. 14
  2.1.3 ODR and z/OS WLM integration .............................. 18
  2.1.4 Shared workload management .................................. 21
2.2 The long-running environment ........................................ 22
  2.2.1 Overview .......................................................... 22
  2.2.2 Long-running environment components ....................... 24
  2.2.3 Anatomy of a long-running batch application ............... 27
2.3 Why WebSphere Batch on z/OS .................................... 35
  2.3.1 WebSphere Extended Deployment on z/OS Benefits .......... 36
2.4 Security .................................................................. 38

Chapter 3. Our environment .............................................. 39
3.1 Infrastructure considerations ........................................... 40
3.2 The architecture we used ............................................... 41
  3.2.1 Our front-end systems .......................................... 42
  3.2.2 Our back-end systems ......................................... 43

Chapter 4. Establishing the OLTP Environment ......................... 49
4.1 Setting up the ODR .................................................... 50
  4.1.1 Planning consideration for setting up the ODR ............ 50
  4.1.2 Creating a new ODR in the administrative console ........ 51
  4.1.3 Modifying default values in the new ODR ..................... 54
4.2 Setting up the OLTP scenario dynamic cluster ................... 56
  4.2.1 Creating the node group for the dynamic cluster ............ 56
4.2.2 Creating the node group for the dynamic cluster .................. 57
4.2.3 Modifying the default values in the new dynamic cluster ........ 58

Chapter 5. Online transaction processing (OLTP) scenario ........... 61
5.1 OLTP scenario description ........................................... 62
   5.1.1 XDStock application ............................................ 63
5.2 Installing the XDStock application .................................. 64
5.3 Configuring WLM on z/OS .......................................... 66
5.4 Configuring WLM WebSphere XD .................................... 71
   5.4.1 Creating the service policies and transaction classes .......... 71
   5.4.2 Adding work classes to the XDStock application ............. 76
5.5 Adjusting servant minimum and maximum .......................... 81
5.6 Activating the propagation of TCLASS .............................. 82
5.7 Executing the scenario .............................................. 85
   5.7.1 Starting the On Demand Routers ............................ 86
   5.7.2 Starting the application servers in the dynamic cluster .... 86
   5.7.3 Preparing to observe the scenario ........................... 87
   5.7.4 Applying the workload ...................................... 91
   5.7.5 Conclusion .................................................... 97

Chapter 6. Establishing the long-running environment ............... 99
6.1 Configuring the long-running environment .......................... 100
6.2 Configuration the long-running scheduler .......................... 100
   6.2.1 Long-running scheduler configuration overview ............. 100
   6.2.2 Creating node group and dynamic cluster for LRS .......... 101
   6.2.3 Additional configuration for the LRS dynamic cluster ....... 107
   6.2.4 Creating the LRS database and tables ...................... 114
   6.2.5 Configuring the JDBC provider and LRS data source ....... 115
   6.2.6 Deploying the Long-Running Scheduler (LRS) application .. 124
6.3 Configuring long-running execution environment ................... 127
   6.3.1 Overview ................................................... 128
   6.3.2 Creating a node group and dynamic cluster for LREE ....... 129
   6.3.3 Creating the LREE database and tables .................... 132
   6.3.4 Defining and testing the LREE data source ................. 133
   6.3.5 Deploying LREE application ................................ 134
6.4 Testing the Long-Running Environment ................................ 135

Chapter 7. Long-running batch workloads .............................. 137
7.1 Long-running sample application overview ......................... 138
   7.1.1 The BatchTrade application ................................ 138
   7.1.2 BatchTrade xJCL ............................................ 140
   7.1.3 Submitting BatchTrade jobs ................................ 140
   7.1.4 Monitoring and managing BatchTrade jobs ................. 140
7.2 Installing the sample applications ................................ 141
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 about the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

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

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

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

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

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

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

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

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

Redbooks (logo)™ ibm.com® DB2® Redbooks™
IBM® IMS™ RACF®
z/OS® MVS™ RMF™
z/VM® OS/390® Tivoli®
zSeries® Rational® WebSphere®
CICS®

The following terms are trademarks of other companies:

EJB, Forte, Java, JDBC, JMX, JVM, J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. 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.

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

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

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

This IBM® Redbook extends the material provided in the *Using WebSphere Extended Deployment V6.0 To Build an On Demand Production Environment*, SG24-7153, by providing a z/OS®-centric description of the On Demand Router (ODR) and the long-running application support provided with WebSphere® Extended Deployment (XD) 6.0.

This book demonstrates the additional value the ODR provides as it is integrated with the existing z/OS Workload Manager. It also describes the long-running application support now available on z/OS with XD 6.0 along with its relationship to the traditional z/OS batch facilities.

This book describes an XD 6.0 installation into our existing ITSO High Availability WebSphere Network Deployment environment as well as step-by-step instructions that take you through the configuration of the ODRs and dynamic clusters. It presents the setting up of a long-running application environment in detail.

It uses a sample scenario to demonstrate the integration of an ODR Service Policy with a corresponding WLM Service Policy and the resulting finer grained WLM workload classification that the ODR now makes possible. Additional scenarios take you through the deployment and execution of a long-running workload.

The team that wrote this IBM Redbook

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

**G Michael Connolly** is an IT consultant at the International Technical Support Organization, Poughkeepsie Center. He has more than 30 years of IBM software development experience in both distributed systems and the mainframe zSeries®. He holds a BA in Humanities from Villanova University. His areas of expertise include TCP/IP Communications, UNIX® System Services, WebSphere for z/OS, and most recently the EWLM product.

**David SK Chan** is an Advisory IT Specialist working for IBM Global Services in Singapore. He has eight years of experience in supporting and helping customers in the mainframe z/OS and, since 2003, in zLinux on the zSeries. He is also a Red Hat Certified Engineer (RHCE). His technical interests include
Linux®, WebSphere, z/OS, and distributed systems. He holds a Bachelor of Science in Information Systems and Computer Science from the National University of Singapore.

**Rajesh Ramachandran** is a Senior Software Engineer in IBM zSeries e-Business Services. He currently works in the Design Center in Poughkeepsie as an IT Architect and DB2® assignee. He has 12 years of experience in application development on various platforms, including z/OS, UNIX, and Linux utilizing COBOL, Java™, CICS®, and Forte™.

**Jeff Sampson** is a Certified Consulting IT Specialist working for IBM WebSphere Technical Sales in the United States. He has 18 years of experience developing software in C, C++, and Java and deploying it on distributed systems and z/OS as well as various embedded systems. He has worked with WebSphere on z/OS and its CORBA predecessor for the past nine years.

**Bart Tague** is a Certified IT Specialist working in the IBM Design Center for On Demand Business located in Poughkeepsie, New York. For the past seven years he has been involved in architecting and designing enterprise WebSphere-based solutions for major IBM clients.

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

Richard Conway  
International Technical Support Organization, Poughkeepsie Center

Chris Vignola  
IBM Poughkeepsie

Keith Smith  
IBM Raleigh

---

**Become a published author**

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

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.
Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

► Use the online Contact us review book form found at:
  ibm.com/redbooks

► Send your comments in an e-mail to:
  redbook@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
Chapter 1. Introduction

WebSphere Extended Deployment (XD) offers enhanced quality of service features focused on the Java 2 Platform, Enterprise Edition (J2EE) application environment and integrates seamlessly into an existing WebSphere Application Server Network Deployment V6 installation on all supported platforms. This chapter introduces IBM WebSphere Extended Deployment for z/OS V6 and highlights several expanded features. These features are the primary focus of this book.
1.1 WebSphere Extended Deployment for z/OS

WebSphere Application Server Extended Deployment for z/OS offers the same leading-edge dynamic operations, manageability, and high performance capabilities available with WebSphere Application Server Extended Deployment for distributed platforms. These capabilities are thoroughly documented in IBM Redbook Using WebSphere Extended Deployment V6.0 To Build an On Demand Production Environment, SG24-7153. If you are new to WebSphere Extended Deployment, we recommend that you read that book before proceeding with this book. This book was written with the assumption that the reader is already familiar with all of the new WebSphere Extended Deployment terminology and features.

WebSphere Extended Deployment introduces a new type of workload to the J2EE world: long-running workloads. WebSphere Application Server Extended Deployment for z/OS expands on the existing z/OS support for WebSphere, such as virtualization, goals-directed policy, and the capability to run both traditional transactional workloads and the new long-running workloads. This book focuses primarily on these expanded capabilities. This book addresses the following scenarios to illustrate these expanded capabilities:

- How workload manager (WLM) integrates with WebSphere Extended Deployment for z/OS to manage goal-oriented transactional workloads
- How WebSphere Extended Deployment for z/OS manages mixed transactional and long-running workloads in dedicated dynamic clusters in the same WebSphere cell
- How WebSphere Extended Deployment for z/OS manages mixed transactional and long-running workloads within the same dynamic cluster

These scenarios are presented using sample applications that are deployed in a WebSphere Extended Deployment for z/OS environment configured to support mixed workloads. The white paper WebSphere for z/OS Extended Deployment XD - Building the Basic Infrastructure, WP100735, available at the following Web site, provides a well-documented description of the procedures for installing and configuring a WebSphere Extended Deployment environment on z/OS.

http://w3-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100735

We recommend that you review this resource before deploying and experimenting with the scenarios presented in the book. This book follows the same approach used in the white paper to guide you through configuring WebSphere Extended Deployment for z/OS to demonstrate these scenarios.
1.2 Overview

Before describing the expanded WebSphere Extended Deployment for z/OS capabilities in detail, it is worthwhile to briefly review all basic qualities and capabilities Extended Deployment provides for all supported platforms, and to identify those capabilities highlighted in the book.

WebSphere Extended Deployment for z/OS extends the existing WebSphere core function and is easily accessible using the same administration console as WebSphere Application Server for z/OS. With the J2EE™ platform as the core foundation for WebSphere Application Server for z/OS, WebSphere Extended Deployment for z/OS delivers improved qualities around:

- Business flexibility: delivers the core capabilities to more efficiently and effectively support both Java batch workloads and Online Transaction Processing (OLTP), while also enabling centralized management for a heterogeneous IT infrastructure consisting 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 helps to reduce the cost and complexity of managing WebSphere’s IT resources.

- Dynamic operations: enables the J2EE 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 leveraging existing Java skills and resources.

The following sections describe the WebSphere Extended Deployment features that implement these qualities.

**Maintaining predictable application quality of service**

WebSphere Extended Deployment focuses on improving the ability of the application infrastructure to deliver the highest quality environment for Java J2EE applications by enabling dynamic operations. Dynamic operations capabilities help increase responsiveness and flexibility. WebSphere Extended Deployment is designed to deliver dynamic operations through two key capabilities — the virtualization of WebSphere environments and a goals-directed infrastructure.

WebSphere Extended Deployment for z/OS dynamic operations augment the value delivered by the platform integration of WebSphere Application Server for
Features like the z/OS WLM deliver complementary function to WebSphere Extended Deployment's policy-based workload prioritization. Service policy is the application of business performance goals to WebSphere workloads using WebSphere Extended Deployment. Policies can be applied to workloads based upon time to complete, importance, and other variables. While z/OS WLM focuses on an IBM zSeries-wide view of varying workloads, WebSphere Extended Deployment's focus is strictly on the J2EE application execution environment. WebSphere Extended Deployment's goals-directed workload management capability integrates with the z/OS Workload Manager to deliver richer classification schemas at the application layer, while still supporting goals at an enterprise level within the z/OS infrastructure. This capability will be explored in detail in this book.

WebSphere Extended Deployment introduces the On Demand Router (ODR) component, which, in addition to offering the central functionality of the on demand functionality, also provides intelligent workload routing capability. The ODR supports the routing of work to application servers within separate LPARs based upon capability, as opposed to a round-robin approach.

**Combined transactional and long-running workloads**

Transactional and long-running workloads can be combined in WebSphere Extended Deployment for z/OS to deliver more efficient use of application infrastructure resources. This is a differentiating feature of WebSphere Extended Deployment on the z/OS platform.

WebSphere Application Server has traditionally been an On Line Batch Processing (OLTP) system for processing transactional workloads. WebSphere Extended Deployment for z/OS expands on the ability of WebSphere on z/OS to run mixed workloads. WebSphere Extended Deployment supports a new application type and workload pattern referred to as long-running applications or workloads. Now both transactional and long-running workloads can easily be managed within the same WebSphere infrastructure.

In contrast to the real-time, request/response behavior of typical transactional WebSphere applications, long-running workloads require dedicated computing resources for longer periods of time. Traditionally, the processing of long-running workloads had to either occur in a separate, isolated environment, or execute 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 long-running workloads resulted in a costly duplication of resources, with limited ability to share resources during the same periods.

WebSphere Extended Deployment supports two general types of long-running workloads in a J2EE application server: batch and compute-intensive. With
WebSphere Extended Deployment for z/OS, however, these workloads can be dynamically balanced with transactional workloads. This z/OS capability is illustrated with the sample scenarios described in this book.

- Batch workloads involve the periodic processing of a large number of non-interactive transactions against a resource. Common types of Java batch processes include statement processing, payroll processing, database loading, and security log processing. In the batch processing model, transactions are stored and then executed in sequence to optimize processing throughput and efficiency. Often, traditional batch workloads are platform-specific, such as z/OS batch jobs accessing native z/OS data sets or VSAM files.

WebSphere Extended Deployment delivers a common set of batch services, such as checkpoint algorithms for transaction management, reducing developer burden and ensuring consistent and non-disruptive record-oriented batch processing of J2EE workloads.

- Compute-intensive workloads require extreme amounts of computing resources, such as CPU, data storage, or I/O to solve complex problems. This type of processing is historically predominant in the scientific communities for applications like weather forecasting, molecular modeling, and flight simulations. More recently, the use of this type of iterative, data-mining analysis has broadened into the business communities for applications such as financial portfolio analysis, computer-aided design (CAD), rules engines, and other applications involving complex what-if scenarios.

A technique called parallelization is often used to speed up the processing of compute-intensive jobs, using algorithms to split or partition the job into independently running parts that can be run in parallel on multiple CPUs.

WebSphere Extended Deployment delivers an infrastructure for the scheduling and processing of compute-intensive workloads, using operational policies to determine where and when to run the jobs within a business grid created by WebSphere Extended Deployment.

WebSphere Extended Deployment for z/OS enables the processing of these long-running J2EE workloads to occur within a shared infrastructure together with transactional workloads without negatively impacting transactional requests. This enables operating efficiencies to be realized when transactional and long-running workloads are consolidated into a single WebSphere environment. A unified environment provides for common development, administration, and management models, further decreasing costs and increasing operational stability.

The new long-running execution environment has often been referred to as J2EE batch, and this has led to the inevitable comparisons to traditional z/OS batch processing. Even though there are many similarities between Extended
Development's long-running support and a traditional batch processing system, the long-running environment should not be considered as equivalent to or a replacement for a traditional batch environment. This will be addressed in more detail in this book.

**Interruption-free application deployments**

Many distributed WebSphere installations are characterized by a large number of application server instances and frequent application updates. Many z/OS WebSphere installations are characterized by strong governance policies and enforcement of service level agreements (SLAs). Simplifying the change management process while maintaining service policies is a very attractive goal for all WebSphere customers for a variety of reasons.

WebSphere Extended Deployment V6.0.1 extends the concept of production and test environment management to include support for not only multiple applications, but also multiple versions of the same application. Deploying a new version of an application into production can present a challenge that, if not done properly, can lead to loss of service. This situation is exacerbated as the number of applications or the frequency of changes increases. In order to maintain continuous operations, administrators are forced to create duplicate staging environments and elaborate custom-developed routing logic to transition from one application version to another. WebSphere Extended Deployment delivers standard administrative and workload management support for application versions through the new feature *Application Edition Manager*. This enables interruption-free deployment and management of application versions. This capability enables application updates to be applied to a production environment with no downtime. The Application Edition Manager also delivers an application versioning model that supports multiple deployments of the same application in a WebSphere cell, each distinguished by a unique edition number. The Application Edition Manager makes it possible to choose which edition to activate on a WebSphere cluster, enabling new application updates to be rolled in for deployment or backed out to revert to a previous version. It is also possible to keep multiple versions of the same applications in production at the same time, each receiving some portion of the total requests according to specified routing rules.

**Optimize data access for improved application performance**

As the use of transactional Web applications increases, an application infrastructure must scale, on demand, to meet the higher request volumes. Although WebSphere Application Server for z/OS currently scales extremely well, certain application characteristics benefit from the scalability provided by WebSphere Extended Deployment for z/OS:

- An *object grid* is a client/server infrastructure that delivers a customizable, plugable object-caching framework for the Java environment created by
WebSphere Application Server for z/OS, and extended by WebSphere Extended Deployment. An object grid allows applications to share data using a variety of consistency models. Objects can be stored in the grid and then accessed from multiple applications, reducing the number of trips to the data source and avoiding the cost of repeatedly recreating objects. The object grid enables quick and easy sharing of object data across application servers. The object grid delivered with WebSphere Extended Deployment for z/OS V6.0.1 supports the client environment, enabling it to be a fully functioning member of an object grid client/server environment.

The partitioning facility supports a partitioned, asymmetric clustering model for highly scalable and reliable transaction processing. Traditional application server clustering, or symmetric clustering, works well for applications that can run in a stateless cluster in front of a database. However, for high write-rate applications or data-intensive applications that require sequenced request handling, a stateless cluster can only scale so far before cluster members will start competing for access to the data source. WebSphere Extended Deployment delivers a partitioning model that enables asymmetric clustering, an approach that enables intelligent request routing based on defined application partitions, resulting in dramatically improved caching rates and transaction performance.

These features offered by WebSphere Extended Deployment decrease contention for data sources, particularly in a shared environment, and dramatically improved application performance and scalability. This allows data access performance to scale along with the rest of the application infrastructure to keep up with growing demand.

**Improved manageability of heterogeneous environments**

The manageability of heterogeneous environments is improved through real-time insight into utilization and health. It is often difficult to visualize and manage complex, distributed IT environments where multiple applications are deployed on many application servers. It can also be difficult to monitor and maintain many applications across multiple logical partitions (LPARs) within a zSeries infrastructure. While the WebSphere Application Server for z/OS administration console delivers excellent built-in capabilities, the special needs of very complex deployments require an aggregated, meaningful view of the application runtime environment.

WebSphere Extended Deployment for z/OS extends the WebSphere administration console to allow operators to visualize at a glance what is happening in the infrastructure and the relative health of the application resources. The visualization features also enhance the existing console capability by charting application performance against business goals. In addition, alerts send notifications when intervention is required in order to
maintain business goals successfully and to help decrease human-intensive monitoring and management.

Visualization features delivered by WebSphere Extended Deployment include:

- **Aggregated view** of application runtime environment
  - *Tree map* delivers a visual summary view of the entire environment, a starting point that can be used to drill down to more specialized views.
  - *Runtime operations* view topology shows what applications are running where there are relationships to other WebSphere artifacts.

- **Health monitoring** - Health policies intelligently detect memory leaks, excessive memory, excessive response times, stuck requests, storm drain situations, and high error rates.

- **Operations view** offers customizable charting, providing a graphical representation of application performance versus goals.

- **Events** alert operations of areas that are in progress of change or require action, with direct links to charting and other, more detailed views.

- **Autonomic options** - three modes of operation: manual, supervised, and automatic.

- **Capacity metrics** assist organizations’ understanding and measure the consumption of IT resources in a virtualized, grid environment, enabling usage reporting and charge backs.

**Mixed-server environment support**

Many application infrastructures are heterogeneous, including multiple operating systems and various types of application servers or Web servers. WebSphere Extended Deployment eases the management and monitoring challenges presented by these mixed environments.

Mixed-server environment support enables the *WebSphere Extended Deployment for Mixed Server Environments V6.0.1* key capability to be deployed within heterogeneous server deployments. WebSphere Extended Deployment for Mixed Server Environments V6.0.1 is a component that can be ordered separately, and provides an operational monitoring agent, called the Remote Agent, which can be used to feed CPU utilization and health information for both WebSphere and non-WebSphere servers on distributed platforms back to the WebSphere Extended Deployment ODR component. This enables a unified workload management and visual performance monitoring approach for mixed-server environments.

A full WebSphere Extended Deployment license should be purchased for each distributed processor on which the On Demand Router component is installed. A
WebSphere Extended Deployment for Mixed Server Environment license should be purchased for each processor on which the Remote Agent component is installed. This is important for environments in which z/OS is the centralized platform, but management is required across multiple distributed platforms.
Concepts

In this chapter we introduce the functions provided by the On Demand Router and long-running execution environment.
2.1 Service policies and the On Demand Router

In this section we discuss service policies and the On Demand Router.

2.1.1 Service policies

A service policy is used by WebSphere Extended Deployment to categorize and prioritize work requests. Through service policies, WebSphere Extended Deployment introduces the capability to designate the performance goal and business importance of different request types. This allows an enterprise’s performance to degrade in a controlled manner in periods of overutilization. Service policies are assigned response time targets that are valid for specified throughput conditions. The performance management done by the autonomic request flow manager, the dynamic workflow manager, and the application placement controller achieves a defined balance of the performance results.

Service policy definitions are made up of two key items:

- **Goal**
  
The goal portion of the service policy defines how incoming work is evaluated and managed in order to ensure that and detect whether work is meeting its assigned service policy levels. Service policies can have four different kinds of goals: discretionary, average response time, response time percentile, or queue time. These are explained in detail in “Service goal types” on page 98.

- **Importance**
  
  Importance is used in times of resource contention to identify the most important work in the system and give it higher priority. The options for importance vary from lowest to highest.

While WebSphere Extended Deployment has made this capability available on all WebSphere Application Server platforms, there is a special relationship between XD’s implementation and the long-established WLM capability present on z/OS. ODRs running on any platform will delegate goal-based workload management to z/OS WLM when the target clustered servers reside on z/OS regardless of the platform on which the ODR itself is running. Because of this, some aspects of the WebSphere Extended Deployment service policy constructs described below are not used for goal-based workload management when working with z/OS resident clusters. Defining these at the XD level still has value in that they are used for monitoring and reporting purposes. When working with z/OS in this way, complementary WebSphere Extended Deployment service policies and z/OS WLM service classes must be defined.
The remainder of this section discusses WebSphere Extended Deployment's definitions while pointing out a few z/OS considerations. A more detailed discussion of this relationship is in 2.1.3, “ODR and z/OS WLM integration” on page 18.

**Service goal types**

There are four performance goal types in IBM WebSphere Extended Deployment V6.0:

- **Discretionary**: indicates work that does not have significant value. Requests are processed when no higher request is waiting. As a result, work of this type can see a degradation in performance when resources are constrained. This is the default service goal.

- **Average response time**: This goal indicates work with a higher priority than discretionary work. You can specify the average response time in milliseconds or seconds. The average response time has a target percentage of 90%. If you must change this percentage then you need to select percentile response time.

- **Percentile response time**: This goal also indicates higher priority than discretionary work. You can specify both the percentage and the average response time. For example, 95% of all requests should be answered below 1000 milliseconds.

- **Queue wait time**: This is a service goal for long-running applications. When the goal reaches this limit, more servers are needed.

**Importance**

Creating levels of importance guarantees that in the event the performance goals for all service policies cannot be satisfied because of prolonged intense overload, WebSphere Extended Deployment can use importance to decide which service policy takes priority. You can select from seven importance levels:

- **Lowest**
- **Lower**
- **Low**
- **Medium**
- **High**
- **Higher**
- **Highest**

Some planning is essential to select the right importance value, because negative results can occur if all work is rated as highest. Such a rating can create a bottleneck within the environment. A better approach might be to leave all your applications with a discretionary goal, to assign a higher goal to the important applications, and to use importance levels only if you need further differentiation.
between the higher goal applications. While this approach is reasonable for WebSphere Extended Deployment in general, it should not be taken when defining importance levels in z/OS WLM. See 2.1.3, “ODR and z/OS WLM integration” on page 18, for more detailed discussion.

**Transaction classes**

Service policies contain one or more transaction class definitions. The service policy creates the goal, while the transaction and work classes are used to connect Universal Resource Identifiers (URIs) to that goal. When working with z/OS resident applications, the goal defined in the service policy is only used for monitoring and reporting rather than active workload management. The transaction also serves the purpose of providing the TCLASS value that is propagated to with the request for utilization by z/OS WLM. The classification accomplished using the work class is applicable regardless of the target platform and represents one of the valuable contributions of ODR to a z/OS topology.

Transaction classes are a subcontainer of the service policy for work being classified into the service policy that can be used for finer-grained monitoring. They can also be used as a mechanism of grouping across application work together for common monitoring.

The relationship between service policies and transaction classes is one to many: A single service policy can have multiple transaction class definitions but each transaction class belongs to exactly one service policy.

Every service policy has a default transaction class, which in most scenarios is sufficient. Additional transaction classes are created when finer-grained monitoring is necessary for the environment. Each transaction class name must be unique within the cell.

### 2.1.2 On Demand Router

The On Demand Router (ODR) is a component located in front of a WebSphere Extended Deployment environment. The ODR is a component that logically replaces and extends the functionality of the WebSphere Web server plug-in. It represents the entry point into your application server environment. Specifically, the ODR is responsible for classification, workload balancing, and dispatching of requests in accordance with your operational policy (service policy and health policy).

It also does version (edition) aware routing, handles the routing that supports the storm drain health monitoring policy, and application lazy start, whereby it can serve an application request, even for applications that are not currently running, by holding the request as it dynamically starts up the application, and then forwards the request to the application once it is started.
z/OS customers have many HTTP routing choices. Refer to Table 2-1 for an overview of the HTTP routing choices. Setting up the WebServer with a WAS plug-in is a common solution, and it is available for all WAS platforms. Importantly, it understands HTTP session affinity and routes accordingly.

Table 2-1 HTTP routing choices

<table>
<thead>
<tr>
<th></th>
<th>Session affinity</th>
<th>Dynamic workload balancing</th>
<th>Automatic capacity management</th>
<th>Storm drain avoidance</th>
<th>Application lazy start</th>
<th>Application edition routing</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebServer with plug-in</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Sysplex Distributor</td>
<td>X</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>WebServer with plug-in and Sysplex Distributor</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>On Demand Router</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

The z/OS sysplex distributor uses cluster addresses to dynamically balance workload across the Sysplex LPARs. However, being part of the TCPIP stack, it does not understand HTTP sessions, so you can only use Sysplex Distributor for stateless HTTP applications. Fortunately, you can combine WebServer, the plug-in with sysplex distributor cluster addresses to get the best of both worlds.

The ODR provides the same function of combining the WebServer with plug-in and the sysplex distributor as well as delivering these additional features:

- **Classification**
  
  XD is able to classify HTTP traffic, and this enables the ODR to pre-select the z/OS WLM service policy and propagate the selection to the platform on the work request.

