

IBM Business Process Manager V7.5 Performance Tuning and Best Practices

Learn valuable tips for tuning

Get the latest best practices

See example settings



IBM Business Process Management
Performance Team



International Technical Support Organization

**IBM Business Process Manager V7.5 Performance
Tuning and Best Practices**

April 2012

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

First Edition (April 2012)

This edition applies to IBM Business Process Manager V7.5, IBM Integration Designer V7.5, and IBM Business Monitor V7.5.

This document was created or updated on April 11, 2012.

© Copyright International Business Machines Corporation 2012. All rights reserved.

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

Contents

Notices	vii
Trademarks	viii
Preface	ix
The team who wrote this paper	ix
Now you can become a published author, too!	x
Comments welcome	x
Stay connected to IBM Redbooks	x
Chapter 1. Overview	1
1.1 Products covered in this publication	2
1.2 Publication structure	3
Chapter 2. Architecture best practices	5
2.1 Top tuning and deployment guidelines	6
2.2 Modeling	7
2.2.1 Common best practices	8
2.2.2 Process Designer architecture best practices	8
2.2.3 Integration Designer best practices	10
2.3 Topology	13
2.3.1 Deploying appropriate hardware	13
2.3.2 Deploying local modules in the same server	13
2.3.3 Best practices for clustering	14
2.3.4 Evaluating service providers and external interfaces	15
2.4 Business Space	16
2.4.1 Optimizing topology	16
2.4.2 Using a network with low latency and high bandwidth	17
2.4.3 Using a high-performing browser	17
2.4.4 Enabling browser caching	18
2.4.5 Locating servers physically near clients	18
2.4.6 Using modern desktop hardware	18
2.5 Large objects	18
2.5.1 Factors affecting large object size processing	19
2.5.2 Large object design patterns	19
2.6 64-bit considerations	20
2.7 Business Monitor	21
2.7.1 Event processing	21
2.7.2 Dashboard	22
2.7.3 Database server	22
Chapter 3. Development best practices	23
3.1 Process Designer development best practices	24
3.1.1 Clearing variables in exposed human services not intended to end	24
3.1.2 Not using multi-instance loops in the system lane or for batch activities	24
3.1.3 Using conditional joins only when necessary	25
3.1.4 Guidelines for error handling	25
3.1.5 Using sequential system lane activities efficiently	26
3.1.6 Ensuring the Process Center is tuned	27
3.1.7 Using a fast connection between Process Designer and Process Center	27

3.1.8 Preventing Web Services Description Language validation from causing slow web service integrations.	27
3.2 Integration Designer best practices.	28
3.2.1 Business object parsing mode considerations	28
3.2.2 Service Component Architecture considerations	34
3.2.3 Business Process Execution Language business process considerations.	35
3.2.4 Human task considerations.	36
3.2.5 Business process and human tasks client considerations	36
3.2.6 Transactional considerations	37
3.2.7 Invocation style considerations	39
3.2.8 Large object considerations	41
3.2.9 Mediation flow considerations.	42
3.3 WebSphere InterChange Server migration considerations.	43
3.4 Authoring environment considerations	44
3.5 Business Space developer considerations	45
Chapter 4. Performance tuning and configuration	47
4.1 Performance tuning methodology	48
4.1.1 Picking a set of reasonable initial parameter settings.	48
4.1.2 Monitoring the system.	48
4.1.3 Using monitoring data to guide further tuning changes.	49
4.2 Tuning checklist	49
4.3 Common tuning parameters	51
4.3.1 Tracing and logging flags	51
4.3.2 Java tuning parameters	52
4.3.3 Message-driven bean ActivationSpec	52
4.3.4 Thread pool sizes	53
4.3.5 Java Message Service connection pool sizes.	53
4.3.6 Java Database Connectivity data source parameters.	53
4.3.7 Messaging engine properties	54
4.3.8 Running production servers in production mode.	55
4.4 Advanced tuning	55
4.4.1 Tracing and monitoring considerations.	55
4.4.2 Tuning for large objects	56
4.4.3 Tuning for maximum concurrency.	57
4.4.4 Messaging tuning	59
4.4.5 Clustered topology tuning	62
4.4.6 Web services tuning	64
4.4.7 Power management tuning.	64
4.4.8 Setting AIX threading parameters.	64
4.5 Business Process Choreographer tuning for Business Process Execution Language business processes	65
4.5.1 Tuning Work Manager-based navigation for business processes	65
4.5.2 Tuning the business process container for Java Message Service navigation.	66
4.5.3 Tuning task list and process list queries	66
4.5.4 Tuning Business Process Choreographer API calls	66
4.5.5 Tuning intermediate components for concurrency	67
4.6 Business Processing Modeling Notation business process tuning	67
4.6.1 Database tuning	67
4.6.2 Business Process Definition Queue Size and Worker Thread Pool	68
4.7 WebSphere ESB tuning	68
4.7.1 Tuning the database if using persistent messaging	68
4.7.2 Disabling event distribution for Common Event Infrastructure	69