- **Capacity management**
  
  It has the ability to start or stop application servers to manage workload demands based on available service unit metric feedback from the platform to enable the ODR to make informed decisions.
- Storm drain avoidance
  The ODR will detect for each cluster member a significant drop in the average response time, coupled with changes to the dynamic workload manager weights for the cluster member. This helps to prevent the system from shifting all of the load to a faulty server with low response time.

- Application lazy start
  XD can serve an application even for applications that are not currently running, by holding the request as it dynamically starts up the application, and then forwarding the request to the application once it is started.

- Version-aware application routing
  This feature provides for interruption-free application update. This is a basic requirement for production application deployments. It relies on the ODR to control HTTP traffic to the application servers while the new edition replaces the old, one application server at a time. Refer to Figure 2-1. This ensures that the necessary quiesce and resume behavior is carried out such that no application requests land on application server instances temporarily incapable of servicing the request.

![Figure 2-1  Application routing via ODRs](image)
This also allows for partitioning of the request stream so that selected requests can be routed to a targeted edition. This allows multiple operational scenarios, including incremental branch rollout and piloting. Refer to Figure 2-2. XD exploits this capability to support a validation mode for application rollout, whereby XD creates a clone of the environment on which an application is deployed and allows selective routing to that clone for validation purposes. Once the application is validated, the edition may be rolled out and the clone shut down and optionally deleted.

Figure 2-2  Validation of application with cloning
2.1.3 ODR and z/OS WLM integration

The ODR brings a unique value-add to the z/OS platform with its integration with z/OS WLM. It is two-way integration, as described below and illustrated in Figure 2-3:

- Available service unit metric feedback from z/OS WLM to enable the ODR to make informed decisions on workload placement
- Leverage of XD’s classification engine to assign and propagate the transaction class that WAS XD will use to register the work request with z/OS and select the z/OS service class that governs its execution

![Figure 2-3  Overview of ODR and z/OS WLM](image)

**XD and WLM transaction classification**

One of the ODR’s roles is to accept incoming requests and dispatch those requests to a WebSphere Extended Deployment backend. If the target is a WebSphere Application Server residing on z/OS, then the request will be immediately dispatched and routed according to available cluster member weights to accomplish workload balancing. On z/OS request queueing and goal-based workload management is performed by z/OS WLM. If the target does not reside on z/OS then request queueing and goal-based dispatching is done in the ODR.
For example, consider the handling of requests for a hypothetical stock application (as shown in Figure 2-3 on page 18). Requests for different application modules will arrive from the WebServer tier to the ODR. Each module has its own service policy. When a request arrives, the ODR classifies it and assigns a TCLASS for the requests by adding it to the HTTP Transport header.

Since the target cluster is running on z/OS, the requests are not queued or prioritized in the ODR but are dispatched immediately according to applicable routing policies and the dynamic weights that have been calculated for the various cluster members. The weights vary between 1 and 20 and represent the maximum number of outstanding requests the ODR will allow to be dispatched to the associated cluster member. Clusters running on nodes for which z/WLM metrics indicate very high utilization will receive lower weights, while those with lower utilization will retain higher weights.

When the request reaches the control region, the TCLASS is extracted from the request header and used to associate the request with an z/OS WLM service class. An enclave is created having the indicated service class and it is dispatched via WLM to a servant region where the Web Application code is executing. Queueing and prioritization to achieve service class goals is done by z/OS WLM at this point. In addition, z/OS WLM also isolates requests having various service classes to different servant regions.
Service policies in WebSphere XD and z/OS WLM

As seen in Figure 2-5, a WebSphere XD service policy has seven importance levels with four goal types, while in z/OS, a WLM service class has five importance levels and four goal types. One consideration when mapping XD service policies to z/OS WLM service policies is that there is less granularity in the importance levels (seven levels compared to five). The importance levels on z/OS WLM also serve a larger role on z/OS. Unlike the importance levels in WebSphere XD, which only represent importance relative to other XD work, z/OS WLM importance is used for service classes for all workloads running in z/OS including other software, system services, traditional batch work, and discretionary services. On z/OS one must be aware of the other service classes defined in the z/OS environment.

In setting up your service classes on z/OS WLM, there are two objectives that you want to achieve:

- It is necessary to distinguish different work and to define service classes in a way that different groups of end users can get the service they need from the operating system.
You should keep your service classes to a minimum. Every 10 seconds WLM examines all service classes to determine if any need help. If there are too many service classes it is possible that WLM needs too many adjustment cycles to really calculate all possibilities to help lower important work.

The results of these requirements tell us that on the one hand we should define service classes to distinguish all types of work and users from each other, and on the other hand to restrict ourselves to help WLM to work efficiently. Here are some guidelines to help us to satisfy both requirements:

- Service classes should only be defined for work when sufficient demand exists for it in the system. The demand can be measured either by the service consumption or by the amount of ending transactions during high utilization periods of this work. Based on this it does not make sense to define service classes with a demand of less than 1% of measured service units at any period or less than 1 transaction ending per minute. In such cases it is usually better to combine this work with other work of the same type.

- It is not meaningful to create service classes for the same type of work with identical service objectives. For example, if you define started tasks for different applications with exactly the same goals and importance, you can combine them in the same service class and use report classes to differentiate them for reporting purposes.

- Do not combine diverse types of work into one service class. For example, do not combine transaction and address space into one service class. SRM sampling data, plots, and projections can become distorted when this is done.

For a more information about workload classification on z/OS WLM, please refer to Chapters 4 and 8 in IBM Redbook *System Programmer's Guide to: Workload Manager*, SG24-6472.

### 2.1.4 Shared workload management

Shared workload management is an additional z/OS-specific capability. On distributed platforms, XD manages workload by dedicating each node to a particular task, such as processing J2EE batch or OLTP. In that environment, each node is typically a separate system and the resource management operates with the default presumption that the entire system is dedicated to executing WebSphere applications for a single cell.

In contrast, the z/OS system is inherently designed and used to manage large, diverse, mixed workloads and manage those workloads with its built-in sysplex-aware workload manager.

With XD on z/OS, it morphs and adjusts its resource management style to cooperate with the z/OS platform workload manager, allowing XD nodes to host
mixed J2EE batch and OLTP along with whatever other work the platform may be hosting. This way, XD conforms to and supports the normal operational and management approach used to host workloads on z/OS. Refer to Figure 2-6.

Figure 2-6  Shared workload management

2.2 The long-running environment

This section introduces you to the WebSphere Extended Deployment long-running environment and the long-running application programming model. Later in this book we illustrate how to configure the WebSphere Extended Deployment on the z/OS long-running environment, how to install long-running workloads into this environment, and how to schedule and manage long-running workloads.

2.2.1 Overview

As mentioned previously, long-running workloads or applications typically require more resources and different types of support from a runtime environment than the standard lightweight, transactional work that is typical of today’s J2EE applications. WebSphere Extended Deployment V6.0.2 introduces a facility to the J2EE environment for supporting these long-running applications. The term business grid is often used in the WebSphere Extended Deployment literature to describe this environment and overall support required for long-running applications. The business grid provides the capability to deploy different types of
applications to different nodes within a WebSphere cell, and to balance the work based on policy information.

The submission of a long-running workload (that is, a job) is asynchronous to the workload being executed. The separation of the submission and execution environments also allows for the submission of work from outside the WebSphere environment. Once long-running work has begun, state information must be persisted to a highly available data store. Administrators also require the ability to monitor and manage long-running work. The environment must be able to schedule and prioritize the work based on policy information set by the user. A number of these functions can be provided within an existing WebSphere J2EE environment by utilizing asynchronous and message-driven beans (MDBs). Some WebSphere customers have successfully exploited this. However, the business grid provides an enhanced environment for long-running applications, making it easier to support this type of work within WebSphere.

The business grid components support two types of long-running workloads or applications, **batch** and **compute-intensive**. A typical batch application does large amounts of work based on repetitive tasks. A batch application must provide the logic for a single unit of work, and the container provides the support to run the job with transactions and the ability to checkpoint and restart the application as required. For example, a typical batch application would process a large number of records. Each record can represent a unit of work. The application provides the logic to process a single record. The environment manages the process of repeatedly invoking the application’s task for processing each record until complete, and performing the appropriate transaction when required.

Compute-intensive applications perform work that requires large amounts of system resources, in particular, CPU and memory. In this case the application provides all of the logic for performing the work, including acquiring the resources. The business grid makes sure that the application is appropriately situated within the environment.
### 2.2.2 Long-running environment components

This section is a high-level overview of the main components involved in the long-running environment or business grid. Each of these components is explained in more detail in following sections.

![Long-running environment component architecture](image)

**Execution environment**

The execution environment provides the runtime resources required by the long-running applications. Long-running applications are deployed into WebSphere Extended Deployment dynamic clusters that have been configured to support long-running execution. The services to enable WebSphere Extended Deployment to host long-running applications are contained in a J2EE application provided by WebSphere Extended Deployment called LREE.ear. This application is installed into a dynamic cluster. A dynamic cluster configured to support long-running applications is referred to as the long-running execution environment (LREE). A database gives the execution environment a transactional datastore that it can use to track job status, do checkpoints, and recover from failures.

Different long-running execution environment instances (that is, LREE-configured dynamic clusters) can be configured in a cell to support different types or styles of long-running work. A particular instance of the execution environment can have its own dedicated database, or it can share a database with one or more other instances of the execution environment. The process of configuring a long-running execution environment is described in...
The WebSphere Extended Deployment V6.0.2 LREE supports two types of execution environments for the two types of long-running workloads:

- **The compute-intensive execution environment** supports long-running applications that expect to consume large amounts of memory or CPU time. This execution environment provides a relatively simple programming model based on asynchronous beans.

- **The batch execution environment** supports long-running, batch-oriented applications. These applications are expected to repetitively perform record-processing units of work similar to more traditional J2EE applications, but are driven by batch inputs rather than interactive users. This environment builds on familiar J2EE constructs, such as entity beans, to provide batch applications with a programming model that supports container-managed restartable processing and the ability to pause and cancel executing jobs.

As with the WebSphere Extended Deployment dynamic operations function, the business grid has autonomic management functions to dynamically adapt the long-running environment to changing workloads. The business grid provides the following functions:

- **Long-running placement:** This autonomic controller is analogous to the application placement controller (APC) that starts and stops instances of transactional applications. The long-running placement controller starts and stops instances of long-running execution environments in response to the jobs in the system.

- **Balancer:** Due to the nature of long-running work, co-locating it on the same system with transactional work may have a negative impact on the transactional work being performed. The balancer makes decisions about when and where transactional and long-running work should run. These decisions are based on a number of factors including how well the service policy goals for the two types of work are being met. z/OS is specifically equipped to handle these mixed and diverse workloads. The balancer is able to share a z/OS system with the transactional work request manager. This scenario is demonstrated in *Using WebSphere Extended Deployment V6.0 To Build an On Demand Production Environment*, SG24-7153.
Long-running scheduler
The long-running scheduler (LRS) is responsible for accepting, persisting, and scheduling the execution of long-running jobs. It manages the job database, assigns job IDs, and selects where and when jobs should be run. The LRS has two key components:

- Job dispatcher: The job dispatcher accepts job submissions, assigns job IDs, persists jobs in the job database, and sends jobs to execution environments.
- Endpoint selector: The endpoint selector uses policy to select which jobs to run where and when.

The scheduling services are deployed in another dynamic cluster configured with the resources required by the LRS. The LRS services are provided by a J2EE application, LongRunningScheduler.ear, installed into this dynamic cluster. The LRS J2EE application can be made highly available by using APC-provided functionality. This capability is described in Using WebSphere Extended Deployment V6.0 To Build an On Demand Production Environment, SG24-7153.

Scheduler interface - submitting long-running jobs
As illustrated in 2.2.2, “Long-running environment components” on page 24, there are several methods to access the LRS to submit long-running workloads or jobs. The following methods are used with the samples provided with this book:

- Command line: A command-line interface is provided to interact with the long-running scheduler to submit and manipulate long-running jobs. The command is located at was_root/bin as the lrcmd.sh or lrcmd.bat script and can be invoked from any location in the WebSphere cell.
- EJB™ Interface: The LRS provides an Enterprise JavaBean (EJB) interface for J2EE applications to programmatically submit and manipulate long-running jobs.
- Web Services Interface: The LRS also supports programmatic access to its functions over a Web service interface for both J2EE and non-J2EE applications. This also enables jobs to be scheduled from outside a WebSphere environment.

Defining long-running jobs - xJCL
Central to all long-running applications is the concept of a job to represent an individual unit of work to be executed. Long-running jobs are defined using an Extensible Markup Language (XML) dialect called XML Job Control Language (xJCL). xJCL has constructs for expressing all of the information needed for both types of long-running workloads (jobs) (that is, compute-intensive and batch), although some elements of xJCL are only applicable to either compute-intensive jobs or batch-type jobs.
Several xJCL samples with annotations are provided with the additional materials for this book. Refer to WebSphere Extended Deployment online Infocenter for more detailed information about defining xJCL:

http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r0/index.jsp

### 2.2.3 Anatomy of a long-running batch application

This section explores the process flow and programming model for long-running batch applications. We focus on the batch job flow and programming model since it is thoroughly explored in this book to demonstrate several unique WebSphere Extended Deployment for z/OS capabilities. Unless otherwise indicated, the shorter terms *batch*, *batch job*, or *batch application* are used interchangeably to refer to a long-running batch application within WebSphere Extended Deployment.

At this time it should be mentioned, again, that long-running batch is different from traditional z/OS batch. Long-running batch applications are J2EE applications that execute inside a WebSphere Application Server process (LREE configured application server) and have access to all of the WebSphere container facilities. In contrast, a z/OS batch application is a conventional program written in any of a variety of languages (COBOL, PL/I, and so on, even Java). Note, however, that Java programs in z/OS batch are Java mains, not J2EE components, and more importantly z/OS batch jobs execute in JES Initiator address spaces. There is no direct access from a z/OS batch program to WebSphere container facilities and management functions (for example, communications, security, transactions, and so on).

Long-running batch applications in WebSphere Extended Deployment V6.0.2 are Enterprise Java Beans (EJBs) based Java 2 Platform Enterprise Edition (J2EE) applications. These applications must conform to a few well-defined interfaces that allow the batch execution environment to manage the execution of batch jobs.
Flow
Now we look at how all the previously mentioned long-running environment components work together to process a batch application. The flow is illustrated in Figure 2-8.

The flow is:
1. The xJCL describing all the information needed to run the long-running job is created.
2. The xJCL is submitted to the LRS via command line, EJB API, Web services API, or administrator console.
3. The LRS job dispatcher accepts job submissions, assigns job IDs, persists jobs in the job database, and sends jobs to the appropriate execution environment (that is, the execution environment where the long-running application is installed). (Remember, there can be more than one LREE dynamic cluster.) The LRS Endpoint Selector uses policy to select which jobs to run, where, and when.
4. The LREE pulls the detailed job information from the database and proceeds to execute the job, either batch or compute-intensive, as described in the xJCL. The next section looks at the execution flow for a batch job in more detail (that is, the batch job programming model).
5. The LREE keeps job state information in the database and can handle checkpoints and restarts as required.

A batch job is comprised of one or more job steps. Dividing a batch application into steps allows for separation of distinct tasks in a batch application. The output stream from one batch step can be an input stream to another batch step.
(See “Flow” on page 28.) A typical flow for a batch step would be:

1. Read input data (one or more records) from an input stream.
2. Perform the appropriate batch logic (business service) required by the input data.
3. Write output data to an output stream if required.
4. Commit work done by batch logic to a database (checkpoint).
5. Loop to get next records.

For J2EE applications, the container handles the application control flow and provides basic services (such as checkpoint). The application provides specific business logic to implement its functional requirements. Next we look at the components within a long-running batch job.

**Job state**

As the long-running scheduler and execution environment process a long-running batch job, the job’s state is updated in the long-running scheduler's database. Figure 2-9 shows the relationship between batch job states and the events that trigger transitions between those states.

---

*Figure 2-9  Long-running batch job states*
You can view the current state of a batch job on the administrative console’s Job Management panel, or retrieve it using the command line, Enterprise JavaBean (EJB), or Web service interfaces to the long-running scheduler. If a failure occurs before a batch step initializes, the batch job goes into execution failed state. Otherwise, it goes into restartable state.

In Chapter 7, “Long-running batch workloads” on page 137, we present a sample application that illustrates this flow.

**Batch application components**

Each batch job step is composed of the following components (see Figure 2-10):

- **Batch data streams:** A batch step can have zero or more input or output batch data streams associated with it. A batch data stream (BDS) is a Java class that implements the com.ibm.websphere.batch.BatchDataStream interface to support the reading and writing of data to the implemented stream.

- **Batch step CMP entity bean:** Each job step is represented by a CMP Entity EJB that implements BatchJobStepInterface. Using the Entity EJB construct provides job step state persistence and a transaction context for the job step. The job ID and step ID are persisted for each batch job step.

![Figure 2-10 Batch application components](image)

Batch application components are developed by the J2EE application programmer. The following sections provide more detail for these components.
**batch data streams**

A batch data streams (BDS) object implements the com.ibm.websphere.batch.BatchDataStream interface. This interface is backend agnostic — that is, the implementing object can retrieve data from any type of data source (for example, files located in the HFS, z/OS data sets, relational databases, and so on). Callback methods on the BatchDataStream interface allow the LREE to manage the BDS at run time. One of the key features of a BDS is the capability to convey its current position in the data stream to the LREE, and the capability to position itself to a given location in the data stream. This feature allows the LREE to record (in the LREE database) how much data a batch step has processed at a given point. This information is recorded on every checkpoint, thereby enabling the LREE to restart a batch job from a recorded position in the data stream in case the job is suspended, cancelled, or fails in a recoverable manner.

The significant methods on the BatchDataStream interface are listed here for your information. The sample batch application provided with this book has a trace feature that you can optionally set to see when these various interfaces are invoked. Refer to the WebSphere Extended Deployment on-line Infocenter for more information about the BatchDataStream interface:

http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r0/index.jsp

The significant methods on the BatchDataStream interface are:

- void open() - called by LREE to open the BDS.
- void close() - called by LREE to close the BDS.
- void initialize(String ilogicalname, String ijobstepid) - called by LREE to initialize the BDS and let it know its logical name and batch step ID.
- String externalizeCheckpointInformation() - called by LREE right before a checkpoint to record the current cursor of the BDS.
- void internalizeCheckpointInformation(String chkpointInfo) - called by LREE to inform the BDS of a previously recorded cursor (chkpointInfo). Usually the positionAtCurrentCheckpoint is called after this call to position the BDS to this cursor.
- void positionAtCurrentCheckpoint() - called by LREE after calling internalizeCheckpointInformation to position the BDS to the cursor indicated by the chkpointInfo passed in through the internalizeCheckpointInformation call.

The BatchDataStream interface does not implement methods for reading or writing data. For example, there are no getNextRecord() and putNextRecord() methods implemented on the interface that a batch step would call to read or write to the BDS. Methods for passing data back and forth between the batch
step and the BDS are optional and left up to the implementation of the BDS object. Look at the batch samples shipped with WebSphere Extended Deployment and samples provided with this book to see examples of how to implement batch data streams.

It is important that the physical resource representing the batch data streams be located on a shared resource if you deploy your batch application in a LREE dynamic cluster with more than one application server instance. The batch job can be scheduled to run on any of the LREE application server instances and needs access to the batch data stream. Also, if a job was suspended it could be restarted on an LREE application server instance other than where it was initially started.

**Batch step entity CMP bean**

Distinct tasks of a batch application can be divided into batch steps. Each batch step is implemented as a local container-managed Enterprise Java Bean (EJB) that specifies the BatchJobStepLocalInterface as its business interface. Callback methods in the BatchJobStepLocalInterface allow the long-running execution environment (LREE) to invoke batch steps when it runs a batch job.

A batch step EJB contains the *batchable* business logic to run a portion of the batch job. Typically, a batch step EJB contains code to invoke the input BDS method to read a record from a batch data stream (or write a record to an output stream). The batch step EJB then executes business logic required by the record. The processJobStep() method of a batch step EJB is called by the LREE in a batch loop. This method should contain all the business logic necessary to process an input record. The business logic may subsequently call an output BDS method to write an output record that will serve as input to the next step. LREE calls the processJobStep() method in a loop until all the input or output records are processed or the batch step informs LREE to stop processing.

The LREE invokes batch step EJB methods under a global transaction. This global transaction is managed by WebSphere. The behavior of the transaction, such as transaction time out or transaction commit interval, is controlled by the checkpoint algorithm associated with the batch job to which the step belongs.

The important methods on the BatchJobStepLocalInterface interface are listed here for your information. The sample batch application provided with this book has a trace feature you can optionally set to see when these various interfaces are invoked. Refer to the WebSphere Extended Deployment on-line Infocenter for more information about the BatchJobStepLocalInterface:

http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r0/index.jsp

The important methods on the BatchJobStepLocalInterface interface are
**setProperties(java.util.Properties properties)** - makes properties defined in xJCL available to the batch step in a java.util.Properties object. This method will be invoked under a global transaction.

**void createJobStep()** - indicates to the step that it has been initialized. Initialization logic, such as retrieving a handle to a batch data stream, can be placed here. This method is invoked as a global transaction.

**int processJobStep()** - invoked by LREE in a batch loop until the return code integer of this method indicates that the step has finished processing. Look at the BatchConstants in the batch API to view the possible return code values. The BatchConstants.STEP_CONTINUE return code indicates to the LREE that it should continue to call this method in the batch loop. The BatchConstants.STEP_COMPLETE return code indicates to the LREE that the step has finished, and it should now call the destroyJobStep() method.

**int destroyJobStep()** - indicates that the job step has completed. The integer return code of this method is arbitrary and can be chosen by the batch application developer. The return code is saved in the LREE database and represents the return code for the batch step. If a results algorithm is associated with the batch job, the return code is passed to it. If return-code-based conditional logic is contained in the xJCL of the batch job, the LREE will use this return code to evaluate that logic.

**Note:** A getProperties() method on the BatchJobStepLocalInterface is not currently called by the LREE. It is included in the interface for symmetry and possible future use.

**Batch controller bean**
In the long-running batch application deployment descriptor, a special stateless session bean is declared. This bean acts as the overall batch job controller for the LREE and must contain the enterprise bean references to all the batch step enterprise beans used in the batch application. The implementation of this bean is provided by WebSphere, not by the batch application. The batch job developer only needs to declare the controller bean in the batch application's EJB deployment descriptor. Only one controller bean is defined per batch application.

**Checkpoint algorithms**
The LREE uses checkpoint algorithms to decide how often to commit global transactions for each batch step. Two checkpoint algorithms are shipped with WebSphere Extended Deployment V6.0.2: a *time-based algorithm* and a *record-based algorithm*. Callback methods contained within the BatchDataStream implemented classes are informed of the checkpoint status. The checkpoint algorithms are applied to a batch job through statements in the xJCL. Properties specified for the checkpoint algorithms in the xJCL influence
checkpoint behavior, such as transaction time outs and checkpoint intervals. These can be customized for each job step. WebSphere Extended Deployment also provides a time-based checkpoint algorithm and defines an SPI for building additional custom checkpoint algorithms. On each job step iteration of the EJB processJobStep() method, the LREE consults the checkpoint algorithm applied to that step to determine whether it should commit the global transaction. Callback methods on the checkpoint algorithms allow the LREE to inform the algorithm when a global transaction is committed or started. This enables the algorithm to decide whether the time has come to commit the global transaction.

The time-based checkpoint algorithm commits global transactions at a specified time interval. The record-based checkpoint algorithm commits global transactions at a specified number of iterations of the processJobStep() method of batch step. Each call to the processJobStep() method is treated as iterating through one record.

**Note:** A single call to the processJobStep() method can retrieve multiple records from a batch data stream. However, the checkpoint algorithm regards one record as the equivalent of a single call to the processJobStep() method.

The sample BatchTrade application provided with this book demonstrates the record-based checkpoint algorithm and enables you to experiment with different record interval settings.

**Results algorithm**
Results algorithms are an optional feature of the batch programming model. Results algorithms are applied to batch steps based on their return codes, and the algorithms are used to manipulate the return codes of batch jobs. Additionally, they act as a place holder for triggers based on step return codes.

**Packaging**
As mentioned previously, a long-running application is packaged as a standard J2EE EJB application inside a J2EE EAR file. Batch step entity enterprise beans and batch data stream classes used by the batch steps are packaged with the application. The batch job controller bean must also be declared in the enterprise bean deployment descriptor of every batch application. Standard J2EE development tools can be used to develop and package the batch application EAR. Also included in the J2EE EAR file are other resources required by the batch job such as EJB jars containing business logic, utility jars, and so on.
2.3 Why WebSphere Batch on z/OS

Most of the batch workloads running on z/OS today are written in COBOL. Many of these workloads contain complex business logic such as interest calculations and process a very large number of input data, such as credit card postings and insurance claims, just to name a few. Customers have very tight batch windows to do these operations. Even with the emergence of Java as a popular and powerful programming language, Java has been viewed as not fast enough for traditional batch workloads, and, more importantly, Java does not have access to many of the same z/OS resources of a typical batch job. With today's quickly maturing, feature-rich WebSphere Application Server container server services, Java fits very well into the OLTP paradigm.

Several questions arise relative to the notion of batch applications and Java: Why do I need a Java batch environment in WebSphere? Why do I need another batch environment on z/OS? To answer these and other related questions we must probe the following areas:

- Who will be developing modern batch applications?
- Which run-time environment is best positioned to host Java batch applications?
- What does WebSphere provide to make it an attractive environment to develop and deploy Java batch-style applications?

**Java and J2EE skills**

In today's development environment Java skills are readily available, and in general it is comparatively less expensive to achieve an equivalent level of application development competency with Java than with older languages. The object-oriented nature of the language enables rapid development cycles. There is a large portfolio of powerful Java development tools from ISVs. 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 J2EE 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 J2EE skill base is rapidly growing in the industry. This makes the Java language and J2EE application servers an interesting and obvious target to host other types of enterprise business workloads, such as batch and compute-intensive applications.
The J2EE run-time environment

Today you can deploy Java batch applications on z/OS in a traditional batch-initiator environment. There are, however, some disadvantages. A JVM is created for every batch job. JVM creation is the most expensive operation in the Java world. This increases the CPU usage, consumes additional resources, can decrease the throughput for a given batch window, and hence, increase the overall cost of implementing a Java batch application as opposed to implementing the same application in a more traditional language. You can also deploy a Java batch-style application in a UNIX System Services shell. However, there are no facilities to start, stop, and manage batch work. You must design and develop all of it.

The J2EE environment as implemented by the WebSphere Application Server provides a natural home for deploying enterprise-class business applications written in Java. First, each application server process (or servant region on z/OS) is a JVM, and within that JVM WebSphere provides for the instantiation, management, and termination of the Java business applications. Second, WebSphere provides many of the overall system services (that is, container services) required by complex business applications. Some typical container services are security, transaction support, high-availability features, Web services, session pooling, connection pooling, caching, and message buses, just to name a few. All this leads to drastically simplifying the Java application’s design, accelerates development cycles, simplifies management, and increases the application’s overall security, availability, performance, and stability.