4.7.3	Configuring WebSphere Service Registry and Repository cache timeout	69
4.8	Business Monitor tuning	69
4.8.1	Configuring Java heap sizes	69
4.8.2	Configuring Common Event Infrastructure	69
4.8.3	Configuring message consumption batch size	70
4.8.4	Enabling key performance indicator caching	70
4.8.5	Using table-based event delivery	70
4.8.6	Enabling the Data Movement Service	71
4.9	Business Space tuning	71
4.9.1	Ensuring browser cache is enabled in Internet Explorer 8	71
4.9.2	Ensuring browser cache is enabled in Firefox 3.6	74
4.9.3	Optimizing complex task message performance	76
4.9.4	Tuning the HTTP server	76
4.9.5	Optimizing performance when not using federation mode	77
4.10	General database tuning	78
4.10.1	Providing adequate statistics for optimization	78
4.10.2	Placing database log files on a fast disk subsystem	78
4.10.3	Placing logs on a separate device from the table space containers	79
4.10.4	Providing sufficient physical memory	79
4.10.5	Avoiding double buffering	79
4.10.6	Refining table indexes as required	79
4.10.7	Archiving completed process instances	80
4.11	DB2-specific database tuning	80
4.11.1	Updating database statistics	80
4.11.2	Setting buffer pool sizes correctly	81
4.11.3	Maintaining correct table indexing	82
4.11.4	Sizing log files appropriately	82
4.11.5	Setting inline length to improve throughput for high volume systems	83
4.11.6	Using System Managed Storage for table spaces containing large objects	83
4.11.7	Ensuring sufficient locking resources are available	84
4.11.8	Setting boundaries on the size of the catalog cache for clustered applications	84
4.11.9	Sizing the database heap appropriately before DB2 9.5	85
4.11.10	Sizing the log buffer appropriately before DB2 9.7	85
4.11.11	Considering disabling current commit in DB2 V9.7 and later	85
4.11.12	Suggestions for Business Process Execution Language business processes in Business Process Manager V7.5	86
4.11.13	Suggestions for Business Monitor V7.5	86
4.12	Oracle-specific database tuning	87
4.12.1	Updating database statistics	88
4.12.2	Correctly setting buffer cache sizes	88
4.12.3	Maintaining correct table indexing	88
4.12.4	Sizing log files appropriately	88
4.12.5	Suggestions for Business Process Execution Language business processes in Business Process Manager V7.5	88
4.13	Advanced Java heap tuning	89
4.13.1	Monitoring garbage collection	90
4.13.2	Setting the heap size for most configurations	91
4.13.3	Setting the heap size when running multiple JVMs on one system	92
4.13.4	Reducing or increasing heap size if OutOfMemory errors occur	93
4.14	Tuning for WebSphere InterChange Server migrated workloads	94
	Chapter 5. Initial configuration settings	95
5.1	Business Process Manager server settings	96

5.1.1	Three-tiered configuration with Web services and remote DB2 system	96
5.1.2	Three-tiered configuration using Human Services with Business Processing Modeling Notation processes	99
5.1.3	Two-tiered configuration using file store for Java Message Service	100
5.2	WebSphere ESB settings	102
5.2.1	WebSphere ESB common settings	102
5.2.2	WebSphere ESB settings for Web services	102
5.2.3	WebSphere ESB settings for Java Message Service	102
5.2.4	DB2 settings for Java Message Service persistent	103
	Related publications	105
	IBM Redbooks	105
	Online resources	105
	How to get Redbooks	106
	Help from IBM	106