WebSphere Extended Deployment services for batch

WebSphere Extended Deployment builds on top of the existing WebSphere J2EE programming model and container services by providing a job scheduler, an execution environment, and additional features specifically designed to provide for the execution and management of long-running batch applications. Just as WebSphere Application Server Network Deployment provides a more natural environment for hosting Java enterprise applications, specifically OLTP applications, WebSphere Extended Deployment provides a more robust environment for deploying long-running Java applications.

2.3.1 WebSphere Extended Deployment on z/OS Benefits

WebSphere Extended Deployment for z/OS provides the same qualities of service and differentiating characteristics that customers have come to expect from WebSphere Application Server for z/OS. There are several additional factors to consider when deploying long-running workloads in WebSphere Extended Deployment for z/OS. These are discussed in the next sections.
WLM policy and long-running workloads
WLM plays an important role on the z/OS platform. Batch workload is usually run in low priority compared to mission-critical OLTP workload. WAS XD integrates with WLM to separate OLTP and batch workload.

Mixed OLTP transaction and long-running workloads
WebSphere Extended Deployment manages workloads on distributed platforms by dedicating each node to a particular task, such as processing long-running workloads or transactional (OLTP) workloads. In that environment, each node is typically a separate system, and the resource management operates with the default assumption that the entire system is dedicated to executing WebSphere applications for a single cell. In contrast, the z/OS system is designed for large, diverse, mixed workloads, and it manages those workloads with its built-in sysplex-aware workload manager. WebSphere Extended Deployment for z/OS cooperates with the z/OS platform workload manager, thus allowing WebSphere Extended Deployment nodes to properly manage both the long-running and OLTP workloads. In this way, WebSphere Extended Deployment for z/OS conforms to and supports the traditional operational and management approach used for workloads on z/OS.

The batch model
z/OS is the most pervasive environment in the industry for running batch workloads. z/OS customers are intimately familiar with configuring and tuning a batch environment; defining complex batch jobs; and planning, scheduling, and managing batch workloads. The WebSphere Extended Deployment long-running services were modeled on z/OS batch. The JES scheduler (LRS), initiators (LREE), JCL (xJCL), SDSF displays (Job Management Console), and batch programming model (steps and data streams) are all familiar constructs in z/OS. All of this makes WebSphere Extended Deployment for z/OS a natural environment when considering the development of new long-running applications or re-engineering existing batch applications. You can build on your IT’s governance, procedures, and skills for handling a batch environment.

Long-running workloads with JZOS
A typical batch application on z/OS involves reading, writing, creating, and deleting traditional z/OS files and data sets. The Java SDK does not have APIs or libraries for accessing traditional z/OS data sets. IBM initially provided JRIO to fill this gap, but there were some limitations. JZOS is a technology that provides a compact set of Java APIs to access z/OS data sets as well as other functions. JZOS can be exploited by Java applications executing in traditional batch, J2EE
applications in WebSphere, and more importantly, long-running batch applications in the WebSphere Extended Deployment for z/OS long-running environment. More information about JZOS can be found at:

http://www.alphaworks.ibm.com/tech/zosjavabatchtk

2.4 Security

Security is very important to z/OS customers and is one of the main reasons for selecting z/OS for deploying applications and services. The z/OS traditional batch environment provides great flexibility with respect to security. In a z/OS environment, you have the ability to grant specific users the privilege to run jobs and to indicate that a job is to execute under a specified user identity.

The same level of security granularity can be provided in WAS XD by exploiting the J2EE security. Each job in XD is an entity EJB. We can set up EJB roles for the methods in each job step, assign users or groups to these roles, and allow only authorized users the ability to submit jobs. In WAS XD an individual user identity is not propagated, and therefore the long-running job executes under the identity of the servant region, hence, the ID under which the servant region runs must be configured to read, write, or create any files required by the long-running application. In a traditional z/OS environment, if a COBOL program is running as a batch job, it can execute under an individual user ID and only that user ID needs access permissions to the files. It is anticipated that in a future release of WebSphere Extended Deployment a long-running application will execute under the identity of the submitter.
Our environment

In this chapter we provide a description of the WebSphere Application Server on z/OS infrastructure we used during the preparation of this book.
3.1 Infrastructure considerations

For our project, our goal was to demonstrate the features and how to configure WebSphere Extended Deployment (XD) on z/OS for various kinds of workload. We detail the setup and demonstrate each workload in the subsequent chapters.

We have adapted an existing architecture that was used in IBM Redbook, *Architecting High Availability Using WebSphere V6 on z/OS*, SG24-6850:

http://www.redbooks.ibm.com/abstracts/sg246850.html

In this chapter we describe the changes we made to the WebSphere Application Server Network Deployment environment to create our environment.
3.2 The architecture we used

Figure 3-1 illustrates the high-availability architecture that we had before we installed WebSphere Extended Deployment into the systems.
3.2.1 Our front-end systems

Figure 3-2 illustrates the architecture of our front end. We have kept the LPAR and software setup intact. We have the following components installed:

- Network Dispatcher/load balancer
- The IBM HTTP servers
- The WebSphere plug-in

In our front-end configuration, there are a total of four zLinux servers running in the same Virtual Machine (VM) LPAR. On this LPAR, there are two Network Dispatcher systems and two IBM HTTP Server (IHS) systems.

The LPAR configuration is running z/VM® V5.1 connected to real OSA-E. There are a number of network configurations that can be set up, and you must be careful in choosing a setup for which the Network Dispatchers will work properly.

To understand some of the networking technologies that are available for the zSeries, see Networking Overview for Linux on zSeries, REDP-3901.

We decided, for the sake of ease and simplicity, to host all of our machines, both the front end and back end, in the same subnet area. In a production environment where security and performance is a concern, considerations should be taken to separate high-traffic components into different subnets and use proxy/cache servers in various places to serve static or pseudo static content.
Network Dispatcher heartbeat

The Network Dispatcher machines have load balancers from WebSphere Edge components. The backup dispatcher machine will be monitoring the up-time status of the primary dispatcher machine through what is commonly known as a heartbeat. If the backup dispatcher machine detects that the first machine is down, it issues the appropriate command to take over the cluster IP address of the primary server.

Note: IP takeover does not happen in the same way as it does in a regular Ethernet configuration, where a gratuitous arp is issued to the router telling it where to send its packets. Since our OSA-E configuration only allows for Layer 3 transport at this time (not Layer 2, which is traditional), we only had the option of using the goActive/goStandby scripts.

Isolating heartbeat

As shown in Figure 3-2 on page 42, we have a CTC connection on each dispatcher machine. We did this to isolate heartbeat traffic away from the load-balancing traffic. It is generally a good idea to have your heartbeat separate from the main subnet body.

The IHS machines have HTTP servers, which were also obtained from WebSphere Edge components. We set up two IHS machines, along with the WebSphere V6 plug-in, and allowed the Network Dispatcher to load-balance traffic to them.

Note: Although we could have utilized a new plug-in generated by the ODR, we chose to simply modify the existing plug-in to include support for the additional WebSphere XD applications.

3.2.2 Our back-end systems

On the back-end systems, we had WebSphere Application Server for z/OS V6.0.1 installed on three system LPAR images (SC49, SC50, and SC54). We updated the level of the WebSphere Application Servers to V6.0.2.6, as this is the minimum level required to support WebSphere Extended Deployment V6.0.1.

We do not detail the installation and setup of WebSphere XD, but we do give an overview of the steps required.

WebSphere Extended Deployment install overview

WebSphere Extended Deployment is a function that is added to an existing WebSphere Application Server Network Deployment configuration. The process
for updating a configuration involves generating a set of customized jobs, then running those jobs to perform the *XD-ifying* of each node. Some key points are:

- This process is applicable to Network Deployment configurations only (not base standalone server configurations).
- The update process is done on a node-by-node basis. This is similar in concept to how maintenance is applied and migration is performed. Those functions are performed on a node-by-node basis as well.
- Three jobs are generated: EEXSTEP1, EEXSTEP2, and EEXSTEP3. All three are run for each node updated with XD. Refer to Table 3-1 for a description of the jobs.

Table 3-1 *Description of XD’s three generated jobs*

<table>
<thead>
<tr>
<th>Job</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEXSTEP1</td>
<td>Modifies the node's configuration HFS to support XD. This includes the creation of directories, files, and symbolic links to the XD HFS.</td>
</tr>
<tr>
<td>EEXSTEP2</td>
<td>Updates (or augments) the configuration profile. WebSphere for z/OS only uses one profile (default), but it is a profile and therefore it must be augmented to support XD.</td>
</tr>
</tbody>
</table>
| EEXSTEP3  | What this job does depends on the type of node being acted upon:  
  ▶ If this is a DMGR node, then this updates the Administrative Console application so it has all of the new XD functions.  
  ▶ If this is a managed node, then this performs a *reverse synchronization* with the DMGR, pushing back up to the master configuration information about the changes that have been made to the node. |

- The Deployment Manager node is always updated first. Managed nodes are done after that.
- When updating managed nodes, the Deployment Manager must be up and running.

**Difference between updating DMGR node and managed node**

This process is done on a node-by-node basis. Our network deployment configuration consisting of one DMGR node and three managed nodes (A, B, and C) implies four separate sets of customized jobs (one set for the DMGR, one set for each of the managed nodes). All four sets will have the same jobs: EEXSTEP1, EEXSTEP2, and EEXSTEP3.

The first two jobs, EEXSTEP1 and EEXSTEP2, do essentially the same thing, regardless of whether the node being updated is a DMGR node or a managed
node. But the third job, EEXSTEP3, behaves quite a bit differently, depending on what kind of node it is. Refer to Table 3-2.

Table 3-2 Types of nodes and actions performed on these nodes

<table>
<thead>
<tr>
<th>Type of node</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMGR node</td>
<td>The EEXSTEP3 job updates the existing administrative console application so that it will reflect the new XD support in the menus and options presented.</td>
</tr>
<tr>
<td>Managed node</td>
<td>The EEXSTEP3 job will read the configuration information in the node’s XML, figure out the IP host and SOAP port of the DMGR, and then make a TCP connection to the DMGR. This connection is used to invoke a reverse synchronization, from node up to DMGR, so that the DMGR can be made aware of the node’s new status as XD-ified.</td>
</tr>
</tbody>
</table>

The EEXSTEP3 job is smart enough to figure out which kind of node it is acting upon. You do not need to tell it. All of this is automatic. But it does bring up some very important issues:

- If the node being updated is a managed node, then the associated Deployment Manager must be up and running when the EEXSTEP3 job for the managed node is run. If it is not, then that attempt to establish the TCP connection will fail and the job itself will fail.
- If global security is enabled for the cell and the node being updated is a managed node, the EEXSTEP3 job will need to know the user ID and password used to authenticate into the DMGR’s SOAP port. There is a place on the customization panel to provide that. If global security is not enabled, then no authentication is needed, so that user ID/password combination would not be required.
After we have updated all of the nodes with WebSphere Extended Deployment features, we proceeded to create two On Demand Routers (ODRs) on LPARs SC50 and SC54. As shown in Figure 3-3, we have also created these dynamic clusters:

- Online transaction processing (OLTP)
- Long-running scheduler (LRS)
- Long-running execution environment (LREE)

![Figure 3-3 Our ITSO environment with two On Demand Routers and three dynamic clusters](image)

We describe the setup of the ODRs as well as these three dynamic clusters in the subsequent chapters.
We installed our DB2 database in data-sharing mode across the same three LPARs.

Besides referring to WebSphere Extended Deployment installation and customization guides, we strongly recommend using the white paper *WebSphere for z/OS Extended Deployment XD - Building Basic Infrastructure*, WP100735, as a reference to set up your infrastructure. It can be obtained by searching on WP100735 at the Techdocs Web site:

http://www.ibm.com/support/techdocs

We used the following hardware and software components for our setup. Table 3-3 lists the release levels of the products we used in our environment.

- **Coupling facility**
  We had two external coupling facilities (CFa) installed.

- **Open system adapter**
  For networking connections, we used three open system adapter (OSA) adapters.

- **WebSphere Application Server V6.0.2.6 for z/OS**

- **TCP/IP with Sysplex Distributor**

- **Resource Access Control Facility**
  Resource Access Control Facility (RACF®) uses a sysplex-wide shared database.

- **Resource Recovery System**
  Resource Recovery System (RRS) was used for two-phase commit.

- **Workload Manager**
  Workload Manager (WLM) was set up in goal mode.

Table 3-3 lists the release levels of the products we used in this project.

<table>
<thead>
<tr>
<th>Product name</th>
<th>Release level</th>
</tr>
</thead>
<tbody>
<tr>
<td>z/OS</td>
<td>1.6</td>
</tr>
<tr>
<td>SuSE Linux for zSeries</td>
<td>SLES 8 SP3 (kernel build 2.4.21-83-default)</td>
</tr>
<tr>
<td>WebSphere Application Server</td>
<td>6.0.2.6</td>
</tr>
<tr>
<td>WebSphere Extended Deployment</td>
<td>6.0.1.0</td>
</tr>
<tr>
<td>Product name</td>
<td>Release level</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>DB2 UDB for z/OS</td>
<td>8.1</td>
</tr>
</tbody>
</table>
Establishing the OLTP Environment

This chapter describes the steps we performed during the configuration of the two On Demand Routers (ODRs) used in our OLTP environment. It also describes setting up a dynamic cluster for use in the OLTP scenario.
4.1 Setting up the ODR

This section describes the steps we performed to set up an ODR used in the OLTP scenario in Chapter 5, “Online transaction processing (OLTP) scenario” on page 61. We later set up a second ODR on a different node using essentially the same steps described below.

4.1.1 Planning consideration for setting up the ODR

When an ODR is created using the administrative console, default values are assigned for various items such as server short name, cluster transition name, and ports. However, appropriate values that are needed are determined by the conventions adopted for the larger WebSphere Application Server environment that the ODR is being added to. There are many ways to establish a naming and port assignment convention. The important thing is that a convention is established and adhered to. The convention used for the names and the ports influence the Resource Access Control Facility (RACF) and Transmission Control Protocol/Internet Protocol (TCP/IP) work needing to be done when servers are added to the system. In many respects, an ODR can be treated much as a normal application server with regard to these considerations.

The WebSphere XD environment used in this book was established using conventions inspired by WP100735 XD - Building the Basic Infrastructure. Figure 4-1 shows a summary of the specific names and ports used.

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP_CONNECTOR_ADDRESS</td>
<td>40230</td>
</tr>
<tr>
<td>DRS_CLIENT_ADDRESS</td>
<td>7873</td>
</tr>
<tr>
<td>SAS_SSL_SERVERAUTH_LISTENER_ADDRESS</td>
<td>0</td>
</tr>
<tr>
<td>CSIV2_SSL_SERVERAUTH_LISTENER_ADDRESS</td>
<td>0</td>
</tr>
<tr>
<td>CSIV2_SSL_MUTUALAUTH_LISTENER_ADDRESS</td>
<td>0</td>
</tr>
<tr>
<td>WC_defaulthost</td>
<td>9080</td>
</tr>
<tr>
<td>DCS_UNICAST_ADDRESS</td>
<td>40233</td>
</tr>
<tr>
<td>WC_defaulthost_secure</td>
<td>9443</td>
</tr>
<tr>
<td>PROXY_HTTP_ADDRESS</td>
<td>40218</td>
</tr>
<tr>
<td>PROXY_HTTPS_ADDRESS</td>
<td>40219</td>
</tr>
<tr>
<td>ORB_LISTENER_ADDRESS</td>
<td>40231</td>
</tr>
<tr>
<td>ORB_SSL_LISTENER_ADDRESS</td>
<td>40232</td>
</tr>
<tr>
<td>BOOTSTRAP_ADDRESS</td>
<td>40231</td>
</tr>
</tbody>
</table>

*Figure 4-1 Port numbers for ODRs*
4.1.2 Creating a new ODR in the administrative console

As shown in Figure 3-3 on page 46, we have two ODRs for our environment. The process for creating one of these is as follows. It is very important to make the modifications described in the latter part of this section since the defaults are not appropriate for most situations. To create a new ODR in the administrative console perform the following steps:

1. Using the Administrative console, select Servers → On Demand Routers. This will present a list of On Demand Routers, as shown in Figure 4-2.

![On Demand Routers list](image)

2. Click New.

The server name for the ODR is the server long name, which can be set to any value, but should follow applicable conventions. We used odr1b. This should not be confused with the server short name that we will be modifying after initial creation of the ODR.

**Note:** The node selected for the ODR can be any node in the cell. This is different from WebSphere Extended Deployment on distributed platforms in that distributed platforms require that the ODRs not run on the same node on which dynamic cluster members run.
Select node **h6nodea** as the node on which to run the ODR and enter **odr1a** for the server name, as shown in Figure 4-3.

![Create a new On-Demand router entry](image)

**Figure 4-3** Node h6nodea

3. Click **Next**.

4. On the Select a server template panel, select the default **odr_zos** template, as shown in Figure 4-4.

![Create a new On-Demand router entry](image)

**Figure 4-4** Select a server template for ODR

5. Click **Next**.
6. On the Specify server specific properties panel, leave Generate Unique Http Ports checked as shown in Figure 4-5. We did this even though we will update the port values to match those specified in Figure 4-1 on page 50 at the beginning of the section.

7. Click Next.

8. A confirmation panel, as shown in Figure 4-6, will be displayed. Verify that the summary is correct.
9. Click **Next**.
10. Click **Save** at the top of the page.
11. Make sure Synchronize changes with Nodes is checked.
12. Click **Save**.

### 4.1.3 Modifying default values in the new ODR

To modify the default values for the ODR that we have just created, perform the following steps:

1. Select **Servers → odr1a**.
2. Expand the **Administration** settings under the Server Infrastructure panel and click **Customer Properties**, as shown in Figure 4-7.

![Figure 4-7 ODR configuration panel](image-url)
3. Click **ClusterTransitionName** and change the value from its default BBoxD01. Figure 4-8 shows the default value of the ClusterTransitionName.

![Figure 4-8 On Demand Router custom properties](image)

4. Click **Value** and select **H6ODR1A**, as shown in Figure 4-9.

![Figure 4-9 ClusterTransitionName custom property](image)

5. Click **OK**, then **Save** and **Synchronize**.
4.2 Setting up the OLTP scenario dynamic cluster

This section describes the steps we used to create the dynamic cluster used in Chapter 5, “Online transaction processing (OLTP) scenario” on page 61.

4.2.1 Creating the node group for the dynamic cluster

We create a dynamic cluster named OTLP_Workload_DC in a new node group named OLTP_NodeGroup.

Perform the following steps to create a node group:

1. Using the Administrative console, select **System administration → Node groups**. Click **New**, located at the top of the node groups list.

2. Provide a node group name and optional description. In this example, we use **OLTP_NodeGroup**, as shown in Figure 4-10.

3. Click **OK**.

4. In the resulting list of node groups, click **OLTP_NodeGroup**. This is the same configuration panel shown in Figure 4-10. However, now the node group members link under Additional properties are available.

5. Click **Node group members** and click **Add** at the top of the node group members list.

6. Select one or more nodes and click **Add**. In our scenario, we selected all three of the available nodes, h6nodea, h6nodeb, and h6nodec.
7. The resulting node group members list now shows the selected nodes as members.

4.2.2 Creating the node group for the dynamic cluster

Perform the following steps to create a dynamic cluster:

1. Using the Administrative console, select **Servers → Dynamic Clusters** and click **New**, located at the top of the dynamic clusters list.

2. Provide a dynamic cluster name and select a node group for the dynamic cluster to run on. For this scenario, we used the name OLTP_Workload_DC and mapped it to OLTP_NodeGroup defined earlier. We accepted the defaults for other values, as shown in Figure 4-11.

![Figure 4-11 Creating a dynamic cluster](image)

3. Select **defaultXDZOS** as the template for all members of the cluster, as shown in Figure 4-12. Click **Next**.

![Figure 4-12 Select a dynamic cluster template](image)
4. We decided to accept the defaults, as shown in Figure 4-13, for the purposes of this scenario. The defaults are more relevant to an environment where automatic mode is used. Since this scenario will be executed in manual mode these minimums and maximums are not used. Click Next.

![Figure 4-13 Dynamic cluster-specific properties](image)

5. Click Finish on the confirmation panel after verifying that everything is correct.

6. Click Save at the top of the page and then click Save to master configuration, making sure the synchronize changes with nodes is checked.

7. Wait until the save is complete before proceeding.

### 4.2.3 Modifying the default values in the new dynamic cluster

There are a number modifications that must be made to the dynamic cluster after it has been created, as well as modifications to each of its member servers. The modifications include:

- Cluster short name
- Server short name for each member server
TCP/IP port values for each server:
- SOAP JMX™ Port
- Bootstrap ORB Port
- HA Manager Comm Port
- HTTP Port
- HTTP SSL Port
- SIB Port
- SIB Secure Port
- SIB MQ Port
- SIB MQ Secure Port
- JMS Direct Port
- JMS Queued Port

Updating these values through the administrative console can be time consuming and error prone. We created a wsadmin jython script to update these values. The script assumes a name and port assignment policy similar to that described in the white paper *WebSphere z/OS V6 -- WSC Sample ND Configuration*, available at:

http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100653

We wrote the script to accept the cluster name, cluster short name, server short name template, and starting port number as parameters. The script is included with the additional materials described in Appendix B, “Additional material” on page 197. Refer to the readme file for specific instructions on copying the script and optional JCL file to your local environment.
Online transaction processing (OLTP) scenario

In this chapter we execute a sample OLTP application to demonstrate the advantages of using an On Demand Router (ODR) along with Workload Manager (WLM) in a z/OS environment.
5.1 OLTP scenario description

As discussed in 2.1, “Service policies and the On Demand Router” on page 12, the On Demand Router adds the following valuable capabilities to WebSphere Application Server for z/OS:

- Classification of requests at the point before they have been dispatched to a specific node. This classification is propagated with the request in the form of a TCLASS and can be utilized by z/OS WLM.
- The classification rules are more extensive than those previously available and are managed through a user-friendly interface in the administrative console rather than in a hand-edited file.
- Workload balancing to avoid over-utilization of any specific node is accomplished by routing requests based on the dynamic weights calculated by WAS XD based on the statistics gathered by z/OS WLM for each node.
- Request routing utilizing administrator-defined routing policies based on a rich set of criteria.
- Request routing to avoid certain health policy scenarios such as the storm drain scenario.

In this chapter we describe an online transaction processing scenario that we executed, enabling us to observe the behavior of the first three capabilities described above along with their synergy with z/OS WLM.
5.1.1 XDStock application

XDStock is a Web application that emulates a hypothetical stock application containing different Web modules, which are classified to different service policy goals and levels of importance. The four Web applications all perform the same iterative CPU-intensive string manipulations in order to amplify the observable effect of the prioritization done by z/OS WLM with regard to execution. A production application implementing these four usage scenarios would likely have very different execution characteristics since ours did not implement a database. See Figure 5-1 for an overview of our XDStock request flows.

Figure 5-1 XDStock request flow
We mapped the application’s Web modules to service policies, as shown in Table 5-1.

Table 5-1  Mapping of application modules to transaction classes

<table>
<thead>
<tr>
<th>Application module</th>
<th>TCLASS</th>
<th>Diagram color code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stock trade</td>
<td>XDPLATSC</td>
<td>Green</td>
</tr>
<tr>
<td>Financial advice</td>
<td>XDGOLDSC</td>
<td>Blue</td>
</tr>
<tr>
<td>Account management</td>
<td>XDSILVSC</td>
<td>Yellow</td>
</tr>
<tr>
<td>Stock query</td>
<td>XDBRONSC</td>
<td>Orange</td>
</tr>
</tbody>
</table>

5.2 Installing the XDStock application

In this scenario we deploy the XDStock application into the dynamic cluster defined in 4.1, “Setting up the ODR” on page 50. This is done as follows:

1. Using the Administrative console, select Applications → Enterprise Applications. Click Install located at the top of the enterprise applications list.

2. Locate the XDStock.ear file using either the local file system or the remote file system option on the path to the new application panel. Do not put anything in the Context root field.

3. Click Next.

4. Accept the defaults on the Choose default bindings and mappings panel. Click Next.

5. Accept the defaults on the Select installation options panel. Click Next.
6. In the Map modules to servers panel, select **OLTP_Workload_DC** from the clusters and servers list, select all four modules and then click **Apply**, as shown in Figure 5-2.

![Map modules to servers](image1)

**Figure 5-2**  Map modules to servers

7. The **OLTP_Workload_DC** should now appear in the server column for each of the modules. Click **Next**.

8. Accept the defaults on the Map virtual hosts to Web modules panel, and click **Next**.

9. On the summary panel, inspect the summary. Ignore the warning concerning no application modules being mapped to Web servers. The routing of requests to this Web server will be the ODR’s responsibility. Click **Finish**.

10. On the resulting page, the message shown in Figure 5-3 appears. Click **Save to Master Configuration**.

![Application XDStock installed successfully](image2)

**Figure 5-3**  Save to master configuration
11. On the Save panel, make sure Synchronize changes with Nodes is checked and click **Save**.

12. After the saving and synchronizing with nodes has completed, click **Ok**.

## 5.3 Configuring WLM on z/OS

We performed these steps to set up the service classes in Workload Manager on z/OS to match the service policies defined in the ODRs. It would be a good exercise to define the service classes and classification rules in z/OS WLM first before defining the corresponding service policies in WAS XD, as z/OS WLM only allows up to eight characters for transaction classes.

To configure z/OS WLM, perform the following steps:

1. Extract the current z/OS Workload Load Management service definition, as shown in Figure 5-4.

   ![Figure 5-4](image)

   *Figure 5-4  Extract definition from WLM couple data set*
2. Create the workload WASXD in option 2 of WLM, as shown in Figure 5-5.

[Table]

<table>
<thead>
<tr>
<th>Definition data set</th>
<th>none</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition name</td>
<td>spstpc (Required)</td>
</tr>
<tr>
<td>Description</td>
<td>WLM for WebSphere Perf monitor</td>
</tr>
</tbody>
</table>

Figure 5-5  Create a workload

[Table]

<table>
<thead>
<tr>
<th>Workload Name</th>
<th>WASXD (Required)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>WebSphere Extended Deployment WL</td>
</tr>
</tbody>
</table>

Figure 5-6  Create workload WASXD

3. Navigate back to the definition menu and choose option 4 to create the platinum service class called XDPLATSC under the workload of WASXD, as shown in Figure 5-7.

[Table]

<table>
<thead>
<tr>
<th>Service Class Name</th>
<th>XDPLATSC (Required)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>XD Service Class - Platinum</td>
</tr>
<tr>
<td>Workload Name</td>
<td>WASXD (name or ?)</td>
</tr>
<tr>
<td>Base Resource Group</td>
<td>(name or ?)</td>
</tr>
<tr>
<td>Cpu Critical</td>
<td>NO (YES or NO)</td>
</tr>
</tbody>
</table>

Specify BASE GOAL information. Action Codes: I=Insert new period, E=Edit period, D=Delete period.

---Period--- Goal----------
Action # Duration Imp. Description

Figure 5-7  Create service class XDPLATSC
4. Create service class XDPLATSC with an average response time of 0.125 with an importance of 1, as shown in Figure 5-8.

![Average response time goal](attachment:average_response_time_goal.png)

**Figure 5-8** Define response time and importance level of service class XDPLATSC

5. Create the rest of the service classes, as shown in Table 5-2.

**Table 5-2 Service class definitions**

<table>
<thead>
<tr>
<th>Service class</th>
<th>Average response time</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDPLATSC</td>
<td>0.125</td>
<td>1</td>
</tr>
<tr>
<td>XDGOLDSC</td>
<td>0.15</td>
<td>1</td>
</tr>
<tr>
<td>XDSILVSC</td>
<td>0.20</td>
<td>2</td>
</tr>
<tr>
<td>XDBRONSC</td>
<td>0.30</td>
<td>2</td>
</tr>
</tbody>
</table>

6. There should be a couple of service classes defined when WebSphere Application Server Network Deployment is installed. Use one of them as the default. In our project, we chose a service class called WASDF with an importance of 1 and a response time of 0.35 seconds with a percentile of 90%. This will be used for all of our default WebSphere workloads.
7. Next, we classify the incoming WAS XD Service Policy by creating classification rules. Select 6 to enter the classification rules menu, as shown in Figure 5-9.

![Figure 5-9 Create classification rules](image)

8. WebSphere Application Server uses the subsystem type of CB, input 3 to create the classification rules, as shown in Figure 5-10.

![Figure 5-10 Modify the classification rule for CB](image)
9. Create the four classification rules shown in Table 5-3. Figure 5-11 shows the mappings of the classification rules after they are created. We create these XD transaction classes later in this chapter.

<table>
<thead>
<tr>
<th>XD transaction classes</th>
<th>z/OS WLM service classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>XDPLATTC</td>
<td>XDPLATSC</td>
</tr>
<tr>
<td>XDGOLDTC</td>
<td>XDGOLDSC</td>
</tr>
<tr>
<td>XDSILVTC</td>
<td>XDSILVSC</td>
</tr>
<tr>
<td>XDBRONTC</td>
<td>XDBRONSC</td>
</tr>
</tbody>
</table>

10. You will be prompted to specify the disposition of the changes made in WLM. Select 2 to save the WLM definition in your installation’s WLM couple data set, as shown in Figure 5-12.
11. After exiting from WLM ISPF panels, vary and activate the definition to affect the changes made, as shown in Figure 5-13.

<table>
<thead>
<tr>
<th>Service policy</th>
<th>Goal</th>
<th>Importance</th>
<th>Transaction class</th>
<th>Service class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platinum_SP</td>
<td>1250 ms</td>
<td>Highest</td>
<td>XDPLATTC</td>
<td>XDPLATSC</td>
</tr>
<tr>
<td>Gold_SP</td>
<td>1500 ms</td>
<td>High</td>
<td>XDGOLDTC</td>
<td>XDGOLDSC</td>
</tr>
<tr>
<td>Silver_SP</td>
<td>2 sec</td>
<td>Medium</td>
<td>XDSILVTC</td>
<td>XDSILVSC</td>
</tr>
<tr>
<td>Bronze_SP</td>
<td>3 sec</td>
<td>Low</td>
<td>XDBRONTC</td>
<td>XDBRONSC</td>
</tr>
</tbody>
</table>

The following steps detail how we defined the platinum service policy for the XDStock trade application. Repeat the following steps for each of the service classes shown in Table 5-4:

1. Using the Administrative console, select Operational Policies → Service Policies. Click New, located at the top of the service policies list.
2. Enter the service policy name Platinum_SP on the Define Service Policy General Properties panel. Also select a goal type of Average Response Time, as shown in Figure 5-14.

![Figure 5-14  Define Service Policy General Properties panel](image)

3. Click Next.

4. On the Define Service Policy Goal Properties panel, enter a goal value of 1250 and select Milliseconds as the goal value units. Select an importance of Highest, as shown in Figure 5-15.

![Figure 5-15  Define Service Policy Goal Properties panel](image)
5. Click **Next**.

6. On the Define Service Policy Memberships panel, we added the XDPLATTC during the service policy creation process. Alternatively, we could have waited until the service policy was created and then added the transaction class as a member. To add a transaction class during service policy creation, click **New** on the Defined Service Policy Memberships panel, as shown in Figure 5-16.

![Figure 5-16 Define Service Policy Memberships panel](image)

7. The first of the transaction class-related panels will be presented, after which the flow will return to the service policy membership panel and allow completion of the service policy. On the first of these, the Define Transaction Class General Properties panel, enter XDPLATTC for the name of the transaction class, as shown in Figure 5-17.

![Figure 5-17 Define Transaction Class General Properties panel](image)

8. Click **Next**.
9. The resulting confirmation panel should appear, as shown in Figure 5-18.

![Figure 5-18 Confirm Transaction Class Creation panel](image)

10. Click **Finish**.

11. The flow now returns to the Create Service Policy panels. The Define Service Policy Memberships panel now lists XDPLATTC as a member, as shown in Figure 5-19.

![Figure 5-19 Define Service Policy Memberships panel containing new transaction class](image)

12. Click **Next**.
13. The Confirm Service Policy Creation panel will display both the service class and the transaction class information, as shown in Figure 5-20.

![Confirm Service Policy Creation panel]

Figure 5-20 Confirm Service Policy Creation panel

14. Click **Finish**.

15. Repeat these steps for each of the entries in Table 5-3 on page 70. After completion, the service policies list should appear, as shown in Figure 5-21.

![Completed service policies list]

Figure 5-21 Completed service policies list
5.4.2 Adding work classes to the XDStock application

Work classes allow for the identification of specific types of HTTP or SOAP requests to a class of work. This work class can then be associated with a transaction class. In general, this relationship indicates that the set of application requests characterized by the work class should be managed against the goals specified by the service policy associated with the transaction class.

Table 5-5  Mapping of modules to work classes and transaction classes

<table>
<thead>
<tr>
<th>Application module</th>
<th>Work class name</th>
<th>Transaction class</th>
</tr>
</thead>
<tbody>
<tr>
<td>StockTrade.war</td>
<td>StockTrade_WC</td>
<td>XDPLATTC</td>
</tr>
<tr>
<td>FinancialAdvice.war</td>
<td>FinancialAdvice_WC</td>
<td>XDGOLDTC</td>
</tr>
<tr>
<td>AccountManagement.war</td>
<td>AccountManagement_WC</td>
<td>XDSILVTC</td>
</tr>
<tr>
<td>StockQuery.war</td>
<td>StockQuery_WC</td>
<td>XDBRONTC</td>
</tr>
</tbody>
</table>

The definition of work classes for an application is done on a service policies tab that WAS XD adds to the enterprise applications panel.

**Note:** Work classes are associated with the enterprise application, so care should be taken to not unnecessarily uninstall the application after establishing the work classes. Uninstalling the application removes all of existing mappings.

In our testing, doing application updates had no negative effect on the work classes and the mapping we brought forward from the previous installation of the application.
The process for adding work classes to an enterprise application and then mapping them to the appropriate transaction is illustrated using the StockTrade_WC work class as an example. We repeated the following steps for each of the service policies listed in the table:

1. On the administrative console, select **Applications → Enterprise Applications**. Click **XDStock** in the enterprise applications list. The resulting page is shown in Figure 5-22.

![Figure 5-22   XDStock enterprise application detail](image)
2. Click the **Service Policies** tab near the top of the panel and expand **Work Classes for HTTP Requests**, as shown in Figure 5-23.

![Figure 5-23 Work classes for HTTP requests](image)

3. Click **New**.

4. Enter a name of `StockTrade_WC` on the Define Work Class General Properties panel, as shown in Figure 5-24.

![Figure 5-24 Defining Work Class General Properties panel](image)

5. Click **Next**.

6. On the Define Work Class Memberships panel, select the `StockTrade.war` module in the module drop-down list.

7. Select all of the available HTTP patterns and click **Add >>**.
8. Verify that the panel looks like Figure 5-25.

![Figure 5-25 Define Work Class Membership panel]

9. Click **Next**. The Confirm Work Class Creation panel should appear, as shown in Figure 5-26.

![Figure 5-26 Confirm Work Class Creation panel]

10. Click **Next**.
11. Back in the Enterprise Applications > XDStock Panel, expand the **Work classes for HTTP requests** section followed by the **StockTrade_WC** section, as shown in Figure 5-27.

![Figure 5-27](image)

12. Under **If no classification rules apply**, then classify this to **transaction class**, select **XDPLATTC (Platinum_SP)** from the drop-down list.

13. Near the top of the panel, click **Apply**.

14. Repeat steps 3–15 for each of the modules in Table 5-5 on page 76.

15. After completing this for all modules, click **Save** at the top of the panel and then save the changes to the master configuration. Make sure “Syncronize changes with nodes” is checked.
5.5 Adjusting servant minimum and maximum

To change the minimum and maximum servant instances, perform the following steps:

1. Using the administrative console, select **Servers → OLTP_Workload_DC_h6nodea**. Click **Java and Process Management** under Server Infrastructure and click **Server Instance**, as shown in Figure 5-28.

   ![Figure 5-28](image)
   
   **Figure 5-28** Java and process management under server infrastructure

2. Ensure that **Multiple Instances Enabled** is checked and that minimum and maximum number of instances is set to 5.

   ![Figure 5-29](image)
   
   **Figure 5-29** Server instance
3. Click **OK** and repeat steps 1–4 for the rest of the OLTP application servers.

4. **Save** and **synchronize** the settings.

### 5.6 Activating the propagation of TCLASS

To propagate the transaction classes set for the service policies in XD, you must create these two custom properties for each of your application servers:

- TrustedProxy = true
- odr_tclass_propogation_enable = 1
To create these, perform the following steps:

1. Using the administrative console, select **Servers → Application servers → OLTP_Workload_DC_h6nodea**. Expand the **Web Container Settings** under Contain Settings and click **HTTP transports**, as shown in Figure 5-30.

![Figure 5-30 Web container for HTTP transports](image-url)
2. Click the asterisk (*) for the non-SSL (port 40638) in order to set the two custom properties, as shown in Figure 5-31.

![Figure 5-31 HTTP transport ports](image)

3. Click **Custom Properties** under Additional Properties, as shown in Figure 5-32.

![Figure 5-32 Configuration for port 40638](image)
4. Create these two custom properties (case sensitive), as shown in Figure 5-33:
   - TrustedProxy = true
   - odr_tclass_propogation_enabled = 1

5. Repeat steps 1 on page 64–4 for the rest of the OLTP workload servers.

6. Restart all three application servers to affect the two custom properties.

5.7 Executing the scenario

In this section we apply a workload to the application using the WebSphere Studio Workload Simulator. The XDStock application is executing on the dynamic cluster created above with all of the cluster members started. Each cluster member is configured to have five servant regions to allow for isolation of the four different TCLASSs. There were two ODRs active during the execution of this scenario, with sysplex distributor spreading the requests across the two ODRs.

During the initial stage of the scenario, we applied a moderate workload and focused on observing the propagation of the WebSphere XD service classification via the TCLASS to WLM and the execution of the requests within the resulting z/OS WLM enclave.

Later, we increased the workload until we saturated the dynamic cluster observing the distribution of work across the cluster members. The fact that the various cluster members in the dynamic cluster were configured on logical partitions (LPARs) having different capacities was fortunate, in that it allowed us
to observe the ODR distributing work across a cluster of members having different capacities for performing work.

5.7.1 Starting the On Demand Routers

Two ODRs were used in the execution of this scenario. We started them on our h6nodea and h6nodeb as follows:

1. Using the administrative console, expand **Servers** and select **On Demand Routers**. See Figure 5-34.

![Figure 5-34 Starting the ODRs for the OLTP scenario](image)

2. Select **odr1a** and **odr1b** and click **Start**.

5.7.2 Starting the application servers in the dynamic cluster

We started all of the application servers that were cluster members of the OLTP_Workload_DC dynamic cluster created in the previous chapter. To do this we performed the following steps:

1. Using the administrative console, expand **Servers** and select **Application Servers**.

...
2. Select all three application servers, as shown in Figure 5-35.

![Figure 5-35: Starting the members of the OLTP workload dynamic cluster](image)

3. Click **Start**.

### 5.7.3 Preparing to observe the scenario

We observed the workload execution using the runtime topology view provided by WebSphere Extended Deployment in the administrative console. We charted two sets of statistics on the same chart. First, we tracked the number of concurrent requests for each server. This allowed us to see how many requests the ODRs were dispatching to the various cluster members. The number of concurrent requests allowed for each server is the weight factor used to balance the requests sent to by the ODRs to avoid over-utilization of a cluster member. In the scenario we observed this weight was adjusted as certain cluster members became saturated.

The second set of statistics were the average response times being achieved for each of the service classes defined. We wanted to observe the achievement of the goals as we pushed the workload to the saturation point.

To prepare for these observations, we configured a chart as follows:

1. Using the Administrative console, expand **Runtime Operations** and select **Runtime Topology**.
2. Under h6cell, expand **Node Groups → OLTP_NodeGroup**.

3. Click **OLTP_NodeGroup** and select **Chart data set → Chart this item** from the menu, as shown in Figure 5-36. This will create a new tabbed display panel containing a chart and associated legend. Other tabbed panels displayed previously by default can be dismissed by clicking the X on their tabs.

Figure 5-36   Creating a chart for the OLTP_NodeGroup

4. The resulting chart was configured by default to display the average response time for the entire node. We removed that data set from the chart by selecting it and clicking **Remove**, as shown in Figure 5-37.

Figure 5-37   Removing the default data set from the chart
5. Next, we clicked **Add data**.

![Figure 5-38  Adding concurrent request’s data set for application servers](image)

6. We then selected data set type.

![Figure 5-39  Adding average response time data set for service policies](image)
7. The resulting chart has a legend similar to that shown in Figure 5-40. The observations in the scenario were based on these statistics.

![Figure 5-40 Runtime topology statistics](image)

*Figure 5-40 Runtime topology statistics*
5.7.4 Applying the workload

To start the scenario we applied a moderate workload using WebSphere Studio Workload Simulator. We slowly increased the workload until the node with the least capacity, h6nodec, neared 80% utilization, as shown in Figure 5-41. Notice that at this same point in time the other two nodes are significantly less utilized. This is because while h6nodec has relatively high utilization, it has crossed the threshold at which its dynamic weight will be decreased to avoid over-saturation.

![Figure 5-41](image_url)  
Figure 5-41  CPU utilization of OLTP_NodeGroup
At this point, we inspected the enclaves display in SDSF to verify that the TCLASS was being propagated by the ODR and picked up by WLM, as shown in Figure 5-42.

Figure 5-42  SDSF enclave panel showing service classes
We were also able to observe from the SYSPRINT for the various servant regions that the requests for the URLs associated with the various service classes were isolated, each in their own servant region.

Figure 5-43  All three nodes fully utilized
We then slowly increased the number of simulated clients until the utilization of all three nodes approached saturation, as shown in Figure 5-43 on page 93. The chart shown in Figure 5-44 shows both the concurrent requests per server and the average response time for each service policy. The horizontal bars are the various service policy goals.

Figure 5-44: Concurrent requests per server and average response time for each service policy
We made the following observations on the chart:

- As the workload increased, priority was given to platinum and gold service requests relative to silver and bronze. This prioritization was managed by z/OS WLM along with PRSM and IRD. RMF™ III snapshots before (Figure 5-45) and after (Figure 5-46 on page 96) saturation give a z/OS view of the goal attainment.

- Concurrent requests for the servers on the different nodes were nearly equal while utilization was moderate, as signified by the light blue line in the bottom left corner of the chart. As the workload increased, the concurrent requests for h6nodea and h6nodeb increased. This was due to the fact that their dynamic weights were adjusted upward while the dynamic weight for h6nodec was adjusted downward. The ODR used these weights to determine how many concurrent requests to allow for each of the cluster members. This allowed the workload to be driven until all of the nodes were near full utilization even though they possessed differing capacity to perform work.

![Figure 5-45 RMF sysplex summary before saturation](image_url)
We were also able to observe the effects of the prioritization provided by z/OS WLM, PRSM, and IRD by looking at the output of our XDStock application.

**Note:** The application logic and iteration parameter being executed for each of the requests was the same code for the same number of iterations.

During execution the XDStock application computes the number of milliseconds (milis) elapsed for each XDStock request and includes this in the SYSPRINT output. We compared the output in two different servants for the application server running on h6nodec at 20:40:52:

- The servant processing platinum requests had the following:
  
  40218/StockTrade/CPUBound elapsed milis = 40
  40218/StockTrade/CPUBound elapsed milis = 45
  40218/StockTrade/CPUBound elapsed milis = 41
  40218/StockTrade/CPUBound elapsed milis = 44
  40218/StockTrade/CPUBound elapsed milis = 91
  40218/StockTrade/CPUBound elapsed milis = 94

- The servant processing bronze requests had the following:

  40218/StockQuery/CPUBound elapsed milis = 232
  40218/StockQuery/CPUBound elapsed milis = 189
  40218/StockQuery/CPUBound elapsed milis = 182
5.7.5 Conclusion

While executing the scenario described in this chapter, we were able to observe the On Demand Router and the Workload Management Facility of z/OS contribute in different yet complimentary ways to optimize the execution of our XDStock application. Specifically, we observed the ODR and z/OS WLM each making the following contributions:

- The On Demand Router moderated the number of requests dispatched to each node to avoid over-utilization of any given node. Recall that our dynamic cluster had member servers running on nodes with different capacities, especially h6nodec, which had significantly less capacity than either h6nodea or h6nodeb. At the beginning of our test when the workload was fairly low, the requests were routed to the three cluster members in a roughly even distribution, resulting in h6nodec registering a 78% utilization, while the other nodes were closer to 50% (refer to Figure 5-41 on page 91). As the workload increased, the dynamic weights of the cluster members were adjusted. Weights are expressed in terms of the maximum number of outstanding requests allowed. We saw this manifested in the divergence of the concurrent requests stats for the different application servers while all three nodes approached full utilization in spite of their differing capacities.

- We also observed the ODR classifying the requests according to classification rules defined using the WebSphere Application Server’s administrative console. The TCLASS associated with the classification was propagated along with the HTTP request where we observed enclaves being created by WLM for each classified request (refer to Figure 5-42 on page 92).

- The Workload Management Facility in z/OS managed the queuing and prioritization of the requests against the service class goals associated with the propagated TCLASS values. We observed the gold and platinum requests being executed with higher priority at the expense of silver and bronze when the system was under heavy load while still making a best effort to meet the lower priority goals as well. Upon inspection of the XDStock application SYSPRINT output, we observed the application logic executing significantly faster for the platinum requests demonstrating WLM along with PRSM and IRD working in unison.

- We also observed various statistics in the charts and graphics we observed in the administrative console. Many of these statistics, along with the statistics used to calculate and adjust the dynamic weights of the servers, were gathered by z/OS WLM and consumed by the WebSphere Extended Deployment services.
Establishing the long-running environment

This chapter describes how to configure WebSphere Extended Deployment on z/OS services to support the execution of long-running workloads.

This chapter follows procedures documented in the white paper WebSphere for z/OS Extended Deployment XD - Building the Basic Infrastructure (WP100735) found at:

http://www.ibm.com/support/techdocs

We recommend that you review this white paper for a more in-depth understanding of the configuration process. Those procedures are repeated here and modified to address the configuration of our ITSO environment.
6.1 Configuring the long-running environment

WebSphere Extended Deployment provides services to enable the execution of long-running workloads. However, these services are not enabled out-of-the-box, but must be configured. In this way, you deploy these services only to that portion of your WebSphere environment that requires them.

In 2.2, “The long-running environment” on page 22, we describe the overall architecture of the long-running environment. Building a long-running environment requires configuring two WebSphere Extended Deployment components:

- Long-running scheduler (LRS)
- Long-running execution environment (LREE)

For each component, we first present an overview of the configuration process, followed by the step-by-step configuration procedures.

6.2 Configuration the long-running scheduler

This section describes how we configure the long-running scheduler in our ITSO environment.

6.2.1 Long-running scheduler configuration overview

The following is an overview of the process to configure the environment for the LRS services:

1. Create a node group and dynamic cluster for LRS.
2. Perform additional configuration to support the LRS cluster.
3. Create the LRS database tables.
4. Configure the JDBC provider (if necessary) and LRS data source.
5. Deploy the LRS application - LongRunningScheduler.ear.
6. Validate the configuration.
Our plan is to define a dynamic cluster consisting of two LRS application servers, on node A and node C on z/OS LPAR SC49. Figure 6-1 represents the ITSO configuration we plan to build for LRS.

**Note:** We are going to add the LRS to the previous OLTP-only configuration.

![Figure 6-1 ITSO configuration with LRS dynamic cluster added](image)

### 6.2.2 Creating node group and dynamic cluster for LRS

Node groups and dynamic clusters are the fundamental building blocks for a long-running environment, just as they are for the dynamic operations (transactional) environment. A node group must be created for LRS, other than the default node group, before creating a dynamic cluster for LRS.
Creating the LRS node group
Create the node group as follows:

1. In the administrative console, select **System administration → Node groups**.

2. The Node groups display lists the node groups currently defined at this point, the DefaultNodeGroup and the StockNodeGroup created in Chapter 4, “Establishing the OLTP Environment” on page 49. Click **New**.

   ![Node groups](image)

   **Figure 6-2  Node groups**

3. In the Node groups configuration screen, enter LRSNodeGroup as the long name for the node group in the Name field, and enter a description of your choice.
4. Click **Apply**. For now, ignore the message to save the configuration. We will save the configuration in a few steps. Clicking **Apply** un-grays the Node group members link. Click the link.

![Figure 6-3 Defining the LRS node group](image)

5. In the Node group members panel, click **Add**. This brings up a screen that lists all of the available nodes in our ITSO WebSphere cell.

![Figure 6-4 Selecting LRS node group members - part 1](image)
6. We do not want the LRS to be a single point of failure. For a high-availability deployment we need at least two managed nodes in the node group that will be used later in the LRS dynamic cluster. Do not select the Deployment Manager node. Select the member nodes for the LRSNodeGroup and click Add.

![Node groups > LRSNodeGroup > Node group members > Add node](image)

Figure 6-5  Selecting LRS node group members - part 2

7. This brings up a screen showing the selected nodes. Now click the **Save** link at the top of the screen. Then check **Synchronize changes with Nodes** and click **Save** to save the configuration.

**Creating the LRS dynamic cluster**

The LRS dynamic cluster is created in the LRSNodeGroup as follows:

1. In the administrative console, select **Servers → Dynamic Clusters**. This brings up a screen with a current list of dynamic clusters. In the Dynamic Cluster screen, click **New**.

2. This brings up the Create new Dynamic Cluster screen Step 1.
   a. Enter LRS in the Dynamic Cluster name field.
b. The Map to node group menu lists the currently defined node groups. Select **LRSNodeGroup** and click **Next**.

![Figure 6-6 Creating a LRS dynamic cluster - step 1](image1)

3. In step 2, we select a dynamic cluster template as follows:

   a. Specify the server **template** WebSphere uses to create application servers in the LRS dynamic cluster. Select **defaultXDZOS** from the Default application server template pull-down.

   b. Leave Existing application server un-selected since there are no LRS servers defined at this point.

   c. Click **Next**.

![Figure 6-7 Creating the LRS dynamic cluster - step 2](image2)
4. The dynamic cluster-specific properties are specified in step 3.
   a. For the Minimum number of cluster instances section, select **Keep one instance started at all times**. By design, only one instance of LRS may be started at any given time.
   b. In the Maximum number of cluster instances section, select **Limit the number of instances that can start** and enter 1. Again, only one instance of LRS may be started at any time.
   c. Click **Next**.

![Create new Dynamic Cluster](image)

*Figure 6-8  Creating the LRS dynamic cluster - step 3*

5. At the Confirm Dynamic Cluster panel, click **Finish**.
6. Click **Save** and synchronize the changes.
At this point, our configuration looks like Figure 6-1 on page 101. There is more LRS configuration work to be done, as described in the next sections.

### 6.2.3 Additional configuration for the LRS dynamic cluster

At this point we perform additional configuration work to support the LRS dynamic cluster. The steps are:

1. Make certain the number of servant regions is set to one.
2. Re-map the server short names.
3. Re-map the cluster short name.
4. Enable the Mbean service.
5. Map the server ports.
6. Create the virtual host aliases for the LRS application servers.

We now perform each step with an explanation of why it must be done.

**Make certain the number of servant regions is limited to one**

WebSphere Extended Deployment V6.0 architecturally limits the number of running LRS instances to one per cell. The LRS dynamic cluster is set to the default mode of manual. This means that LRS application servers do not start automatically. The same is true for the servant regions. WebSphere, by default, sets the servant regions to “Minimum=1, Maximum=1”. Do the following to verify this:

1. In the administrative console, select **Servers → Application Servers**.
2. This presents a list of all of the application servers configured so far. Click the link for one of the two LRS dynamic cluster server members (for example, select **LRS_h6nodea**).

![Application Server panel with the LRS application servers](image)

*Figure 6-9 Application Server panel with the LRS application servers*
3. Then go to server infrastructure. Expand **Java and process management** and select **Server Instance**.

4. Verify that both the minimum number of instances and maximum number of instances are set to 1. This means that only one LRS will be started initially, and only one regardless of the workload. This is how WebSphere Extended Deployment is designed to manage the LRS dynamic cluster.
Re-mapping server short names

Server short names are important in WebSphere for z/OS because they end up relating to RACF STARTED profiles and the JOBNAME of the servant regions. This applies to both WebSphere Application Server for z/OS static clusters and WebSphere Extended Deployment for zOS static and dynamic clusters. The name is limited to seven characters.

By default, the value WebSphere for z/OS assigns to the cluster members is BBOS00n, where n is an incremented number. We assign the server short names H6LRS1A and H6LRS1C to the LRS_h6nodea and LRS_h6nodec application servers, respectively. Refer to the white paper *XD - Building the Basic Infrastructure* found at the following Web site for additional information:

http://w3-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100735

Follow these steps to re-map server short names:

1. In the administrative console, select **Servers** → **Application Servers**.
2. Select one of the LRS servers in the dynamic cluster, for example, **LRS_h6nodea**.
3. The default short name is BBOB001. Change this to H6LRS1A.

   ![Figure 6-12 Setting the LRS application server short names: before and after](image)

4. Click **OK**.
5. Click the **Application servers** link to go back to Application servers screen.
6. Repeat the above for the other LRS application server, LRS_h6nodec. Change its short name to H6LRS1C.
7. Click OK.
8. Save and synchronize your changes.