Notices

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

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

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

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

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

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

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

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

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

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


COPYRIGHT LICENSE:

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

Trademarks

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

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

AIX®	IBM®	Redbooks (logo)  ®
alphaWorks®	IMS™	Tivoli®
CICS®	MQSeries®	WebSphere®
Cognos®	POWER6®	z/OS®
DB2®	Redbooks®	
developerWorks®	Redpaper™	

The following terms are trademarks of other companies:

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

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

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

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

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

Preface

This IBM® Redpaper™ publication provides performance tuning tips and best practices for IBM Business Process Manager (BPM) V7.5 (all editions) and IBM Business Monitor V7.5. These products represent an integrated development and runtime environment based on a key set of service-oriented architecture (SOA) and business process management technologies. Such technologies include Service Component Architecture (SCA), Service Data Object (SDO), Business Process Execution Language for Web services (BPEL), and Business Processing Modeling Notation (BPMN).

Both BPM and Business Monitor build on the core capabilities of the IBM WebSphere® Application Server infrastructure. As a result, BPM solutions benefit from tuning, configuration, and best practices information for WebSphere Application Server and the corresponding platform Java Virtual Machines (JVMs).

This paper targets a wide variety of groups, both within IBM (development, services, technical sales, and others) and customers. For customers who are either considering or are in the early stages of implementing a solution incorporating BPM and Business Monitor, this document proves a useful reference. The paper is useful both in terms of best practices during application development and deployment and as a reference for setup, tuning, and configuration information.

This paper introduces many of the issues influencing the performance of each product and can serve as a guide for making rational first choices in terms of configuration and performance settings. Similarly, customers who have already implemented a solution using these products might use the information presented here to gain insight into how their overall integrated solution performance might be improved.

The team who wrote this paper

This paper is authored by the BPM performance team, with members in Austin, Texas; Böblingen, Germany; and Hursley, England.

- ▶ Mike Collins
- ▶ Weiming Gu
- ▶ Paul Harris
- ▶ Ben Hoflich
- ▶ Kean Kuiper
- ▶ Sam Massey
- ▶ Rachel Norris
- ▶ David Ogren
- ▶ David Parish
- ▶ Chris Richardson
- ▶ Randall Theobald

Now you can become a published author, too!

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- ▶ Send your comments in an email to:

redbooks@us.ibm.com

- ▶ Mail your comments to:

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

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/pages/IBM-Redbooks/178023492563?ref=ts>

- ▶ Follow us on twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

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

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

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

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

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



Overview

In this chapter, we introduce the products covered in this paper and the overall document structure.

1.1 Products covered in this publication

This paper describes the following products:

► **BPM V7.5. (all editions)**

BPM combines simplicity, ease-of-use, and task management capabilities that support enterprise integration and transaction process management requirements as part of an overall SOA.

BPM adds value in the following ways:

- Optimizes business processes by providing visibility to all process participants, fostering greater collaboration and enabling continuous process improvement across distributed platforms and IBM z/OS®.
- Increases efficiency with a federated view for performing tasks, managing work items, tracking performance, and responding to events, all in real time.
- Empowers users with real-time analytics to optimize business processes.
- Enhances time-to-value through business user-focused design capabilities, including process coaching to guide users easily through the steps of a process.
- Confidently manages change with a unified model-driven environment that makes the same process version visible to everybody.
- Combines simplicity, ease-of-use, and task management capabilities for IBM WebSphere Lombardi Edition with key capabilities from IBM WebSphere Process Server (Advanced Edition only). These capabilities support enterprise integration and transaction process management requirements as part of an overall SOA.
- Offers compatibility with earlier versions for the latest versions of WebSphere Process Server (Advanced Edition only) and WebSphere Lombardi Edition.