**Re-mapping cluster short name**

The cluster short name defaulted to BBOC001 for the LRS dynamic cluster. That value is used as the WLM application environment name for all of the servers in the cluster. It is also related to the CBIND and SERVER profiles in RACF.

Following the advice in the white paper *XD - Building the Basic Infrastructure*, we recommend that the cluster short name be a subset of the server short names of the server in the cluster.

The following procedure re-maps the cluster short name:

1. In the administrative console, go to **Servers → Dynamic Clusters**.
2. Select **LRS**, the link for the LRS dynamic cluster.
3. Under General Properties, change the short name value from the default BBOC001 to H6LRSA.

4. Click OK.
5. Save and synchronize your changes.
**Enabling the startup Mbean service**

A *startup bean* is a J2EE session bean that gets started when the application server itself is started. This is a requirement for LRS. Do the following to enable this service for LRS:

1. In the administrative console, go to **Servers → Application Servers**.
2. Click **LRS_h6nodea**, the link for the first LRS application server in the dynamic cluster.
3. Under Container Settings, expand **Container Services** and select the **Startup beans service** link.

   ![Figure 6-14 Navigating to the Startup beans service link](image)

4. Under General Properties, check **Enable service at server startup**.

   ![Figure 6-15 Checking the status of the startup beans service](image)

5. Click **OK**.

6. Click the **Application servers** link to go back to the list of application servers and repeat the above procedure for the other LRS application server in the dynamic cluster, **LRS_h6nodec**.

7. When complete, save and synchronize your changes.
Mapping the server ports

When WebSphere creates new application servers, the TCP ports assigned to the applications are set by WebSphere. The ports assigned most likely are not the ports you want for your installation. The issue of remapping the TCP ports is the same for WebSphere Extended Deployment for zOS dynamic cluster members as it is for WebSphere Application Server on z/OS static cluster members. A dynamic cluster member is still just an application server with the same TCP ports as any other server.

Refer to XD - Building the Basic Infrastructure for additional information about mapping server ports. That resource suggests that you first have a plan for how you assign port numbers and provides excellent resource references to help you develop that plan. Once you have a plan, follow the procedure outlined in the following section.

Non-HTTP ports

For non-HTTP ports, do the following:

1. In the administrative console, go to Servers → Application Servers.
2. Click the link for one of the two LRS application servers we configured previously.
4. The resulting panel displays a list of port names and the default port numbers assigned.
5. Perform the following for each port entry:
   a. Click the Port Name link. The properties for the that port are displayed.
   b. Change the Port field to the port number you identified in your plan.
   c. Click Apply.
6. Repeat the above for the other LRS application servers.
7. Save and synchronize.

HTTP ports

Perform the following steps to map the HTTP ports:

1. In the administrative console, go to Servers → Application Servers.
2. Click the line for one of the two LRS application servers.
3. Under Web container settings, click HTTP Transports.
4. The resulting panel shows a table with two entries: one for the non-SSL port and one for the SSL port. Perform the following steps for each entry:
   a. Click the asterisk (*) under the Host column.
b. On the resulting General Properties panel, change the default assignment in the Port field to the one you identified in your plan.

c. Click Apply.

5. Repeat the above for the other LRS application server.

6. Save and synchronize.

Creating the virtual host aliases for the LRS application servers

The servers in the LRS dynamic cluster need to have their HTTP and HTTPS ports included in the list of aliases for a virtual host. We used the default_host alias in our ITSO configuration.

1. In the administrative console, go to Environment → Virtual Hosts.

2. Click the default_host link.


4. Click New.

5. On the resulting General Properties panel, leave the asterisk in the Host field and change the Port field to the non-SSL port identified in your plan.

6. Click Apply.

7. Repeat for the above for the SSL port.

8. Save and synchronize.

6.2.4 Creating the LRS database and tables

LRS requires a table to manage the submission and dispatching of long-running jobs. This involves customizing and submitting some batch jobs. The ITSO environment uses DB2 V8 for z/OS. Our procedures follow the steps in the white paper XD - Building the Basic Infrastructure.

Default schema names

DB2 database table names have qualifiers referred to as the schema name. Queries made to DB2 must employ the schema name as well as the table name. The tables for LRS (and LREE) have default schema names. You can use these names or chose your own. We chose to use the default in order to match the procedures in the existing documentation. Obviously, if you define another WebSphere cell configured for long-running execution, choose a different schema name if you intend to share the same DB2 system. Refer to the white paper XD - Building the Basic Infrastructure for recommended procedures for choosing a unique schema name.
Database creation scripts
WebSphere Extended Deployment for zOS comes with sample database creation scripts:

- The scripts are located in the ..//longRunning directory for each node. The name of the script to create the LRS database and tables is SPFLRSV7. Note that this script works for both DB2 V7 and V8.
- We used SPUFI to create the table. Copy the script from the HFS to a PDS for SPUFI.
- The supplied database and table scripts have the CREATE STOGROUP statement commented out:
  
  -- CREATE STOGROUP LRSCHED1 VOLUMES9DB2WK1) VCAT USER104;

- Un-comment the line (remove the two dashes) and set the volumes and VCAT definitions appropriate for your system.

6.2.5 Configuring the JDBC provider and LRS data source

Once again, refer to the white paper XD - Building the Basic Infrastructure for an in-depth guide and discussion of defining JDBC™ providers and data sources.

JDBC provider definition
In our ITSO environment, we use one JDBC provider for the multiple data sources required by the long-running environment components and sample applications. Here we define the JDBC provider. Note that all definitions are at the cell scope.

The procedure for defining a JDBC provider in WebSphere Extended Deployment for zOS V6 is the same as any V6 version of WebSphere. It is presented here for completeness. The procedure is rather lengthy. However, it only has to be performed once. You should perform the following:

- Define a DB2 JDBC provider.
- Define DB2 environment variables for the JDBC provider.
- Delete the DB2 environment variables at the node level.
- Define a DB2 subsystem properties file.
- Create a servant JVM custom property to point to the properties file.

Defining a DB2 JDBC provider
Do the following to configure the JDBC provider for the long-running environment:

1. In the administrative console, go to Resources → JDBC Providers.
2. Scope the resource to the cell level by clearing the value in the Node field and clicking **Apply**. There should be a little red arrow pointing to Cell.

3. Click **New** to define a new JDBC provider.

![JDBC providers panel at cell scope](image)

**Figure 6-16   JDBC providers panel at cell scope**

4. In the General Properties box that appears, enter the following properties from the pull-down menu for each step:
   
   a. Step 1: Select the database type by selecting **DB2**.
   
   b. Step 2: Select the provider type by selecting **DB2 Universal JDBC Driver Provider**.
   
   c. Step 3: Select the implementation type by selecting **Connection pool data source**.
5. Click **Next**.

![Figure 6-17 Selecting JDBC provider properties](image)

6. The next panel displays the general properties for the JDBC provider just created. The class path and native library path variables are used by WebSphere to find the location in which DB2 is installed. WebSphere provides default variables at the node scope for newly created JDBC providers. These variables must be defined at the cell scope and accurately reflect where these libraries are installed on your system. This is done in the next section.

![Figure 6-18 Setting JDBC provider class path and native library path](image)

7. Click **Apply**.
8. Save and synchronize.

**Defining the JDBC variables**

We have to define two variables at the cell scope for the JDBC provider defined in the previous section. This assumes that DB2 is installed at the same location on all systems, or that the DB2 installation is shared across all systems in a data sharing Sysplex. Otherwise, the JDBC provider has to be defined at the node scope (that is, defined uniquely for each node). For our configuration we have to define two variables and their content:

- DB2UNIVERSAL_JDBC_DRIVER_PATH
- DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH

Follow this procedure to define the JDBC driver variables:

1. In the administrative console, go to **Environment → WebSphere Variables**.
2. Set the WebSphere Variables panel to cell scope.
3. The panel displays the current list of variables defined at the cell scope. Click **New**.
4. In the Name field, enter the first environment variable name, DB2UNIVERSAL_JDBC_DRIVER_PATH. In the Value field, enter the HFS location in which the db2jcc.jar file is located.

![Figure 6-19 Setting WebSphere environment variables for the JDBC provider](image)

5. Click **OK**.
6. Repeat the above to define another new variable.
7. In the Name field, enter the second environment variable name, DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH. In the Value field enter the HFS location where the driver library is located.
8. Save and synchronize.

**Deleting the DB2 variables at the node level**

Recall that two DB2 JDBC provider variables are defined, by default, at the node level by WebSphere. Variables defined at the node scope supersede variables defined at the cell scope we just defined. Therefore, we must remove these variables from each managed node and the DMGR node, as follows:

1. Navigate back to the WebSphere Variables panel.
2. Click **Browse Nodes**.
3. Select a node from the list and click **OK**.
4. Select the following variables from the list and click **Delete**:
   - `DB2UNIVERSAL_JDBC_DRIVER_PATH`
   - `DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH`
5. Click **OK**.
6. Repeat the above procedure for each managed node and the DMGR node.
7. Save and synchronize.

**Defining DB2 subsystem properties file**

Type 2 JDBC drivers sometimes need an additional property to point to the DB2 subsystem that it is connected to. The property is a JVM custom property defined for each servant. The property is used to point to a file in the HFS containing the DB2 subsystem ID. First we create the file, then the property.

**Creating the subsystem properties file**

Our ITSO configuration spans three z/OS images. Each image has its own DB2 subsystem that is part of a bigger DB2 data sharing group. The subsystem IDs for our ITSO configuration are shown in Table 6-1.

<table>
<thead>
<tr>
<th>z/OS image (node)</th>
<th>DB2 subsystem ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC49 (node A + Deployment Manager)</td>
<td>D8I1</td>
</tr>
<tr>
<td>SC50 (node B)</td>
<td>D8I2</td>
</tr>
<tr>
<td>SC54 (node C)</td>
<td>D8I3</td>
</tr>
</tbody>
</table>

Following the procedures and recommendations from *XD - Building the Basic Infrastructure*, do the following for each z/OS image:

1. Create an ASCII file at a convenient location in the HFS.
2. Set the permissions to 755 and the owner to the DB2 admin ID.

3. Edit the file by adding a property name followed by the appropriate DB2 subsystem ID from Table 6-1 on page 119 in the form:

   \[\text{db2.jcc.ssid=<DB2 Subsystem ID>}\]

4. Save the file.

**Creating servant JVM custom property to point to properties file**

A servant JVM custom property now has to be defined to point to the DB2 subsystem properties file generated above. This property has to be defined for the two LRS servants and the DMGR servant. For each of the LRS servants do the following:

1. In the administrative console, go to **Servers** → **Application Servers**. Click the link for one of the LRS application servers.

2. On the Application Servers configuration panel, go to **Server Infrastructure**, expand **Java and Process Management**, and select **Process Definition**.

3. On the Process Definition page, click the **Servant** link.

4. On the Servant General Properties page click the **Java Virtual Machine** link.

5. On the Java Virtual Machine configuration page, under Additional Properties, click the **Custom Properties** link.

6. On the Custom Properties page, click **New**.

7. Now we can enter the property definition. Set the Name field to \[db2.jcc.propertiesFile\]. Set the Value field to the HFS location where you stored the DB2 subsystem properties file on this z/OS image.

8. Click **OK**.
9. Repeat the above procedure for the other LRS servant.

10. Now we do the same for the DMGR servant, with a slightly different procedure. In the administrative console, go to System administration → Deployment manager.


12. Click the Servant link.


14. Click New.

15. Create the same property as above: Name field = db2.jcc.propertiesFile; and Value field points to the location of the DB2 subsystem properties file on the Deployment Manager z/OS image.

16. Click OK.

17. Save and synchronize.

**Defining and testing the LRS data source**

Now we create a data source for the LRS. Once again, this is a basic WebSphere configuration procedure presented here for completeness:

1. In the administrative console, select JDBC Providers. In the JDBC providers panel make sure it is set to cell scope.

2. Select the link for the JDBC provider created in the previous section, that is, DB2 Universal JDBC Driver Provider.
3. Under Additional Properties, click **Data Sources**.

4. Click **New**.

5. Here we enter the properties for the data source:
   a. Enter a name of your choice.
   b. Enter `jdbc/LRS` for the JNDI name.
   c. If you already have an appropriate component-managed authentication alias, select it from the menu. If not, you can create the alias in a separate step after you apply the data source definition.
   d. Under Db2 Universal data source properties, in the Database name field specify the name of the database previously created.
   e. We use a Type 2 driver. Select **2** from the Driver Type.
   f. The server name and port number are not required for a Type 2 driver.
   g. Click **Apply**.
h. Clicking **Apply** highlights the J2EE Connector Architecture (J2C) authentication data entries link under Related Items. If you did not have a component-manager authentication alias previously defined, click the link to define one now.

6. After you have defined the alias, if needed, go back to the LRS data source properties page and select the alias from the Component-managed authentication alias menu.

7. Click **OK**.

8. Save and synchronize.
9. We can test the data source connection by selecting the data source just created and click **Test Connection**. If all goes well, you should see the message shown in Figure 6-22 indicating a successful connection.

![Figure 6-22 Successful test connection message](image)

A failure indication would be in red with an explanation of the problem.

### 6.2.6 Deploying the Long-Running Scheduler (LRS) application

To summarize what has been configured so far in preparation for deploying the LRS application:

- A WebSphere Extended Deployment node group consisting of two nodes and a dynamic cluster defined for LRS. The application servers have been modified to adhere to naming standards and port allocation. This is the environment where LRS executes.
- A database table has been created for LRS to use for scheduling long-running jobs.
- WebSphere has been configured with the appropriate JDBC provider and data source for LRS.

**Installing the LRS application**

At this point, we are ready to install and test the LRS application. The install procedure has a few special considerations:

1. On the administrative console, go to **Applications → Install New Application**.
2. LRS is packaged in a J2EE EAR file and located in:

   `/<install-root>/DeploymentManager/ installableApps/LongRunningScheduler.ear`
3. Select **Remote File System**. Browse to the location of the LRS application and click **Next**. See Figure 6-23.

![Figure 6-23 Installing LRS: browsing to the location of the LRS application](image)

4. On the Preparing for the application install panel, perform the following steps:
   a. Check the **Generate Default Binding** check box at the top of the panel.
   b. Select the **Use Default Virtual Host** radio button and take the default name found in the box.
   c. Click **Next**.

5. If you get an Application Security Warning page, ignore it and click **Continue**.

6. For Step 1 make sure the **Deploy enterprise beans** check box is selected. Leave the other defaults. Click **Next**.
7. For Step 2 we map the two LRS application components to the LRS cluster. Check the box for the two LRS components, select the **LRS** cluster from the list under Clusters and Servers, and then click **Apply**. See Figure 6-24.

![Figure 6-24](image)

8. For step 3 we specify the database type and schema:
   a. From the Deploy EJB option - Database type menu, select **DB2UDBOS390_V8**.
   b. Set the Deploy EJB option - Database schema to **LRSSCHMA**, which is the default.
   c. Click **Next**.

![Figure 6-25](image)

9. Take the defaults for step 4 to step 8. Click **Next** for each panel to see the settings.

10. On the step 9 summary page, click **Finish**. The deployed code must be created, so this takes some time.

11. When the application is successfully installed, save and synchronize.
Configuring administrative console job management function
There is one last procedure before we start LRS. Perform the following steps:

1. In the administrative console, go to System Administration → Long-running scheduler.

2. Enter the following under General Properties:
   a. Set the database schema name to LRSSCHMA.
   b. Select jdbc/LRS from the Data source JNDI name menu.
   c. For the data source alias, select the alias defined earlier from the pull-down.

![General Properties]

Figure 6-26 Setting the long-running scheduler general properties

Starting the LRS application
Before continuing it is worth checking to make sure that the LRS application starts successfully:

1. On the administrative console, go to Servers → Application servers.
2. Select one of the LRS applications. Click Start to start the application server.
3. After a few minutes you should see a message indicating the result of the operation.
4. Stop the LRS application server. Then repeat this test for the other LRS application server.

6.3 Configuring long-running execution environment

This section describes how we configure the long-running execution environment in our ITSO environment. First, we present an overview of the configuration process, followed by the step-by-step procedures.
6.3.1 Overview

The configuration for the long-running execution environment (LREE) is similar to that just performed for LRS. We only focus on the specifics for LREE. The steps are:

1. Create a node group and dynamic cluster for LREE.
2. Perform additional configuration to support the LREE cluster.
3. Create the LREE database tables.
4. Define and test the LREE the data source.
5. Deploy the LREE application - LREE.ear.
6. Validate the configuration.
Our plan is to define a dynamic cluster consisting of three LREE application servers as the environment for executing long-running workloads. We plan to configure two application servers on node A and node C on z/OS LPAR SC49, and one application server on node B on z/OS LPAR SC50. Figure 6-27 shows our ITSO configuration with both the LRS and LREE configurations.

6.3.2 Creating a node group and dynamic cluster for LREE

A node group must be created for LREE, other than the default node group, before creating the LREE dynamic cluster.
Creating the LREE node group
Create the node group as follows:

1. In the administrative console, select **System administration → Node groups**.

2. The node groups display lists the current node groups currently defined at this point. Click **New**.

3. In the Node groups configuration screen, enter LREENodeGroup as the long name for the node group in the Name field, and enter a description of your choice.

4. Click **Apply**. Clicking **Apply** un-grays the node group members link. Click the link.

5. In the Node group members pane, click **Add**. This brings up a screen that lists all of the available nodes in our ITSO WebSphere cell. Check the nodes that you want to for the LREENodeGroup and click **Add**.

![Figure 6-28 Adding nodes to the LREENodeGroup](image)

6. This brings up a screen showing the selected nodes. Now click the **Save** link at the top of the screen. Then check **Synchronize changes with Nodes** and click **Save** to save the configuration.

Creating the LREE dynamic cluster
The LREE dynamic cluster is created in the LREENodeGroup as follows:

1. In the administrative console select **Servers → Dynamic Clusters**. This brings up a screen with a current list of dynamic clusters. In the Dynamic Cluster screen, click **New**.

2. This brings up Create new Dynamic Cluster screen Step 1.
   a. Enter LREECLUSTER in the Dynamic Cluster name field.
b. The Map to node group drop-down menu lists the currently defined node groups. Select **LREENodeGroup**. Click **Next**.

3. In step 2 we select a dynamic cluster template with the following steps:
   a. Specify the server template WebSphere uses to create application servers in the LRS dynamic cluster. Select **defaultXDZOS** from the Default application server template menu.
   b. Leave Existing application server un-selected since there are no LREE application servers defined at this point.
   c. Click **Next**.

4. The dynamic cluster-specific properties are specified in step 3, as follows:
   a. For the Minimum number of cluster instances section, select **Keep one instance started at all times**.
   b. In the Maximum number of cluster instances section, select **Do not limit the number of instances that can start**.
   c. Click **Next**.

5. At the Confirm Dynamic Cluster panel, click **Finish**.

6. Click **Save** and synchronize the changes.

At this point our configuration looks like Figure 6-27 on page 129. There is more LREE configuration work to be done, as described in the next sections.

**Additional configuration for the LREE dynamic cluster**

At this point we perform similar additional configuration work to support the LREE dynamic cluster, as we did for the LRS dynamic with the exception that we do not want to limit the number of instances to one. These steps are:

1. Re-map the server short names.
2. Re-map the cluster short name.
3. Enable the Mbean service.
4. Map the server ports.
5. Create the virtual host aliases for the LRS application servers.
**Re-mapping server short names**

Re-map the names of the LREE application servers following the same procedure as described in “Re-mapping server short names” on page 110. Choose a naming algorithm. We chose to do the following shown in Table 6-2.

<table>
<thead>
<tr>
<th>LREE application server name</th>
<th>Re-mapped short name</th>
</tr>
</thead>
<tbody>
<tr>
<td>LREECLUSTER_h6nodea</td>
<td>H6MWA1A</td>
</tr>
<tr>
<td>LREECLUSTER_h6nodeb</td>
<td>H6MWA1B</td>
</tr>
<tr>
<td>LREECLUSTER_h6nodec</td>
<td>H6MWA1C</td>
</tr>
</tbody>
</table>

**Re-mapping cluster short name**

Follow the procedure described in “Re-mapping cluster short name” on page 111 to re-map the LREE cluster short name. We set our LREECLUSTER short name to H6MWAA.

**Enabling the startup Mbean service**

Follow the procedure in “Enabling the startup Mbean service” on page 112 to enable the Mbean service for each of the LREE application servers.

**Mapping the server ports**

Follow the procedure described in “Mapping the server ports” on page 113 to map the server ports for each of the LREE application servers.

**Creating the virtual host aliases for the LRS application servers**

Follow the procedure described in “Creating the virtual host aliases for the LRS application servers” on page 114 to create virtual host aliases for each of the LREE application servers.

### 6.3.3 Creating the LREE database and tables

Creating the LREE database procedure is similar to the LRS database and is repeated here for completeness.

LREE requires a table to manage job state. This involves customizing and submitting some batch jobs. The ITSO environment uses DB2 V8 for z/OS. Our procedures follow the steps in the white paper *XD - Building the Basic Infrastructure*.
Default schema names
The table for LREE has a default schema name. You can use this name or chose your own. We chose to use the default.

Database creation script
WebSphere Extended Deployment for zOS comes with a sample database creation script for LREE:

- The script to create the LRS database and tables is located in the `../longRunning` directory for each node. The name of the script is SPFLREV7. Note that this script works for both DB2 V7 and V8.
- We used SPUFI to create the table. Copy the script from the HFS to a PDS for SPUFI.

DB2 subsystem properties file
A DB2 subsystem properties file was created for each z/OS image in “Creating the subsystem properties file” on page 119. All we must do here is to define a JVM custom property for the LREE servants.

Creating servant JVM custom property to point to properties file
Follow the same procedure as described in “Creating servant JVM custom property to point to properties file” on page 120 for each of the three LREECLUSTER application servers.

6.3.4 Defining and testing the LREE data source
We use the same JDBC provider configured in “Defining a DB2 JDBC provider” on page 115. Here we create a new data source for LREE with the following steps:

1. Follow the same predictor described in “Defining and testing the LRS data source” on page 121, except set the data source name to something like “lree” and set the JNDI name to jdbc/lree.
2. Apply, save, and synchronize.
3. Test the connection and view the message at the top of the panel to determine the status of the test.
6.3.5 Deploying LREE application

Now we can install the long-running execution environment (LREE) application. The install procedure has a few special considerations, as follows:

1. On the administrative console, go to **Applications → Install New Application**.

2. LREE is packaged in a J2EE EAR file and located in:
   
   `/<install-root>/DeploymentManager/installableApps/LREE.ear`

3. Select **Remote File System**. Browse to the location of the LRS application and click **Next**.

4. On the Preparing for the application installation panel, perform the following steps:
   
   a. Check the Generate Default Bindings check box at the top of the panel.
   
   b. Select the **Use Default Virtual Host** radio button and select **default_host** from the box.

5. For step 1, make sure that the **Deploy enterprise beans** check box is selected.

6. For step 2, map the two LREE modules to the LREECLUSTER dynamic cluster. Click **Apply**.

7. For step 3, perform the following steps:
   
   a. Make sure that the Deploy EJB option - Database type is DB2UDBOS390_V8.
   
   b. Set Deploy EJB option - Database schema to LREESCHM.

   ![Figure 6-29 Step 3: database type and database schema]

8. Take the defaults for step 4.

9. For step 5, perform the following steps:
   
   a. Scroll to the bottom of the page and select the check box next to the LREE EJB module - **BatchJobExecutionEnviornmentEJBs**.
b. Go back to the top of the page and set the JNDI name for the data source to be used, jdbc/lree, from the menu.

c. Click **Apply**.

If you scroll back to the bottom of the page you should see jdbc/lree under JNDI name for the EJB.

d. Scroll up a little and select the authentication alias from the Select authentication data entry menu.

e. Click **Next**.

10. For step 6, perform the following steps:

a. Scroll to the bottom of the panel and click the icon used to select all EJB modules form the list.

b. Scroll back to the top and select jdbc/lree JNDI name for the data source to use with these modules.

c. Click **Apply**.

d. Scroll back down to select the authentication alias.

e. Click **Next**.

11. Take the defaults from steps 7–10. If you get any security or resource warnings, ignore them by clicking **Continue**.

12. On the last page, click **Finish**. The deployed code must be created, so this takes some time.

13. When the application is successfully installed, save and synchronize.

**Starting the LREE applications**

Before continuing it is worth checking to make sure that all of the LREE application servers start successfully:

1. On the administrative console, go to **Servers → Application servers**.

2. Select one or more of the LREECLUSTER application servers. Click **Start**.

3. After a few minutes you should see a message indicating the result of the operation.

4. Make sure that you test each LREECLUSTER.

### 6.4 Testing the Long-Running Environment

You now have a long-running environment (LRS and LREE) configured on WebSphere Extended Deployment for zOS and you have verified that all of the LRE and LREE application servers and applications have started successfully. At
this point, you can install one or more of the sample long-running applications shipped with WebSphere Extended Deployment: SimpleCI, Mandelbrot, and postings. The instructions for doing this are documented in the Infocenter. SimpleCI is a simulated compute-intensive workload that makes a good installation verification test for the long-running environment. Mandelbrot is excellent for visually illustrating jobs executing in parallel in a multi-node LREE dynamic cluster. Postings is a multi-step batch workload similar to the application we introduce in the next chapter.
Long-running batch workloads

This chapter focuses specifically on long-running batch workloads in WebSphere Extended Deployment for z/OS. First, we introduce you to the sample long-running batch application and related utilities designed to illustrate several basic features of the WebSphere Extended Deployment long-running environment. Then there are several exercises demonstrating how to submit, track, and manage jobs in the long-running environment.
7.1 Long-running sample application overview

In this section we introduce the long-running application BatchTrade, used to demonstrate several features of the WebSphere Extended Deployment for z/OS long-running environment.

7.1.1 The BatchTrade application

The BatchTrade sample application is a long-running, multi-step, batch application that is a front end to existing business services (that is, stock trading services). The BatchTrade application, illustrated in Figure 7-1 on page 139, consists of a single EJB jar file packaged in a J2EE ear file, BatchTrade.ear.

The components of BatchTrade are:

- Step 1 - BatchGen: The first step optionally creates a file consisting of a randomly generated number of buy-stock and sell-stock trade records. Note that there is no input batch data stream to the BatchGen step. This was done to simplify the setup and demonstration of the BatchTrade sample. Step 1 generates the trade records for you and writes them to an output batch data stream. This stream is used as input to step 2.

- Step 2 - BatchTrade: The second step reads the batch input stream and processes the stock trade records and invokes the appropriate stock service (buy-stock or sell-stock via IIOP) for each trade record.
The stock-trading services are packaged in a J2EE application called TraderServices. The TraderServices application represents a traditional J2EE application consisting of presentation services in a Web container and an EJB jar file containing the stock-trading business logic. The long-running application invokes the business logic in the EJB jar. The TraderServices application is actually a modified version of the ITSO Trader application (not to be confused with Trade6) used in several previous WebSphere for z/OS IBM Redbooks. To simplify the WebSphere configuration for this book, only the JDBC connection is used to access the simulated stock resources in a database on DB2.