► **Business Monitor V7.5**

Business Monitor provides comprehensive Business Activity Monitoring (BAM). This feature enables user visibility into real-time, end-to-end business operations, transactions, and processes to help optimize processes and increase efficiency.

Business Monitor adds value in the following ways:

- Provides a high-performance BAM solution for processes and applications running in disparate environments which might or might not be implemented using any BPM technology.
- Offers built-in tools and runtime support for integrated BAM for BPM.
- Integrates IBM Cognos® Business Intelligence Server V10.1.1 for advanced analysis and reporting on historical data
- Automatically generates dashboards for your Business Process Modeling Notation V2.0 (BPMN2) processes. This feature allows real-time visibility to process instances, key performance indicators (KPIs), Cognos reports, and annotated BPMN2 process diagrams
- Includes fine-grained security to enable or prevent anyone from seeing a wide range of information depth or detail
- Enables enhanced business user customization of data filtering and dashboard controls and reports.

- Enables views of KPIs, metrics, and alerts through web interfaces, iPad, mobile devices, and corporate portals.
- Is available for distributed platform and z/OS.

1.2 Publication structure

The following list summarizes each chapter of this document:

- ▶ Chapter 1, “Overview” on page 1
The current chapter presents the scope of the content of this paper.
- ▶ Chapter 2, “Architecture best practices” on page 5
This chapter provides suggestions about architecture and topology decisions that produce well-performing and scalable solutions.
- ▶ Chapter 3, “Development best practices” on page 23
This chapter presents guidelines for solution developers that lead to high-performing systems.
- ▶ Chapter 4, “Performance tuning and configuration” on page 47
This chapter explains the configuration parameters and settings for the major software components that comprise a business process management solution.
- ▶ Chapter 5, “Initial configuration settings” on page 95
This chapter provides details about the software configurations used for representative workloads used by the IBM performance team using BPM and Business Monitor.
- ▶ “Related publications” on page 105
This chapter provides links to best practices, performance information, and product information for both the products described in this publication and related products such as WebSphere Application Server, IBM DB2®, and other applications.



Architecture best practices

This chapter provides guidance on how to design a high-performing and scalable Business Process Manager (BPM) solution. The purpose of this chapter is to highlight the best practices associated specifically with the technologies and features delivered in the BPM and Business Monitor products covered in this paper. However, these products are based on existing technologies (such as WebSphere Application Server and DB2). Each of these technologies has its own associated best practices.

This paper does not enumerate these best practices outside BPM. See “Related publications” on page 105 for a set of references information about and links to these other technologies.

2.1 Top tuning and deployment guidelines

This chapter details architectural best practices for BPM V7.5 solutions. Development best practices and performance tuning and configuration are covered in subsequent chapters.

If you read nothing else in this document, read and adhere to the following key tuning and deployment guidelines. They are relevant in virtually all performance-sensitive customer engagements:

- ▶ Use a high-performance disk subsystem. In virtually any realistic topology, you need a server-class disk subsystem (for example, a RAID adapter with multiple physical disks) on the tiers that host the message and data stores to achieve acceptable performance. We cannot overstate this point. In many cases, several factors improve the overall performance of a solution by using appropriate disk subsystems.
- ▶ Use a 64-bit Java virtual machine (JVM), and set an appropriate Java heap size to deliver optimal throughput and response time. Memory usage data obtained through JVM verbose garbage collection command (**verbosegc**) helps determine the optimal settings. Further information is available in 4.3.2, “Java tuning parameters” on page 52.
- ▶ Use a production quality database such as DB2, a high-performing, industrial strength database designed to handle high levels of throughput and concurrency. DB2 scales well and delivers excellent response time.
- ▶ Tune your database for optimal performance. Correct tuning and deployment choices for databases can greatly increase overall system throughput. For details, see 4.10, “General database tuning” on page 78.
- ▶ Disable tracing. Tracing is important when debugging, but the resources required to do so severely affects performance. More information is available in 4.4.1, “Tracing and monitoring considerations” on page 55.
- ▶ Configure thread and connection pools to enable sufficient concurrency. This configuration is important for high-volume, highly concurrent workloads because the thread pool settings directly influence how much work the server can concurrently processes. For more information, see “Configuring thread pool sizes” on page 58.
- ▶ Use a fast network connection between IBM Process Designer and IBM Process Center. The Process Designer communicates frequently with the Process Center, so minimizing network latency is essential.
- ▶ Do the following for business processes that use Business Process Execution Language (BPEL) for Web services:
 - Where possible, use non-interruptible processes (also known as *microflows* or *short-running processes*) instead of long-running processes (also known as *macroflows* or *long-running processes*). Many processes need macroflows (for example, if human tasks are employed or a state needs to be persisted). However, a significant amount of performance resources is associated with macroflows. If some portion of the solution needs macroflows, separate the solution into both microflows and macroflows to maximize use of microflows. For details, see “Choosing microflows where possible” on page 10.
 - For task and process list queries, use composite query tables. Query tables are designed to produce excellent response times for high-volume task and process list queries. For details, see “Choosing query tables for task list and process list queries” on page 10.