At this point, it is worth noting a simplification we made to the design and deployment of BatchTrade. As mentioned in “Batch application components” on page 30, each job step in a long-running batch job is represented by a CMP Entity EJB. Each CMP EJB requires a backing table in a relational database to persist the EJB’s data. Normally this requires that a relational database table be created for each step for each long-running batch job, as you would create a table for each CMP EJB. However, since the schema for each job step is identical, and the keys are unique, a single table can be deployed as the backing store for all job steps of a long-running batch job. This simplifies the deployment of multi-step batch jobs. We successfully used this technique for the BatchTrade application that is provided as a sample application in Appendix B, “Additional material” on page 197.

For simplicity, we also used the same database for the tables required by the BatchTrade and TraderServices applications.
7.1.2 BatchTrade xJCL

The xJCL for submitting BatchTrade jobs is provided in Appendix B, “Additional material” on page 197. The xJCL is annotated to provide a basic understanding of the specific information used to invoke BatchTrade. Several xJCL input parameters have been provided to allow you to change the operational characteristics of BatchTrade at run time, such as turn diagnostic tracing on or off, specify the file system to use for the batch data stream (HFS or z/OS data sets), define naming conventions for the BatchTrade input/output files, modify the checkpoint algorithm record intervals, and other operations. These input parameters are explained in more detail in 7.4, “Running BatchTrade” on page 146.

7.1.3 Submitting BatchTrade jobs

Once the LREE and LRS environments are configured and validated (see Chapter 6, “Establishing the long-running environment” on page 99), a long-running job’s xJCL can be submitted. The xJCL is submitted by command line, EJB API, and Web Services API. A 2EE application, WebServicesBatchConsole, included with the samples for your use, provides a Web interface that programmatically submits jobs to the LRS via either the EJB or the Web Services interface. Additionally, the Web interface provides screens to enable you to dynamically change xJCL input parameters before submission.

7.1.4 Monitoring and managing BatchTrade jobs

BatchTrade jobs are managed and monitored from the WebSphere Extended Deployment administrative console’s Job Management panel. In addition, the WebServicesBatchConsole sample included with this book contains Web utilities to manage and monitor the jobs programmatically submitted from the Web interface. These utilities are provided as a convenience for demonstrations and should not be considered for production use.
7.2 Installing the sample applications

Long-running applications are deployed into WebSphere Extended Deployment dynamic clusters configured to support the long-running execution environment (LREE). Once an LREE dynamic cluster is configured and validated, long-running applications, for example, BatchTrade, can be deployed into it. For this exercise we also installed the TraderServices OLTP application into the same dynamic cluster. The steps for installing and validating BatchTrade and TraderServices are first summarized as follows and then described in detail in the following sections:

1. Create the BatchTrade and TraderService tables and data source.
2. Install the BatchTrade and TraderServices into the LREE dynamic cluster.
3. Optionally, create a shared library for JZOS executables.
4. Create security for accessing batch data stream files.

7.2.1 Creating the tables and data source

As mentioned in 7.1.1, “The BatchTrade application” on page 138, BatchTrade requires several DB2 tables. These are listed in Table 7-1.

<table>
<thead>
<tr>
<th>Table</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRADER.TRADEBATCH</td>
<td>Used by LREE to persist batch job step state information for BatchTrade</td>
</tr>
<tr>
<td>TRADER.COMPANY</td>
<td>Used by TraderServices to manage a list of stock IDs</td>
</tr>
<tr>
<td>TRADER.CUSTOMER</td>
<td>Used by TraderServices to keep track of trades for customers</td>
</tr>
</tbody>
</table>

The DDL for creating these tables is included with the additional materials provided with this book. Also, there is sample DDL to prime the TRADER.COMPANY table with a set of stock IDs.

After you create the tables, create a data source for BatchTrade and TraderServices. The procedure is the same as always:

1. Go into the administrative console and locate the JDBC Driver we defined for the long-running environment (“JDBC provider definition” on page 115).

2. Create a new data source for this driver:
   a. Select a name of your choice, for example, TraderDB2.
   b. Set the JNDI Name to jdbc/TraderDB2.
c. Make sure that the **Use this Data Source in container managed persistence (CMP)** option is checked.

d. Select a Component managed authentication alias. We used the same alias as we did for the LRS and LREE components.

   All other fields are the same as the LRS and LREE data sources.

e. Apply, save, and synchronize.

f. Test the connection.

### 7.2.2 Installing the applications into the LREE dynamic cluster

The BatchTrade and TraderServices applications are installed into the LREE dynamic cluster we defined previously, LREECLUSTER, using the same procedure as any typical J2EE application. We highlight a few of the install steps below:

1. From the administrative console, go to **Applications → Install New Applications** and browse to where you stored BatchTrade.ear.

2. Proceed forward, accepting the defaults, to step 2.

3. At step 2, select all of the BatchTrade components. Under Clusters and Servers select **LREECLUSTER**. Click **Apply**. Click **Next**.

![Figure 7-2  Selecting target dynamic cluster for BatchTrade components](image-url)
4. In step 3, select **DB2UDOS390_V8.1** from the Current BackendId pull-down.

![Specify the selection for the BackendID](image)

**Figure 7-3  Selecting the BackendId for BatchTraderEJB**

5. In step 5 and step 6, make sure that the JNDI Name for all of the EJBs is set to jdbc/TraderDB.

6. Proceed through to step 12 Summary, accepting the defaults. If you get any application resource warnings, click **Continue** and move on.

7. At step 12, click **Finish**.

8. When the deployment is done, save and synchronize.

9. Repeat the previous steps for the TraderServices application.

### 7.2.3 Creating a shared library for the JZOS executables

The sample BatchTrade application is designed to use either the HFS (that is, the native Java file system on z/OS) or z/OS data sets for the output and input batch data streams (BDS) between step 1 and step 2. The target file system is selected via a xJCL parameter. If z/OS is specified as the target file system, then the BatchTrade application utilizes JZOS to access the z/OS data sets. You can find more information about JZOS on the Web at:

http://www.alphaworks.ibm.com/tech/zosjavabatchtk

The HFS is the default file system for the BatchTrade BDS. If you intend to utilize a z/OS data set as the BDS you must perform the following configuration steps. Otherwise, skip this section. We assume that you have already installed the JZOS product as per the documentation found at the above link. We also assume that JZOS is either installed in a shared HFS or installed at the same location on each LREE node.

You must configure a **shared library** to use JZOS for the BatchTrade application. Perform the following steps:

1. From the administrative console, go to **Environment → Shared Libraries**.
2. Set the Shared Libraries panel to cell scope. Click **New**.
3. In the General Properties panel, select an appropriate name for the shared library, for example, jzos, and enter the classpath and native library path where you installed JZOS. Figure 7-4 shows these fields for our ITSO configuration.

4. Click **Apply**. Save and synchronize.

![General Properties](image)

*Figure 7-4  Defining the JZOS shared library general properties*

5. Now we associate this shared library to the BatchTrade application. Go to **Applications** → **Enterprise Applications**.

6. Click the **BatchTrade** link.

7. Under Additional Properties, click the **Libraries** link.

8. In the Library reference panel, select **jzos** and click **Add**.

9. Click **Apply**. Save and synchronize.

You must re-associate a shared library to an application whenever the application is re-installed or updated.
7.2.4 Security for accessing BDS files

A few more details must be checked to ensure that BatchTrade can access the files associated with the BDS:

- Access to the HFS: Make sure that the file permissions are set to at least 775.

7.2.5 Invoking BatchTrade

At this point, you are ready to submit xJCL to run BatchTrade. BatchTrade takes several xJCL input parameters that modify its operational behavior at runtime. In order to demonstrate some basic characteristics of batch jobs in the long-running environment, we developed a utility WebServicesBatchConsole, introduced in the next section, to enable you to easily set these parameters and submit jobs.

7.3 BatchTrade test utility

The additional materials for this book include a J2EE Web utility application called WebServicesBatchConsole to simplify the submission, monitoring, and experimentation of BatchTrade jobs.

We used this application to validate the long-running execution environment and to demonstrate several features of the WebSphere Extended Deployment for z/OS long-running services. Some of the features of this Web utility are that it:

- Dynamically creates BatchTrade xJCL based on input parameters you select.
- Submits one or more BatchTrade jobs sequentially or at specified time intervals.
- Changes the checkpoint interval.
- Specifies the BDS file system, HFS or z/OS.
- Indicates how to dynamically create the BatchTrade records that are input to BatchTrade step 2.
- Monitors the status of the jobs you submitted.
- Invokes additional LRS functions such as cancel, suspend, resume, restart, and purge for the jobs you submitted.
- Dynamically sets trace options. You can use this to monitor the overall progress of a BatchTrade job and monitor all of the BDS and job step callbacks from LREE.
Specify overall operational parameters such as:
- The job scheduler interface to use: Web Services or EJB
- Job scheduler invocation: Web Services endpoint or IIOP URL
- Limits: maximum number of jobs to submit, job suspend time

Plus some general-purpose utilities such as:
- Upload, optionally edit, and submit any xJCL.
- Monitor and manage all jobs on the system (a Job Management Console).

### 7.3.1 Installing the WebServicesBatchConsole utility

The WebServicesBatchConsole utility is a J2EE application consisting of a single Web module. This utility is not designed to be clustered. Therefore, for simplicity, we installed the utility application in the LRS dynamic cluster. (You can deploy other J2EE applications in the LRS dynamic cluster provided they do not impact the LRS performance you require and you can tolerate only one instance of the application executing.)

This WebServicesBatchConsole utility application is self-contained (that is, it does not require any additional WebSphere configuration or resources). We recommend that you just install it in the LRS dynamic cluster and use it for running the tests and demonstrations in the following sections.

If you choose not to use this utility, you can write your own application using the LRS API, or write scripts that submit work via the `lrcmd` command. We provide sample BatchTrade xJCL as additional material that you can use, modify, or both, for this purpose.

### 7.4 Running BatchTrade

The section walks you through several demonstrations to familiarize you with the long-running environment of WebSphere Extended Deployment for z/OS. We use the WebServicesBatchConsole utility application. These demonstrations are:

1. Submit a single job to validate the functionality of the entire long-running environment and the BatchTrade application.
2. Submit multiple jobs at one time to observe when and where they are dispatched and executed.
3. Demonstrate checkpoint and restart.
7.4.1 Validating the long-running environment: submit a job

In this section, we submit a single BatchTrade job via the WebServicesBatchConsole utility to validate that the overall environment is functioning correctly and to illustrate some of the basics of the long-running environment.

Setting up for the long-running environment

Check for the following:

1. Make sure that one LRS application server is running in the LRS dynamic cluster. (Remember, the design of the long-running scheduler in WebSphere Extended Deployment V6.0.2 only allows one LRS instance to run at a time.)

2. Make sure that one LREECLUSTER application server is running. (Remember, the default operational mode for a dynamic cluster after it is created is manual. You must manually start one of the LREECLUSTER application servers.)

3. Make sure that the LongRunningScheduler, LREE, BatchTrade, TraderServices, and WebServicesBatchConsole enterprise applications are running.

The WebServicesBatchConsole utility overview

It is worth spending a few minutes getting familiar with the WebServicesBatchConsole utility we use to submit BatchTrade jobs.

1. From a browser enter the URL for the test utility:
   
   \[ http://<host>:<port>/Batch \]

   You can either go directly to the host and port of the LRS application server, or to the host and port of the ODR.
2. This brings up the main menu of the WebServicesBatchConsole utility. The first thing we must do is set a few operational parameters. Go to the Business Grid Demo Setup page.

![WebServicesBatchConsole utility main menu](Figure 7-5)

3. Here we select the interface the utility uses to invoke the long-running scheduler services, IIOP, or Web Services.
   a. If you select **IIOP**, in the Job Scheduler IIOP URL field enter the host name of the LRS application server followed by the well-known IIOP port - 900. For example:
      
      \[http://<LRS Application Server>:900\]
   
   b. If you select **Web Services**, in the Job Scheduler Web Services Endpoint field enter the LRS Web service URL:
      
      \[http://<host>:<port>/LongRunningJobSchedulerWebSvcRouter/services/JobScheduler\]
      
      The host and port are either for the ODR application server or the LRS application server.
   
   c. Click **Update Business Grid Variables** to save your settings. If you selected **IIOP**, the setup utility immediately attempts to get the EJB home of the LRS service and informs you whether the service is available.
d. Click the **Long-Running Job Console** link to go back to the main menu.

![Business Grid Demo Setup Utility](image)

**Figure 7-6** Specifying the WebServicesBatchConsole utility settings

4. New we are ready to run a job. From the main menu, click **Go** for BatchTrade.

5. This brings up the Batch Trade Job Console panel, as shown in Figure 7-7.

![Batch Trade Job Console](image)

**Figure 7-7** Batch Trade Job Console utility

We perform our long-running BatchTrade job tests and demonstrations from this panel. First, note all the options tables. These options control operational
characteristics for how the jobs are executed and may dynamically change the xJCL to be submitted. Also note the grayed-out submit buttons below the options table, cancel job, suspend job, and so on. These represent functions on the LRS API. As the long-running scheduler and execution environments process a long-running batch job, the job state changes as it passes from its initial state, submitted, to its final state, ended. You are only allowed to execute certain long-running scheduler functions in certain states. The Batch Trade Job Console utility has a finite-state machine built into it to only highlight those functions that you are allowed to invoke for a job in its current job state. This prevents LRS from throwing unwanted exceptions for state violations.

Table 7-2 is an explanation of the basic BatchTrade submit options.

<table>
<thead>
<tr>
<th>Batch Trade Job Console option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of BatchTrade jobs to submit</td>
<td>Self explanatory. The number of jobs submitted at one time when you click Submit Jobs.</td>
</tr>
<tr>
<td>Trace setting</td>
<td>Defines the log messages BatchTrade components write to the sysout log.</td>
</tr>
<tr>
<td></td>
<td>- NONE: Only errors and warnings.</td>
</tr>
<tr>
<td></td>
<td>- TRACE: The above plus Job/step start and end messages (default).</td>
</tr>
<tr>
<td></td>
<td>- DIAG: The above plus a log message for each method call and other interesting events. Use this if you want to monitor the order in which LREE invokes various BDS and job step callback methods.</td>
</tr>
<tr>
<td>Batch input: file system type</td>
<td>Specify the file system used for the BatchTrade input and output streams:</td>
</tr>
<tr>
<td></td>
<td>- Native: Native Java file system (that is, HFS on z/OS - default)</td>
</tr>
<tr>
<td></td>
<td>- z/OS: z/OS data sets via JZOS</td>
</tr>
<tr>
<td>Record-based checkpoint algorithm interval: record count</td>
<td>BatchTrade xJCL is defined to use the record-based checkpoint algorithm for BatchTrade step 2. This field specifies the number of trade records processed by step 2 before a checkpoint is taken by LREE.</td>
</tr>
</tbody>
</table>
BatchTrade Step 1 can be configured to automatically generate trade record output streams for you, as a convenience, to feed into BatchTrade step 2. If you plan to submit multiple BatchTrade jobs you must use this feature. A unique name is required for each file or data set that contains the trade records associated with each batch job. Otherwise, the batch jobs will be reading each other's files. BatchTrade step 1 can automatically generate trade record files with unique file or data set names. You select the file path and file prefix name and BatchTrade step 1 appends the unique LRS assigned job ID (job name + job number) as the file name suffix, to generate a unique BDS file. The auto-generated file names for the supported environments are shown in Table 7-3.

<table>
<thead>
<tr>
<th>Batch Trade Job Console option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set a time delay between job submissions</td>
<td>You can stagger the submission of multiple jobs by specifying a time (ms) delay between each job submission to LRS. Note: If the total submission time is deemed to be lengthy, the jobs are submitted asynchronously, thereby freeing the browser to view job status.</td>
</tr>
<tr>
<td>Job list (status, cancel, suspend, resume, restart, or purge)</td>
<td>This is actually an output field. It is a pull-down list of all of the jobs you currently submitted in this HTTP session. These are the jobs upon which you can perform LRS operations. Jobs remain on this list until purged or the HTTP session is invalidated.</td>
</tr>
</tbody>
</table>

BatchTrade Step 1 can be configured to automatically generate trade record output streams for you, as a convenience, to feed into BatchTrade step 2. If you plan to submit multiple BatchTrade jobs you must use this feature. A unique name is required for each file or data set that contains the trade records associated with each batch job. Otherwise, the batch jobs will be reading each other's files. BatchTrade step 1 can automatically generate trade record files with unique file or data set names. You select the file path and file prefix name and BatchTrade step 1 appends the unique LRS assigned job ID (job name + job number) as the file name suffix, to generate a unique BDS file. The auto-generated file names for the supported environments are shown in Table 7-3.

**Table 7-3 BatchTrade automatically generated BDS file or data set name format**

<table>
<thead>
<tr>
<th>Environment</th>
<th>File Name Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX (HFS) BatchTrade files</td>
<td><code>/&lt;path&gt;/&lt;prefix&gt;_&lt;Job Name&gt;_&lt;Job Num&gt;</code></td>
</tr>
<tr>
<td>z/OS BatchTrade data sets</td>
<td><code>//&lt;prefix&gt;.&lt;Job Name&gt;.&lt;Job Num&gt;</code></td>
</tr>
<tr>
<td>Windows® BatchTrade files</td>
<td><code>\&lt;path&gt;\&lt;prefix&gt;_&lt;Job Name&gt;_&lt;Job Num&gt;</code></td>
</tr>
</tbody>
</table>

In addition, for each unique file created, the utility generates a random number of trade records. Each trade record consists of a randomly selected stock ID from the TradeDB database, a randomly generated customer ID, a randomly selected buy or sell order, and a randomly selected number of stocks to buy or sell. This is done for variability.
Table 7-4 shows you how to tell the utility to automatically generate unique files for BatchTrade.

<table>
<thead>
<tr>
<th>Batch Trade Job Console option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auto generate: batch input file path</td>
<td>Specifies the file system path where the trade files are to be generated. The format is:</td>
</tr>
<tr>
<td></td>
<td>▶ Win: Drive:\dir\</td>
</tr>
<tr>
<td></td>
<td>▶ UNIX: /path/ (Native file system only)</td>
</tr>
<tr>
<td>Auto generate: batch input file prefix name</td>
<td>The prefix or starting characters for each BatchTrade file or data set.</td>
</tr>
<tr>
<td>Auto generate: random number of input records</td>
<td>Specifies the upper bound for generating a random number of trade records for each file or data set.</td>
</tr>
<tr>
<td>Auto generate: random number of user IDs</td>
<td>Specifies the upper bound for generating a random number of user IDs associated with a trade record.</td>
</tr>
<tr>
<td>Auto generate: delete input files upon job</td>
<td>When set to yes, the BatchTrade job deletes the auto-generated file containing its trade records when the job successfully completes.</td>
</tr>
<tr>
<td>completion</td>
<td></td>
</tr>
<tr>
<td>Fixed batch input file</td>
<td>Specifying a specific file location of the batch input file in this field overrides all of the above auto-generate features.</td>
</tr>
<tr>
<td></td>
<td>Use this option when you want to create your own BatchTrade input file for testing purposes, for example, for introducing errors,</td>
</tr>
<tr>
<td></td>
<td>validating checkpoints, and so on. Make sure that the number of jobs is 1.</td>
</tr>
<tr>
<td></td>
<td>Format example:</td>
</tr>
<tr>
<td></td>
<td>▶ Win: Drive:\dir\InputFile.txt</td>
</tr>
<tr>
<td></td>
<td>▶ UNIX: /path/InputFile.txt</td>
</tr>
</tbody>
</table>
Submitting a job to test the long-running environment

Now we submit one BatchTrade job to test the long-running environment:

1. On the Batch Trade Job Console screen, click **Defaults**. Set the following overrides (refer to Figure 7-7 on page 149):
   a. Set the batch input file path and batch input file prefix names.
   b. Set **Delete input files upon completion** to **No** so that you can later view the auto-generated trade record input file.

2. Click **Submit Jobs**.

3. If the submission is successful, the Batch Trade Job Console screen looks similar to Figure 7-8. Note the following:
   - The message returned from the LRS submit job API is displayed indicating that the job is now in the submitted state. The message format changes as the job proceeds from state to state. The basic message format is:

```
MessageID:[Data&Time][Job ID][job state][Job Type][][Node][LREE Server]
```
   - The job ID is added to the Job List pull-down.
   - Several grayed-out submit buttons now become visible, indicating the operations you can perform on the submitted job (for example, Job Status, Cancel a Job).

![Figure 7-8 Successful submission of a BatchTrade job](image)

4. You can follow the status of a job displayed in the Job List pull-down by clicking **Job Status** periodically. Note how the Submit button's perspective changes as the job's state changes.

5. Once the job successfully completes, the state is *ended* and it can be purged. Clicking **Purge Job** removes the job from the LREE job database and from the Batch Trade Job Console Job List.
6. You can follow the progress of long-running jobs via SDSF by viewing the SYSOUT for the servant region that ran the job. The top of Figure 7-9 shows a typical message LREE writes to SYSOUT as it processes jobs. The bottom shows TRACE messages written by the BatchTrade job.

![Figure 7-9  SYSOUT messages from LREE (top) and BatchTrade (bottom)](image)

7. To completely check out the long-running environment you may want to repeat this test for each LREECLUSTER application server in the dynamic cluster.

### 7.4.2 Submitting multiple jobs

Now we submit several BatchTrade jobs at once to observe the behavior of the long-running environment and introduce other WebSphere Extended Deployment job management facilities.

#### Setting up for the long-running environment

Check for the following:

- Make sure that one LRS application server is running in the LRS dynamic cluster. (Remember, the design of the long-running scheduler in WebSphere Extended Deployment V6.0.2 only allows one LRS instance to run at a time.)

- Make sure that all LREECLUSTER application servers are running. We want to observe when and where the long-running environment schedules and executes jobs.

- Make sure that the LongRunningScheduler, LREE, BatchTrade, TraderServices, and WebServicesJobConsole enterprise applications are running.

- Optionally, purge all previous jobs to give you a clean environment.
Submitting the jobs
To submit the jobs:

1. From the Batch Trade Job Console, perform the following steps:
   a. Set the number of BatchTrade jobs to the number you want to submit, for example, 10.
   b. Leave the auto-generate fields as they were from the previous run.
   c. Optionally, you can set Delete input files upon completion to Yes if you want to preserve the trade record input files.

2. Click Submit Jobs.

3. If successful, you see a screen similar to Figure 7-10. The Job List pull-down contains a list of all of the jobs submitted. If you select a job from the Job List pull-down and click Job Status, you see the current state of that job via the LRS message. The non-grayed-out submit buttons reflect the operations you are permitted to perform on a job depending on its current state.

WebSphere Extended Deployment job management
The status of submitted jobs is normally viewed and managed from the WebSphere Extended Deployment administrative console’s Job Management panel by, from the administrative console, going to Runtime Operations → Job
Management. This brings up the Job Management panel. Figure 7-11 shows how the Job Management panel looks after the 10 jobs were submitted in the previous section.

![Administrative console Job Management panel](image)

*Figure 7-11 Administrative console Job Management panel*

You use the Job Management panel to determine the status of jobs and where they are executing, and perform various operations such as cancel, suspend, remove, and so on depending on the job’s state. On this panel you can also see when and on which node and application server in the LREE dynamic cluster each job is, or has, executed.
7.4.3 Checkpoint-restart demonstration

The long-running execution environment uses checkpoint algorithms to determine when to commit global transactions for each batch step. The xJCL defined for the batch job specifies which checkpoint algorithm to use and the checkpoint interval. (Refer to “Checkpoint algorithms” on page 33.) The BatchTrade sample application uses the record-based checkpoint algorithm provided by WebSphere Extended Deployment.

The Batch Trade Job Console panel of the WebServicesBatchConsole utility allows you to set the record count interval to determine when the checkpoint is invoked. The larger the number, the longer the interval between transaction commits and the longer locks are held on resources. If you set trace to DIAG you can view the SYSOUT in SDSF to verify when the checkpoint is triggered.

A long-running batch job becomes restartable either when it is explicitly suspended or a failure occurs during execution. When a failure occurs, the operator can intervene, attempt to fix the problem, and restart the job. The job picks up processing from the last checkpoint. All updates to resources (for example, DB2, CICS) up to the checkpoint are committed. Updates that occurred between the last checkpoint and a failure are backed out.

The checkpoint-restart scenario demonstrated in this section does the following:

1. An input file of BatchTrades is manually created. An error causing BatchTrade to fail is explicitly introduced into one of the records.
2. The record interval is set to trigger a checkpoint before BatchTrade reads the failed record.
3. The job is submitted, the failure occurs, and the job is placed in restartable state and terminates.
4. The database is checked to verify that the records processed before the checkpoint caused the database to be updated.
5. The trade record with the error is corrected. The job is restarted and runs to completion.
6. The database is checked to verify that only the records processed after the last checkpoint were processed.

Setting up for the long-running environment

Check for the following:

- Make sure that one LRS application server is running in the LRS dynamic cluster.
Make sure that one LREECLUSTER application servers is running. We want to observe when and where the long-running environment schedules and executes jobs.

Make sure that the LongRunningScheduler, LREE, BatchTrade, TraderServices, and WebServicesJobConsole enterprise applications are running.

Checkpoint-restart demonstration

Perform the following steps to run the demonstration:

1. Manually create an input file consisting of several BatchTrade records as follows. One trade record is to contain an error that causes BatchTrade to fail:
   a. Figure 7-12 shows the record format and the trade records we created for the demo. We designed a simple comma-delimited format for a trade record:
   
   ```plaintext
   [Buy or Sell],[Customer ID],[Stock ID],[Num Shares]
   ```
   b. Make sure that you select customer IDs not already in the database to make verification easier. Select a stock ID that is the database.
   c. We injected an error in line 5 by setting the Num Shares field to a non-numeric value. BatchTrade is designed to stop processing when this situation occurs.
   d. Save the file (in an HFS file or z/OS data set).

```
****** **************************** ******************************************
==CHG> B,raj6,Casay_Import_Export,69
000002 B,raj1,Glass_and_Luget_Plc,42
000003 B,raj2,Glass_and_Luget_Plc,42
000004 B,raj3,Glass_and_Luget_Plc,42
000005 B,raj4,Glass_and_Luget_Plc,xx
000006 B,raj5,Glass_and_Luget_Plc,42
000007 B,raj6,Glass_and_Luget_Plc,42
000008 B,raj7,Glass_and_Luget_Plc,42
****** **************************** ******************************************
```

Figure 7-12  BatchTrade input records for checkpoint-restart demonstration
2. Go into the Batch Trade Job Console panel of the WebServicesBatchConsole utility. Click **Defaults** to clear previous settings, remove all previous jobs (if any) from the Jobs List, and set the fields shown in Table 7-5.

**Table 7-5  Options and functions**

<table>
<thead>
<tr>
<th>Batch Trade Job Console option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of BatchTrade jobs</td>
<td>Set to 1 to control the test.</td>
</tr>
<tr>
<td>File system type</td>
<td>Set to the file system type you used to save the trade records.</td>
</tr>
<tr>
<td>Trace</td>
<td>Set to DIAG if you want to view when the checkpoint callbacks are invoked in SYSOUT.</td>
</tr>
<tr>
<td>Checkpoint interval record count</td>
<td>Set the record interval to some count before the failed record. Example: We created a failure in record 5, so we set the checkpoint interval to 4.</td>
</tr>
<tr>
<td>Fixed batch input file</td>
<td>Set to the absolute location where you saved the trade records file. Setting this field cause BatchTrade step 1 to bypass the auto-generate input file facility and read in this file.</td>
</tr>
</tbody>
</table>

3. Click **Submit Jobs**. You immediately see the reply message from LRS indicating that the job is in submitted state.

4. Periodically click **Job Status** to view the job status as it transfers from one state to another. Eventually it reaches the restartable state and stops processing. Note that the Restart a Job submit button is now highlighted.

5. Now we look at the database to verify what updates have been made. Using SPUFI, or a tool of your choice, issue the following SQL query to determine which trade records were processed:

   ```sql
   select * from trader.customer where customer like 'raj%'
   ```

   You should only see results for customer IDs contained in trade records 1 through 4.

6. Edit the trade record file created above. Fix the bad record by changing the Num Shares field to a numeric value. Save the file.

7. Go back to the Batch Trade Job Console panel. Click **Restart a Job**.

8. You immediately see the reply message from LRS, indicating the job is in submitted state, indicating that LRS is in the process of re-starting the job.
9. Periodically click **Job Status** to view the job status as it transfers from one state to another. The job eventually reaches the ended state when it successfully completes.

10. Look at the database again to verify the updates. Issue the same SQL query entered in 5 on page 159 above. You should see results for all customer IDs from the trade records. The data for the customer IDs in trade records (1 through 4) up to the checkpoint remain unchanged. There are now entries for customer IDs from trade records 5 through 8 indicating the appropriate number of stocks.

### 7.5 Some observations and lessons learned

The WebSphere Extended Deployment long-running environment is very new. Undoubtedly these services are going to mature and the programming model will evolve. As more long-running production applications are deployed, best practices emerge and are refined, and more product requirements are identified. It is also conceivable that best practices that emerge may differ between z/OS and distributed systems.

Nevertheless, during the course of our developing, deploying, and executing applications in the long-running environment, we have made some interesting observations and learned a few lessons. These are by no means conclusive or absolute. We list them here as a getting started now with long-running batch aid.

**Note:** These observations and recommendations are applicable only to WebSphere Extended Deployment for z/OS V6.0.1, and have not yet been verified on a distributed platform.

#### 7.5.1 Backing datastore for long-running batch job steps

“Batch application components” on page 30 introduced you to the components and programming model for a long-running batch application. Recall that each step of a batch application is implemented as a CMP entity EJB to enable LREE to persist job information. This means that every step for every batch job must be mapped to a relational database table. We noted that the required LREE schema for each job step is identical and the keys are unique. We took advantage of this when creating a backing table for BatchTrade, as described in 7.1.1, “The BatchTrade application” on page 138.

We created a single DB2 table with the required schema for both BatchTrade job steps. Using the EJB mapping wizard of Rational® Application Developer, we
mapped each job step EJB to a DB2 single table with the appropriate schema. This simplified the deployment of the BatchTrade application.

A batch application may have requirements such that a job step would need to persist application-specific information. In theory, additional fields could be defined in the LREE database table to do this, rather than creating yet another table. In such situations, the single table for multiple job steps may not be a wise implementation decision. However, the design point for the BatchTrade application was only to provide a batch facade to existing enterprise services. Therefore it did not require persistent data to meet its own functional requirements.

### 7.5.2 Batch data streams (BDSs) and bean cache setting

When multiple batch jobs are submitted, several jobs may execute concurrently in a given LREE servant region (JVM) depending on the dynamic cluster configuration. We noticed some unexpected behavior (basically a batch job failing with a null pointer exception) when we submitted a large number of BatchTrade jobs, which resulted in several jobs running concurrently. Before describing the observed behavior, it is worth going into a little more technical detail about the long-running environment and batch programming model.

Recall from “batch data streams” on page 31 that a batch job step may have one or more input or output batch data streams, or both, for reading and writing data. The LREE instantiates a BDS object from the class specified in the xJCL, and provides a service for a batch job step EJB to obtain the instantiated BDS object. The job step EJB calls methods on the BDS object to read or write records. The BDS object keeps track of the number of records read or written in the event of restarts. The BatchTrade application (designed by simply following the methodology in the multi-step postings sample application provided with WebSphere Extended Deployment) saves the BDS object in an instance variable for use by the job step EJB as it iterates through each input or output record. This is not a good design since there are transactions associated with each job step.

Job step EJBs run under a transaction started by the batch controller bean, as described in “Batch controller bean” on page 33. The checkpoint properties on the job’s xJCL specify the transaction boundaries, as described in “Checkpoint-restart demonstration” on page 158. When a checkpoint occurs, a transaction is committed. The default behavior of an entity EJB in WebSphere, at this juncture, is to call the ejbPassivate() method. The ejbCreate() method is called when the bean is invoked again to continue processing of the next set of records from the input stream. After the ejbPassivate() method is called, the bean instance is returned to the EJB pool. When the ejbCreate() method is called the bean is returned from the pool. There is no guarantee that the same instance variables are returned to the bean when an EJB is returned from the pool. The
EJB specification mentions this behavior very clearly. Only the variables that represent the entity bean’s persisted data (job ID and step ID) are set to their correct state. The EJB should re-initialize instance the variables.

Therefore, with a large number of jobs submitted, several running concurrently, we observed seemingly random situations where a BDS instance variable was not restored to an EJB after ejbCreate(), resulting in either a null pointer exception or an incorrect BDS object. One obvious fix would be for the job step EJB to obtain the BDS instance from LREE for each record (that is, each invocation of the processJobStep() method), or when ejbCreate() is called. We did this but found that a new BDS instance was returned rather than the existing BDS instance with its current state information.

We eventually discovered that by manipulating the Bean Cache setting on the deployment descriptor for the BatchTrade Step 2 CMP entity bean, we achieved the desired behavior. The Rational Application Developer Infocenter states the following for defining Bean Cache settings. See Figure 7-13.

---

<table>
<thead>
<tr>
<th>4. In the <strong>Activate</strong> at field, select one of the following values to specify the point at which an enterprise bean is activated and placed in the cache.</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>ONCE</em>: Indicates that the bean activates when it is first accessed in the server process, and passivates (and is removed from the cache) at the discretion of the container, for example, when the cache becomes full. If you select to activate at ONCE, then all five of the options listed below are available.</td>
</tr>
<tr>
<td><em>ACTIVITY_SESSION</em>: Indicates that the bean activates and passivates as follows: 1) On an ActivitySession boundary, if an ActivitySession context is present on activation. 2) On a transaction boundary, if a transaction context (but no ActivitySession context) is present on activation, or otherwise. 3) On an invocation boundary.</td>
</tr>
<tr>
<td><em>TRANSACTION</em>: Indicates that the bean activates at the start of a transaction and passivates (and is removed from the cache) at the end of the transaction.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>5. In the <strong>Load at</strong> field, select one of the following values to specify when the bean loads its state from the database. The value of this setting implies whether the container has exclusive or shared access to the database:</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>ACTIVATION</em>: Indicates that the bean loads when it is activated (regardless of Activate at setting) and implies that the container has exclusive access to the database.</td>
</tr>
<tr>
<td><em>TRANSACTION</em>: Indicates that the bean loads at the start of a transaction and implies that the container has shared access to the database.</td>
</tr>
<tr>
<td><em>INTERVAL</em>: (For EJB 2.x only) Indicates that the bean loads at intervals, determined by the integer set in the Load at interval field.</td>
</tr>
<tr>
<td><em>DAILY</em>: Indicates that the bean loads its state on a daily basis.</td>
</tr>
<tr>
<td><em>WEEKLY</em>: Indicates that the bean loads its state on a weekly basis.</td>
</tr>
</tbody>
</table>

---

*Figure 7-13  Bean Cache settings as documented from RAD V6 Infocenter*
When the Activate at is set to Once, the container does not render the EJB passive when a transaction occurs. With Load at set to Activation, the container has exclusive access to the database. By design, no other application modifies rows within the BatchTrade CMP backing datastore, so we are guaranteed that EJB variables are correct. We set these values in the deployment descriptor for CMP representing BatchTrade Step 2 using the RAD deployment descriptor wizard.

![Figure 7-14 Setting Bean Cache values in the RAD deployment descriptor Bean view](image)

### 7.5.3 Behavior of LREE dynamic cluster

In our ITSO environment, we configured the LREE dynamic cluster with three nodes. (Refer to Chapter 6, “Establishing the long-running environment” on page 99.) The operational mode of a dynamic cluster is set to manual by default. We studied the operational characteristics of this cluster in both manual and automatic modes.

#### LREE dynamic cluster in manual mode

Before describing the observed behavior of long-running work in the LREE dynamic cluster, it is worthwhile to explain some of the WebSphere Application Server properties and the default settings that govern the behavior. Listed below are some operational properties and their defaults. Refer to the WebSphere Application Server for z/OS Info Center at the following URL for more information about these and other properties, and how to set them:

http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/

- The com.ibm.websphere.longrun.EndPointCapacity custom property is set to 2, indicating that two jobs can run concurrently per servant.
- The minimum and maximum number of servant regions is set to 1.
- When the LREE application is installed, the server_use_wlm_to_queue_work is set to 0 and the server_work_disrtibution_algorithm is set to 1. With these settings WLM for z/OS does a round-robin. WLM goes through all of the threads in the first servant before moving to the threads in the next servant.
We set the minimum number of servant regions to 2 and manually started all three application servers in the LREE dynamic cluster. Since the com.ibm.websphere.longrun.EndPointCapacity custom property was set at 2 (default), only two long-running jobs can run concurrently in an application server irrespective of the number of servants. We observed that the long-running jobs were dispatched by WLM to multiple servants in a round-robin fashion.

**LREE dynamic cluster in automatic mode**

Using the administrative console, we set the LREE dynamic cluster to automatic mode and ran several tests.

- **Test 1:** At the start of the test, the “Minimum number of cluster instances” was set to “Keep one instance started at all times” for the LREE dynamic cluster. The com.ibm.websphere.longrun.EndPointCapacity custom property was set to 10, indicating that ten jobs can run concurrently per servant. By default the minimum and maximum number of servant regions is set to 1. This means that 10 jobs can run concurrently per servant. We submitted 100 BatchTrade jobs and observed that the LREECLUSTER application server automatically started on other two nodes in the dynamic cluster to handle the load.

  This is expected behavior. The LRS application receives WLM statistics and makes a decision to start LREECLUSTER application servers on other available nodes in the dynamic cluster as required.

- **Test 2:** We set the minimum number of servants to 1 and the maximum number of servants to 5 and re-ran the above test. We noticed the same behavior as above (that is, as load increased, additional LREECLUSTER application servers were started on other dynamic cluster members). However, there were no additional servants started in any of the nodes to handle the load.

  This was not the expected behavior. The WebSphere Extended Deployment development team has been made aware of this situation.

- **Test 3:** We set the minimum and maximum number of servants to 5 and re-ran the test. Jobs were dispatched across all five servants and additional LREECLUSTER application servers were started as required.

In summary, LREE application servers were automatically started but the servants were not. A recommended practice for this release is to ensure that the minimum and maximum number of servants is set to the same number, and that the total number of servants defined across the LREE dynamic cluster must represent the maximum required capacity.

**Automatic versus manual mode**

The current implementation of WebSphere Extended Deployment for z/OS starts LREE application servers in a dynamic cluster automatically when set to
automatic mode. However, is this really the best way to manage dynamic clusters for long-running workloads? Considering the time that it takes to bring up a control region and servant region, it may be better to leave the LREE dynamic cluster definition in manual mode, start the servers depending on workload requirements, and let WLM dispatch the work.

### 7.5.4 Transactional considerations

Describing best practices relative to handling transactions in a long-running batch job is not an objective of this book. However, there is one observation we mention here for your consideration. Experienced J2EE designers are familiar with J2EE OLTP transaction constructs such as container transaction boundaries, isolation levels, access intent, and so on and know how to configure an infrastructure to handle multiple instances of J2EE applications in clusters to meet expected performance and capacity requirements. Many best practices for the above must be re-considered for a long-running batch environment.

Recall that BatchTrade is designed as a batch front end (batch facade) to stock trading services deployed in an existing J2EE application. We considered this to be one design pattern that may be of interest to early adopters of a WebSphere Extended Deployment business grid. The design and deployment of long-running J2EE applications now must take into consideration some of the principles learned in the traditional MVS™ batch environment such as when is the best time to schedule batch jobs, how much concurrent batch job processing can be tolerated without causing deadlocks, what are the optimum checkpoint intervals, and so on. The designer must also explore other considerations when batch jobs access existing services (via IIOP, Web Services, JMS, and so on) resident in J2EE applications designed for OLTP or SOA (for example, what are the transaction requirements of the existing service, what are the trade-offs relative to various checkpoint intervals at the batch end, is there a need for compensating logic in the batch component in the event of a service failure between checkpoints, should the existing application provide a new service interface for batch access to mitigate some of these issues, and so on).
Running mixed workloads

This chapter illustrates the WebSphere Extended Deployment for z/OS capability for running mixed J2EE workloads in dedicated and shared environments within a WebSphere cell.
8.1 Mixed workloads

One of the differentiating features of z/OS is its ability to simultaneously run and manage mixed workloads. The workloads may be combinations of various batch or OLTP applications. The workloads are assigned service classes and priorities enabling WLM to manage them. Usually OLTP workloads are assigned to a high priority and batch workloads to a lower priority. Batch workloads are usually run during off-peak times.

WebSphere Extended Deployment for z/OS brings this traditional z/OS capability to the J2EE paradigm, providing the ability to run both OLTP and long-running (batch-like) workloads on the same WebSphere z/OS resources. The value of this capability should grow in importance over time as more long-running workloads are deployed, and more business functions are deployed in an SOA environment as services. The Internet never sleeps. Many OLTP workloads are on a 24-hour clock, thereby extending these workloads into the traditional batch window. It is also conceivable that both OLTP and long-running workloads will need to consume the same services deployed in a SOA. Therefore, the ability to prioritize and manage mixed workloads and the ability to apply these priorities to shared services becomes critical.

WebSphere Extended Deployment for z/OS is differentiated from its distributed counterpart by supporting mixed workloads on the same nodes within different clusters, or on the same nodes within the same cluster. WebSphere Extended Deployment's integration and synergy with z/OS WLM makes this possible. On distributed platforms, WebSphere Extended Deployment manages the assignment of resources to various OLTP and long-running workloads by starting and stopping dynamic cluster members within the dynamic clusters associated with the workloads in question. To properly allocate resources within this scheme in distributed environments, the LREE application server never shares a dynamic cluster with an OLTP application server and WebSphere Extended Deployment does not start LREE and OLTP cluster members on the same nodes.

Since WebSphere Extended Deployment on z/OS relies on z/OS WLM, IRD, and PRSM to accomplish workload management and resource allocation, this isolation of long-running and OLTP workloads into separate clusters is not required. In this chapter we investigate various scenarios in which we combine (that is, mix) long-running and OLTP workloads. The three scenarios discussed are:

- Mixed workloads on the same nodes and separate dynamic clusters
- Mixed workloads in the same dynamic cluster
- Mixed workloads in the same dynamic cluster sharing business logic
8.2 Mixed workloads on same nodes and separate dynamic clusters

Up to this point we have discussed and demonstrated OLTP and long-running workloads executing separately in their own dedicated dynamic clusters. Although they shared the same nodes, they were not run concurrently. In this section we run them concurrently.

8.2.1 Setting up the environment

If you have followed all of the scenarios and exercises in the previous chapters, you already have an environment in which to run mixed OLTP and long-running workloads on the same nodes and dedicated clusters. Figure 3-3 on page 46 shows the completed infrastructure we used for this scenario.

8.2.2 Executing the workload

Chapter 5, “Online transaction processing (OLTP) scenario” on page 61, and Chapter 7, “Long-running batch workloads” on page 137, describe how to set up and run the OLTP and long-running workloads, respectively. We used the same applications and load generation tools as described in those chapters. We used the same service policies for the OLTP workloads. We used all of the defaults for the long-running workloads.

Rather than repeat those activities here, we encourage the reader to reference these chapters for instructions to simultaneously run both of the OLTP and long-running sample applications in their own environment.

8.2.3 Observations

With WebSphere Extended Deployment for z/OS, OLTP workloads have service class policies that have a TCLASS assigned to them. This is propagated to WLM on z/OS by the ODR, and WLM maps it to a service class that is defined to WLM. Refer to Chapter 5, “Online transaction processing (OLTP) scenario” on page 61. The current release of WebSphere Extended Deployment for z/OS does not assign service class polices to long-running workloads. Long-running workloads run under the default transaction class and service class that has been defined for WebSphere in the Sysplex.

We ran the OLTP and long-running workloads simultaneously under moderate load and observed that each workload behaved as previously described in Chapter 5, “Online transaction processing (OLTP) scenario” on page 61, and Chapter 7, “Long-running batch workloads” on page 137. We also observed that
that as the load increased, the service class goals were enforced relative to each other, with platinum and gold being favored over silver and bronze. All OLTP requests were favored over the batch steps represented by the WAS default service class.

8.3 Mixed workloads in the same dynamic cluster

In the next scenario we deployed the same XDStock application used in Chapter 5, “Online transaction processing (OLTP) scenario” on page 61, to the dynamic cluster that already contained the LREE used in Chapter 7, “Long-running batch workloads” on page 137. One of the implications of this environment is that our OLTP application had to run in a cluster that had certain properties related to z/OS WLM dispatching customized for the needs of the LREE. We will describe those properties in this section along with some of their considerations and ramifications.

We also took advantage of the edition management feature of WebSphere Extended Deployment to avoid having to redefine all of the service policies and work classes we defined for the XDStock application in Chapter 5, “Online transaction processing (OLTP) scenario” on page 61. The steps for accomplishing this are also described in the following section.

8.3.1 Setting up the environment

The instructions that follow assume that the XDStock application and its associated service policies and work classes have been established as described in Chapter 5, “Online transaction processing (OLTP) scenario” on page 61. It is also assumed that the long-running execution environment has already been established as described in Chapter 7, “Long-running batch workloads” on page 137.

Custom properties for LREE cluster
As described in 7.5.3, “Behavior of LREE dynamic cluster” on page 163, the server_use_wlm_to_queue_work and server_work_distribution_algorithm were automatically modified when we established the LREE in the dynamic cluster. To view these values:

1. In the administrative console, go to Servers → Application Servers.
2. Click the link for one of the two LREE application servers that were defined when we set up the dynamic cluster.
3. In the Server Infrastructure section, expand Administration.
4. Click **Customer Properties**. The customer properties set for the LREE should appear as shown in Figure 8-1.

![Customer Properties](image)

*Figure 8-1 Custom properties for LREE dynamic cluster*

These custom property settings were needed by the LREE to allow an even distribution of work across the servants within each of the application servers when running LREE workload. Since long-running workloads are scheduled by the long-running scheduler, which does not demonstrate the same request response characteristics of typical OLTP requests, LREE workloads would tend to be concentrated in the same servant region if these values were not set. A good description of these custom properties and their effect on the workload distribution can be found at the WebSphere for z/OS infocenter at:

[http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0](http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0)

The effect that these properties have when we add an OLTP application to the dynamic cluster are twofold. First, we lose the isolation of different service classes to specific servants that we saw in the strictly OLTP scenario. Secondly, the priority-based queueing done by z/OS WLM is replaced by a round-robin distribution between the servants within an application server. The enclaves for the request still have the appropriate TCLASS values associated with them, allowing their execution to be prioritized by IRD and PRSM. We found that our OLTP service policy goals were still achieved, although the loss of queueing and blending of requests for different service classes did seem to compromise the ability to favor all high-priority requests in very heavy workload situations. In spite of this, the test did show the service policy goals were enforced.

**Using edition management to re-use service policy mappings**

For this scenario we installed the same XDStock application from 5.2, “Installing the XDStock application” on page 64, into the LREE cluster we configured in Chapter 6, “Establishing the long-running environment” on page 99. We used the edition management feature of WebSphere XD, which allowed us to not have to
redefine the work classes and service policy mappings for the XDStock application. To accomplish this we performed the following steps:

1. In the administrative console, go to **Applications → Install New Application**.

2. On the Preparing for the application installation panel, identify the location of the XDStock EAR file using either option and click **Next**.

3. On the Select installation options panel enter LREE Deployment for the Application Edition field, as shown in Figure 8-2. Specifying a unique application edition name will cause the application to be installed as a distinct edition not only with regard to the EAR, but also the installation options specified in the following steps.

![Figure 8-2 Specifying an application edition during installation](image)

4. Click **Next**.
5. On the Map modules to servers page, select **LREECLUSTER**, as shown in Figure 8-3.

![Figure 8-3 Map modules to servers](image)

6. Click **Next**.

7. Accept the defaults on the Map virtual hosts to Web modules page, as shown in Figure 8-4.

![Figure 8-4 Map virtual hosts for Web modules](image)

8. Click **Next**.

9. On the Clone Existing Work Classes page, select **XDStock** as the application edition from which to clone the work classes. Work classes are used to identify various types of application requests, which are then mapped to
transaction classes. By cloning the previously defined work classes, this information, along with the mappings to transaction classes and service classes defined for the XDStock application in Chapter 5, “Online transaction processing (OLTP) scenario” on page 61, can be re-used.

10. Click **Next**.

11. Continue through the remaining application installation panels. After the application has installed successfully, from the administrative console, go to **Applications** → **Edition Control Center**.

12. Check **Base Edition**, as shown in Figure 8-6. This is the edition we installed in 5.2, “Installing the XDStock application” on page 64.
13. Click **Deactivate**. For the purpose of this scenario, since we are only using the edition control center to move the application from one cluster to another, we will not make use of the rollout feature. The rollout feature is more appropriate for transitioning from one edition of an application to another, where continuous availability of the application is desired. That is not the case here.

14. Click **Save to Master Configuration** on the Deactivate edition page.

15. After the application has deactivated successfully, from the administrative console, go to **Applications → Edition Control Center**.

16. On the Manage Editions page, check the **LREE Deployment** edition, as shown in Figure 8-7.

17. Click **Activate**.

18. On the Activate Edition page click **Save to Master Configuration**.

### 8.3.2 Executing the workload

We drove the workload for this scenario in the same way that we drove the long-running and OLTP workloads in Chapter 5, “Online transaction processing (OLTP) scenario” on page 61, and Chapter 7, “Long-running batch workloads” on page 137. The only difference is that we only stated the LREE dynamic cluster member on a single node (h6nodeb) in order to focus and simplify our observations. Workload balancing and execution across multiple nodes was addressed in the chapters mentioned previously.
We increased the workload until the CPU utilization on h6nodeb reached 100%, as shown in Figure 8-8. We then observed the average response times shown in Figure 8-9 on page 177.

Figure 8-8  Runtime Topology showing CPU utilization
Notice that in Figure 8-9, for the most part, the platinum (in pink) and gold (in green) requests continue to meet their goals (shown as horizontal lines). To a lesser extent the silver (in yellow) and bronze (in blue) requests managed to nearly achieve their goals.

![Figure 8-9 Chart showing response time of transaction classes](image)

A noteworthy exception are the spikes that occur at a few points where the platinum and silver response times shoot up but do not overshoot their goals by very much. Notice that at these same times the silver and bronze requests experienced improvements in response time. We believe that this may be an indication of WLM not queueing requests due to introducing the OLTP application server into the same dynamic cluster as the LREE application server discussed at the beginning of 8.3, “Mixed workloads in the same dynamic cluster” on
We utilized SDSF to view the WLM Service class enclaves and observed the results shown in Figure 8-10. Take note that we are running two different workloads, an OLTP application and a long-running application, and we are seeing the enclaves of the service classes defined for those two applications.

![Figure 8-10 Enclaves of the XD’s WLM service classes](image)

We also checked RMF Monitor III for our sysplex and noticed that the online transaction application service classes are favored over the long-running BATCHHI service classes. There was even further favoritism shown for the more important XDPLATSC and XDGOLDSC within the online transaction application service classes, as shown in Figure 8-11.

![Figure 8-11 RMF Monitor III sysplex system performance summary](image)
On displaying further RMF information about the XDPLATSC service class, we observed that the work is performed on two servants, H6MWA1BS, as shown in Figure 8-12, while the long-running job is also executing on the same node.

| Command ==/>_ | RMF V1RS Work Manager Delays - WTSCLPLXI | Line 1 of 2 |
| WLM Samples: 400 | Systems: 16 Date: 04/27/06 Time: 15.03.20 Range: 100 Sec |
| Class: (XDPLATSC) | Period: 1 | Avg. Resp. time: 0.024 sec for 2844 TRX. |
| Goal: 0.125 sec average | Avg. Exec. time: 0.023 sec for 6 TRX. |
| Actual: 0.024 sec average | Abnormally ended: 0 TRX. |
| Sub P | Response time breakdown (in %) | Switched-- |
| Type | Tot Act Rdy Idle--Delayed by--Time (%) | Delayed by-- |
| **CPU** | 71.2 69.7 | 0 0 1.5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |

--- Address Spaces Serving this Service Class XDPLATSC ---
| Jobname | M ASID | System | Serv-Class | Service | Proc-Usg | I/O-Usg | Veloc | Capp | Quies |
| H6MWA1BS | Y | 140 | SC50 | SYSSTC | 36 | 3.1 | 60 | 0 | 0 |
| H6MWA1BS | Y | 147 | SC50 | SYSSTC | 26 | 15 | 55 | 0 | 0 |

Figure 8-12   Address spaces serving service class XDPLATSC

### 8.4 Mixed workloads in dynamic cluster sharing common business logic

One benefit of WebSphere Extended Deployment’s Batch feature is that it allows for an invocation pattern, based on stream-based record processing and checkpoint transaction management, to be used with business logic existing in a J2EE OLTP application. When this is done the choice needs to be made as to how the existing J2EE application and the job step entity beans developed to support the batch programming model will be deployed. The choices can be summarized as follows:

- **Deploy two separate instances** of the existing J2EE application, one for the purpose of servicing OLTP requests and the other deployed with the additional long-running EJBs and other assets to support the batch workload. These can be deployed into separate clusters, as described in 8.2, “Mixed workloads on same nodes and separate dynamic clusters” on page 169. This has the benefit of allowing for separate administration and software life-cycle management for each, but also introduces the possibility of having two different versions of the re-used business logic existing within the environment.

- **Deploy both the existing J2EE application containing** the common business logic and the new assets supporting the batch programming model together. This allows for a single copy of the re-used business logic to reside and be administered in the system.
It is this second option that we tested in the scenario described in this section.

**Note:** At the time of publication there is a limitation on of packaging long-running and OLTP application code in the same EAR file. The presence of the batch deployment descriptor information and other batch assets within the EAR file causes the OLTP Web application request information to not be made available for mapping to work classes, as shown in Figure 8-16 on page 182. We circumvented this limitation by packaging the long-running and OLTP applications in separate EAR files but deploying them into the same dynamic cluster. This may have added some overhead, in that only EJB remote interfaces were available to the application for the job step EJBs.

### 8.4.1 Setting up the environment

The BatchTrade application we deployed in Chapter 7, “Long-running batch workloads” on page 137, also contained an OLTP application named TraderServices that is invoked in a traditional OLTP manner. In this scenario we add a work class to the TraderServices application and map it to a transaction class similar to what we did for the XDStock application in Chapter 5, “Online transaction processing (OLTP) scenario” on page 61. The significant difference is now we are only working with a single type of OLTP request that we intend to compare with the performance of the long-running workload. Therefore we only defined a single work class and mapped it to the existing platinum transaction class (XDPLATTC). This was done as follows:

1. In the administrative console, go to **Applications** → **Enterprise Applications**.

2. In the list of enterprise applications find the TraderServices application that was used previously. Click **TraderServices**.

3. Click the **Service Policies** tab, as shown in Figure 8-13.

![Enterprise Applications](image)

*Figure 8-13  Service Policies tab for TraderServices application*
4. On the Service Policies tab, expand **Work Classes for HTTP Requests**, as shown in Figure 8-14. Click **New**.

![Figure 8-14](image)

**Figure 8-14   New work class for TraderServices application**

5. Enter **Trader_WC** as the name of the work class, as shown in Figure 8-15.

![Figure 8-15](image)

**Figure 8-15   Define work class general properties**

6. Click **Next**.

7. Select module **TraderDBWeb.war**.
8. Under Available HTTP patterns select /TraderDBWeb/TraderSQLJDBCServlet(TraderDBWeb.war).
9. Click Add >>. The page shown in Figure 8-16 should appear.

![Figure 8-16   Define work class membership](image)

10. Click Next.

![Figure 8-17   Work class confirmation](image)

11. Click Finish.
12. Expand Trader_WC.
13. Go down to the “If no classification rules apply, then classify to this transaction class” portion of the panel, near the bottom. Select the existing transaction class XDPLATSC, as shown in Figure 8-18.

![Figure 8-18 Map transaction class to work class](image)

14. Click **Apply**.

### 8.4.2 Executing the workload

We injected workload using the WebSphere Studio Workload Simulator (WSWS) to drive the TraderServices OLTP application active in the LREECLUSTER on h6nodeb.
While WSWS was driving the workload for the online trades, we used the Batch Trade Job Console, as shown in Figure 8-19, to submit the long-running jobs.

![Batch Trade Job Console](image)

**Figure 8-19 Batch Trade Job Console**

We observed in the SDSF enclave panel shown in Figure 8-20, that the OLTP transactions are classified as XDPLATSC and the BatchTrade jobs were classified as BATCHHI.

![Enclaves for OLTP and BatchTrade](image)

**Figure 8-20 Enclaves for OLTP and BatchTrade**

### 8.5 Final thoughts and recommendations

Presented here are some final thoughts and recommendations from our experiences running the mixed OLTP and long-running workloads.
8.5.1 The best approach for deploying mixed workloads

Whether or not WebSphere OLTP and long-running workloads should be mixed, and how they should be mixed, is dependent on many factors. However, some recommendations can be made based on our mixed workload scenarios.

z/OS customers are very sensitive to the fact that batch workloads should not affect OLTP workloads. Our first scenario, mixed workloads in the same cell but on different dynamic clusters, provides that kind of resource separation and isolation. If a long-running application malfunctions and brings down the server (control region), then the OLTP workloads will continue running uninterrupted. Additionally, the default behavior of the LREE dynamic cluster is to spray the workload across multiple servants. This facilitates long-running workload throughput and scalability as well as providing for an easier maintenance model.

The third scenario, running mixed workloads in the same dynamic cluster sharing common business logic, may be more representative of the WebSphere world. Many WebSphere OLTP applications contain complex business logic that could be reused by long-running applications. There are many ways to do this. We chose a scenario where the long-running application made EJB calls to the business logic. This exploited the local EJB interface optimizations for performance and provided the necessary transaction support.

One drawback to the third scenario is that only one servant region is available to execute the long-running workloads. This could negatively affect throughput rates. The long-running workload cannot scale vertically. The rest of the servants are used for the OLTP workloads, depending upon the number of service classes (that is, one servant per service class). Another drawback is that both the OLTP and long-running workloads cannot be packaged as a single ear file. We noticed that when we did this, we were unable to associate service policies with the OLTP workload.

8.5.2 Service policy determination

WebSphere Extended Deployment for z/OS in conjunction with WLM met the service policy goals for OLTP workloads in all our mixed workload scenarios. Ensuring that WLM can manage the service policies you plan to implement requires the application of appropriate capacity-planning techniques.

One best practice is to do a performance test of your J2EE applications and to determine their performance profiles on production hardware before setting more granular WAS XD and WLM service policies. Goals such as average response times for the service policies are only useful when achievable running under moderately loaded conditions. WAS XD and z/OS WLM cannot magically transform a poorly performing application or configuration into one that...
accomplishes arbitrary service policy goals. The goals defined in the service
policies should allow a well-tuned application to continue to achieve feasible
goals under heavy load conditions, especially for requests classified as having
high importance. Other requests with lower importance will be managed to
maximize the utilization of scarce remaining resources to achieve their goals as
much as possible in order of importance.

Note: The z/OS WLM service policy is used to manage the attainment of
performance goals, while the WAS XD service policy is used for classification
(WAS XD performs classification at a more effective location in the ODR rather
than the Application Server Control Region as was the case previously). Both
service policies can be used for reporting purposes within their respective
tools (that is, WAS XD charts and health monitoring for WAS XD and RMF,
SDSF for z/OS WLM).

There is a simple set of steps to follow to evaluate the readiness an application
for deployment:

1. Test the application on a single servant to determine its response time under
a load as follows:
   a. Tune the single servant operating environment (HW, OS, LPAR, and
      storage).
   b. Tune access (drivers, connection pooling) to external resources (RDB,
      MQ, CICS, IMS™, LDAP, and so on).
   c. Tune WebSphere (Java heap, caches, class loaders, and so on).
   This gives a first-pass set of average response-time numbers for the
   application’s transactions.

2. Test the application under a load in a static cluster to determine if its response
time to load curve is linear as new servants are engaged.
   - This includes determining the capacity of the back-end resources.
   - This validates the average response-time numbers for the application.

3. Set the service policies for the application based on the previous set of
   information.

4. Test the application in a dynamic cluster with other J2EE applications you
   plan to deploy to ensure that the entire environment can still support the
   established service policies.
Troubleshooting and performance tips

In this appendix we provide approaches to tuning and troubleshooting WebSphere Extended Deployment for z/OS as well as document where to obtain further information about tuning and troubleshooting WebSphere Application for z/OS.
WebSphere Infocenter

The Infocenter for WebSphere Application Server for z/OS has a wealth of information about tuning and troubleshooting. You can find it on the Web at:


We document some of the important information about tuning and troubleshooting in this appendix, which we referred to during our project.

Java Virtual Machine Storage tuning tips for z/OS

Specifying a sufficient JVM heap size is important to Java performance.

The JVM has thresholds it uses to manage the JVM’s storage. When the thresholds are reached, the garbage collector (GC) gets invoked to free up unused storage. GC can cause significant degradation of Java performance.

Use the administrative console to specify the initial heap size and the maximum heap size for the JVM.

To view this administrative console page, click Servers → Application Servers → server_name → Process Definition → Java Virtual Machine. Access the Configuration tab to change these settings.

- In the majority of cases you should set the maximum JVM heap size to a value higher than the initial JVM heap size. This allows for the JVM to operate efficiently during normal, steady state periods within the confines of the initial heap, but also to operate effectively during periods of high transaction volume by expanding the heap up to the maximum JVM heap size. In some rare cases where absolute optimal performance is required you might want to specify the same value for both the initial and maximum heap sizes. This will eliminate some overhead that occurs when the JVM needs to expand or contract the size of the JVM heap. Make sure that the region is large enough to hold the specified JVM heap.

- Beware of making the initial heap size too large. While it initially improves performance by delaying garbage collection, it ultimately affects response time when garbage collection eventually kicks in (because it runs for a longer time).

- Paging activity on your system must also be considered when you set your JVM heap size. If your system is already paging heavily, increasing the JVM heap size might make performance worse rather than better.
For WebSphere Application Server for z/OS the default JVM heap size settings are 128 m/256 m (initial/maximum) for the controller and 256 m/512 m for the servant. For the majority of applications these defaults should work quite well. You can monitor the frequency of garbage collection by enabling verbose GC (see below). If garbage collection is occurring too frequently you should increase the size of the JVM heap.

To determine whether you are being affected by garbage collection, you can enable verbose garbage collection on the JVM Configuration tab. The default is not enabled. This will write a report to the output stream each time the garbage collector runs. This report should give you an idea of what is going on with Java GC. Figure A-1 shows an example of a verbose GC report.

```
<af type="tenured" id="7" timestamp="Mon Nov 14 22:58:18 2005"
    intervalms="3724.328">
    <minimum requested_bytes="72" />
    <time exclusiveaccessms="0.034" />
    <tenured freebytes="4026368" totalbytes="134217728" percent="2"
        >
        <soa freebytes="0" totalbytes="130191360" percent="0" />
        <loa freebytes="4026368" totalbytes="4026368" percent="100" />
    </tenured>
    <time exclusiveaccessms="0.034" />
    <tenured freebytes="87994768" totalbytes="134217728" percent="65"
        >
        <soa freebytes="85310864" totalbytes="131533824" percent="64" />
        <loa freebytes="2683904" totalbytes="2683904" percent="100" />
    </tenured>
</af>
```

Figure A-1  VerboseGC report
Key things to look for in a verbose GC report are:

- Time spent in garbage collection

  Ideally, you want to be spending less than 5% of the time in GC. To determine the percentage of time spent in GC, divide the time it took to complete the collection by the time since the last AF and multiply the result by 100. For example:

  \[
  \frac{83.29}{3724.32} \times 100 = 2.236\%
  \]

  If you are spending more than 5% of your time in GC and if GC is occurring frequently, you may need to increase your Java heap size.

- Growth in the allocated heap

  To determine this, look at the %free. You want to make sure that the number is not continuing to decline. If the %free continues to decline, you are experiencing a gradual growth in allocated heap from GC to GC, which could indicate that your application has a memory leak.

You can also use the MVS console command `modify display, jvmheap` to display JVM heap information. Refer to “Modify command” on page 190 for details. In addition, you can check the server activity and interval SMF records. The JVM heap size is also made available to PMI and can be monitored using the Tivoli® Performance Viewer.

**Modify command**

During our project, we used the tracing facility for WAS extensively to see if the servers are performing well as well as to diagnose application problems. In this section we document the various tracing options that can be turned on.

Use the `modify` command from the MVS console to dynamically modify WebSphere Application Server for z/OS operations.

You can use the `modify` command to display the status of various server components and activities, including:

- Active controllers
- Trace settings
- Servants
- Sessions
- JVM heap
- Java trace

The modify command syntax is:

`f <server>, options`
The first argument is required. The argument is the name of the server for which the `modify` command will be directed.

### Parameters

The options for the `modify` command include:

- **CANCEL**

  This is used to cancel a server. *Server* refers to the server short name.

  You can specify the following options:

  - **ARMRESTART**
    
    Specify whether you are using ARM and want ARM to restart the server after it terminates. If you do not specify the ARMRESTART option on the CANCEL parameter, ARM will not restart the server.
  
  - **HELP**

    This gets help for the CANCEL syntax.

    **Note:** You cannot use the CANCEL parameter to cancel a cluster from the MVS console. Instead, you must cancel each of the servers that make up the cluster.

- **TRACEALL=n**

  Use TRACEALL to establish a general trace level for the server.

  Valid trace levels are 0 (none), 1 (exception), 2 (basic), and 3 (detailed tracing). Under normal conditions and in production, use 1 (exception).

  **Note:** Be careful when using a level of 3 (detailed for all components) because it can potentially yield more data than can be handled reasonably.

- **TRACEBASIC=n**

  Specify the WebSphere Application Server for z/OS components for which you want to switch on a basic level of tracing.

  This command has the ability to override a different tracing level established by TRACEALL for those components.

  **Note:** Do not change this variable unless directed by IBM service personnel.
You can specify one or more of the following options for either TRACEBASIC or TRACEDETAIL:

- 0 - RAS
- 1 - Common Utilities
- 3 - COMM
- 4 - ORB
- 6 - OTS
- 7 - Shasta
- 9 - OS/390® Wrappers
- A - Daemon
- E - Security
- F - Externalization
- J - JRAS (internal tracing-via direction from IBM support)
- L - J2EE

- TRACEDETAIL=n

Specifies the WebSphere Application Server for z/OS components for which you want to switch on a detailed level of tracing.

This command activates the most detailed tracing for the specified WebSphere Application Server for z/OS components and overrides different settings in TRACEALL. The selected components \((n,...)\) are identified by their component-ID (valid values are the same as for TRACEBASIC above. Subcomponents, specified by numbers, receive detailed traces. Other parts of WebSphere Application Server for z/OS receive tracing as specified on the TRACEALL variable.

**Note:** Do not change this variable unless directed by IBM service personnel.

- TRACESPECIFIC=xxyyyyzzz

Specifies tracing overrides for specific WebSphere Application Server for z/OS trace points.

Trace points are specified by 8-digit, hexadecimal numbers. To specify more than one trace point, use parentheses and separate the numbers with commas. You can also specify an environment variable name by enclosing the name in single quotation marks. The value of the environment variable will be handled as though you had specified that value on TRACESPECIFIC.

**Note:** Do not use TRACESPECIFIC unless directed by IBM service personnel.
- **TRACE_EXCLUDE_SPECIFIC=xxyyyzzz**
  
  This specifies trace points to exclude.

  Trace points to exclude are specified by 8-digit, hexadecimal numbers. To specify more than one trace point, use parentheses and separate the numbers with commas. You can also specify an environment variable name by enclosing the name in single quotation marks. The value of the environment variable will be handled as though you had specified that value on TRACE_EXCLUDE_SPECIFIC. You can use TRACE_EXCLUDE_SPECIFIC as a mask to turn off otherwise-on traces. For example, use the TRACESPECIFIC command to turn on tracing for a whole part and then use TRACE_EXCLUDE_SPECIFIC to turn off one trace within that part.

  **Note:** Do not use TRACE_EXCLUDE_SPECIFIC unless directed by IBM service personnel.

- **TRACEINIT**
  
  Reset to the initial trace settings.

- **TRACENONE**
  
  Turns off all trace settings.

- **TRACETOSYSPRINT={YES|NO}**
  
  Select whether to send the trace to sysprint.

  YES specifies to send the trace to sysprint and NO stops sending the trace to sysprint.

- **TRACETOTRCFILE={YES|NO}**
  
  Select whether to direct the trace to the TRCFILE DD card.

  YES specifies to send the trace to the TRCFILE DD card and NO stops sending the trace to the TRCFILE DD card.

- **TRACEJAVA**
  
  Modify the Java trace string.

  The Java trace specification is used to control Java tracing and conforms to the Java trace specification rules. *=all=enabled means to enable all types of tracing for all registered trace components.

- **HELP**
  
  Display a list of all the keywords you may use with the `modify` command.
You can also use the HELP parameter after the CANCEL and DISPLAY parameters to display lists of all the keywords you can use with either of these parameters.

- **PAUSELISTENERS**
  Prevents work from being accepted into the server. Use this command if you want to shut down the communication listeners and purge any pending work in the work registry.

- **RESUMELISTENERS**
  Use this command to restart the communication listeners after issuing a PAUSELISTENERS command. Allows work to be accepted into the server.

- **DISPLAY | DISPLAY,**
  Displays the server's name, the system name where it is running, and the current code level.

You can specify the following options:

- **SERVERS:** This command is directed to a server and displays the name, system name, and code level for each active server in the sysplex that is in the same cell.

- **SERVANTS:** displays a list of ASIDs of servants attached to the server against which you issued the display command.

- **TRACE:** displays trace information for a server controller. You can further modify this command with one of the following options:
  - **SRS:** displays trace information for all servants, one at a time.
  - **ALL:** displays trace information for the controller and all servants one at a time.
  - **JAVA:** displays the Java trace string settings for a server controller. You can further modify this command with one of the following options:
    - **SRS:** displays Java trace information for all servants, one at a time.
    - **ALL:** displays Java trace information for the controller and all servants one at a time.
    - **HELP:** displays a list of all the keywords you may use with the modify display trace Java command.
  - **HELP:** displays a list of all the keywords you may use with the modify display trace command.

- **JVMHEAP:** displays the JVM heap information for a server controller. You can further modify this command with one of the following options:
  - **SRS:** displays the JVM heap information for all servants, one at a time.
• ALL: displays the JVM heap information for the controller and all servants, one at a time.

• HELP: displays a list of all of the keywords that you may use with the modify display Javaheap command.

– SESSIONS: displays session information for the server. You can further modify this command with one of the following options:
  • LISTENERS: displays the listening port numbers for each protocol. This is actually the default, so the server, sessions and the server, sessions, listeners commands would have the same outcome.

• SERVER: displays the number of sessions in use for each protocol on the server. You can further modify this command with one of the following options:
  – TCPIIOP: displays the number of TCP/IP IIOP sessions active on the server. You can further modify this command with one of the following options:
    • LIST: lists the server session information for the TCP/IP IIOP protocol. The LIST parameter is also available with the other session protocol display commands.
    • HELP: displays a list of all the keywords you may use with the modify display session server tcpipiiop command. The HELP parameter is also available with the other session protocol display commands.
  – LOCALIIOP: displays the number of LOCALIIOP sessions active on the server.
  – SSLIIOP: displays the number of SSLIIOP sessions active on the server.
  – HTTP: displays the number of HTTP sessions active on the server.
  – HTTPS: displays the number of HTTPS sessions active on the server.
  – HELP: displays a list of all of the keywords that you may use with the modify display session server command.

• HELP: displays help for the modify display sessions command.

– HELP: displays a list of all of the keywords that you may use with the modify command.
WAS XD and UNIX system services

During our project we encountered occasions when the /tmp directory was full and some of the servers failed to start. The /tmp directory is full because WebSphere Application Server would generate JAVADUMPs as well as HEAPDUMPs if there are any abends. This would fill up the /tmp directory, and because the visualization directory for WAS XD sits inside /tmp, it would cause WAS to fail to start up.

To check whether the /tmp directory is full, check the following areas:

- Check the server’s job log to see if there are any BPX messages.
- Visually check whether the /tmp directory and the root HFS for the server in question are not full but using the df command while in omvs. Refer to z/OS UNIX System Services Command Reference for more information about the df command.
Additional material

This book refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG247267

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 book form number, SG247267.

Using the Web material

The additional Web material that accompanies this book includes the following files:

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG247267.zip</td>
<td>Zipped code samples, readme file</td>
</tr>
</tbody>
</table>
How to use the Web material

Create a subdirectory (folder) on your workstation and unzip the contents of the Web material zip file into this folder. Open the readme file. This file contains the description of the accompanying files and the instructions for their use.
Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 200. Note that some of the documents referenced here may be available in softcopy only.

- *Using WebSphere Extended Deployment V6.0 To Build an On Demand Production Environment*, SG24-7153
- *Architecting High Availability Using WebSphere V6 on z/OS*, SG24-6850

Online resources

These Web sites and URLs are also relevant as further information sources:

- *WebSphere for z/OS Extended Deployment XD - Building the Basic Infrastructure*, WP100735
  http://w3-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP100735
- WebSphere Extended Deployment on-line Infocenter
  http://publib.boulder.ibm.com/infocenter/wxdinfo/v6r0/index.jsp
- The WebSphere Application Server Version 6.0 Information Center
  http://publib.boulder.ibm.com/infocenter/ws60help/
- The IBM WebSphere Studio Workload Simulator for z/OS and OS/390 home page
  http://www.ibm.com/software/awdtools/studioworkloadsimulator/
How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services
Index

Numerics
6.2 Installing the application 64

A
Admin 50, 97, 102
application placement controller (APC) 12
application server 3, 15, 82, 101, 124
various types 8

B
Basic Infrastructure 47, 50
Batch 30
batch application 23
controller bean 33
distinct tasks 28
enterprise bean deployment descriptor 34
Batch Controller Bean 33
Batch Data Stream 30
multiple records 34
batch job 26–27, 114, 132
current state 30
execution flow 28
return codes 34
z/OS resources 35
Batch Model 37
batch step 28
global transactions 33
output stream 28
return code 33

C
cluster member 7, 16, 51, 85, 110
dynamic weights 97
dynamic workload manager weights 16
Concepts 11
Creating 129

D
data source 7, 31, 100, 115
jdbc/lree JNDI name 135
JNDI name 135

WebSphere Extended Deployment decrease
contention 7
dedicated dynamic cluster
mixed transactional and long-running workloads 2
DMGR 44
dynamic Cluster
current list 104
first LRS application server 112
LRS servers 110
dynamic cluster 24, 46, 51, 56, 64, 85, 100, 110,
168, 185
application servers 86
node group 56
various cluster members 85
XDStock application 64

E
EEXSTEP3 job 45
Establishing 99
execution environment 11, 23, 46, 100
other instances 24
particular instance 24

G
garbage collector (GC) 188
global transaction 32
batch step EJB methods 32

H
he 22

I
IBM HTTP server (IHS) 42
ITSO configuration 101
ITSO environment 99

J
J2EE application 22, 185
dynamic cluster 186
J2EE skill 35
Java Virtual Machine (JVM) 35, 115, 188
JDBC provider 115
  DB2 environment variables 115
  WebSphere environment variables 118
JNDI name 122
JVM heap 188
  information 190
  size 188

L
long-running application 13, 26, 185
  enhanced environment 23
  programming model 22
  service goal 13
long-running batch
  application 27
  application deployment descriptor 33
  Java application 36
  job 29
Long-Running Environment 6, 22, 100
  Component 28, 115
  fundamental building blocks 101
  good installation verification test 136
  JDBC provider 115
  overall architecture 100
Long-Running Execution Environment (LREE) 5, 11, 32, 46, 100, 128
long-running job 26, 28
Long-Running Scheduler (LRS) 26, 46, 100, 124
long-running work 23
long-running workload 2, 23, 26, 37, 99, 129, 168–169
  general types 4
LREE application
  server 129, 131
  server instance 32
LRS application
  component 126
  server 107, 111
LRS application server
  configuration panel 109
  instance 109
  short name 110

M
mixed workload 2, 167

N
Node Group 56, 88, 100
  average response time 88
  long name 102
  managed nodes 104
  member 56, 103
  members pane 103

O
ODR 4, 14, 50–51, 61, 169
  Modifying default values 54
OLTP scenario 62
On Demand Router 4, 11–12, 46, 49, 86, 97
Online Transaction Processing (OLTP) 3, 168
Open System Adapter (OSA) 47
Our back-end architecture with two on demand routers and three dynamic clusters 46

R
Redbooks Web site 200
  Contact us xi
Resource Access Control Facility (RACF) 47
Resource Recovery System (RRS) 47

S
Service Class 66, 168, 185
Service Policy 4, 6, 12, 66, 172, 185
  average response time 94
  goal portion 12
  performance goals 13
Setting up the On Demand Router 50
soa freebytes 189
SRS Display
  Java trace information 194
  trace information 194
Sysplex Distributor 47
  cluster 15

T
TCLASS 19, 71, 169
traditional z/OS batch
  processing 5
Transaction class 14, 66, 70, 177–178
  creation 74
  definition 14
  general properties panel 73
  general property 73
information 75
items panel 73
name 14, 71
response time 177
XDPLATSC 179

V
Virtual Machine (VM) 42

W
WebSphere Application
  Server 1, 3, 36, 39, 43, 97, 110, 188
  Server environment 50
  Server Extended Deployment 2
  Server Network Deployment 68
  Server process 27
  Server V6.0.2.6 47
WebSphere Application Server 189
  Extended Deployment 2
  process 35
  V6.0.2.6 43
WebSphere Extended Deployment ix, 1, 7, 12–14,
22, 40–41, 51, 87, 97, 99–100, 138, 187, 208
  backend 18
  differentiating feature 4
  dynamic cluster 24
  environment 1, 14
  feature 46
  Installation 47
  license 8
  literature 22
  long-running batch application 27
  long-running batch applications 27
  long-running environment 22
  long-running workloads 22
  node 37
  ODR component 8
  on-line Infocenter 27, 31
  service 36
  terminology 2
  V6.0 6, 107
  V6.0.1 6, 43
  V6.0.2 22
  V6.0.2 LREE 25
WebSphere Workload Simulator (WSWS) 183
WLM Policy 37
work class 14, 76, 174, 180
  negative effects 76

Work Load Management
  Service Classes 66
Work Load Management (WLM) 66

X
XML Job Control Language (XJCL) 26

Z
z/OS benefit 36
  WebSphere Extended Deployment 36
z/OS customer 15, 185
z/OS service 99
z/OS static (ZOS) 119
z/OS WLM 4, 15, 66
  Classification Rules 66
  information workload classification 21
  Integration 18
  Service Class 70
  TCLASS 85
z/OS WLM Classification Rules 66
Mixed Workloads in WebSphere XD V6.0 on z/OS
Mixed Workloads in WebSphere XD V6.0 on z/OS

This IBM Redbook extends the material provided in the Using WebSphere Extended Deployment V6.0 To Build an On Demand Production Environment, SG24-7153, by providing a z/OS-centric description of the On Demand Router (ODR) and the long-running application support provided with WebSphere Extended Deployment (XD) 6.0.

This book demonstrates the additional value the ODR provides as it is integrated with the existing z/OS Workload Manager. It also describes the long-running application support now available on z/OS with XD 6.0 along with its relationship to the traditional z/OS batch facilities.

This book describes an XD 6.0 installation into our existing ITSO High Availability WebSphere Network Deployment environment as well as step-by-step instructions that take you through the configuration of the ODRs and dynamic clusters. It presents the setting up of a long-running application environment in detail. It uses a sample scenario to demonstrate the integration of an ODR Service Policy with a corresponding WLM Service Policy and the resulting finer grained WLM workload classification that the ODR now makes possible. Additional scenarios take you through the deployment and execution of a long-running workload.

For more information: ibm.com/redbooks