- Use Work Manager-based navigation to improve throughput for long-running processes. This optimization reduces the number of objects allocated, the number of objects retrieved from the database, and the number of messages sent for Business Process Choreographer messaging. For more information, see 4.5.1, “Tuning Work Manager-based navigation for business processes ” on page 65.
- Avoid using asynchronous invocations unnecessarily. You often need synchronous invocation on the edges of modules, but not within a module. Use synchronous preferred interaction styles, as described in “Setting the Preferred Interaction Style to Synchronous when possible” on page 39.
- Avoid overly granular transaction boundaries in Service Component Architecture (SCA) and BPEL. Every transaction commit results in expensive database and messaging operations. Design your transactions with care, as described in 3.2.6, “Transactional considerations” on page 37.

2.2 Modeling

This section describes best practices for modeling. BPM V7.5 Advanced Edition offers two authoring environments:

- ▶ *Process Designer* is used to model and run high-level Business Processing Modeling Notation (BPMN) business processes, which often involve human interactions. The Process Designer is the only authoring tool for BPM V7.5 Standard Edition.
- ▶ *Integration Designer* is used to build and implement services that are automated or start other services. These services include Web services, enterprise resource applications, or applications running in IBM CICS® and IBM IMS™, which exist in the enterprise. Integration Designer is also the tool to use to author BPEL business processes.

These authoring environments both interact with the Process Center, which is a shared repository and runtime environment.

Two individuals with separate roles and skill sets work together to develop business process management applications using these environments:

- ▶ The *business author* is responsible for authoring all business processes. The business author is able to use services but is not interested in the implementation details or how they work. The business author uses Process Designer to create business process diagrams (BPDs) and advanced integration services (AISs).
- ▶ The *integration programmer* is responsible for doing all of the integration work necessary to support the processes the business author creates. For example, the integration programmer implements all the AISs and produces mappings between back-end formats and the requirements of current applications. The integration programmer uses Integration Designer.

The rest of this section is organized based on user type, with separate sections describing common best practices and best practices for Process Designer (for business authors) and Integration Designer (for integration programmers).

2.2.1 Common best practices

This section outlines general measures to take in designing and configuring elements of BPM V7.5.

Choosing the appropriate granularity for a process

A business process and its individual steps should have business significance and not try to mimic programming-level granularity. Use programming techniques such as Plain Old Java Objects (POJOs) or Java snippets for logic without business significance. This material is explained further in the IBM developerWorks® article *Software components: Coarse-grained versus fine-grained*, available at the following website:

<http://www.ibm.com/developerworks/webservices/library/ws-soa-granularity/>

Using events judiciously

The purpose of event emission in BPM V7.5 is business activity monitoring. Because event emission uses a persistent mechanism, it inherently has high performance processor resources. Use Common Base Events for events that have business relevance only. Do not emit events to a database. Instead, perform event emission through the messaging infrastructure. Do not confuse business activity monitoring and IT monitoring. The Performance Monitoring Infrastructure (PMI) is far more appropriate for IT monitoring.

The following principles generally hold for most customers:

- ▶ Customers are concerned about the state of their business and their processes. Therefore, events that signify changes in state are important. For long-running and human task activities, this change in state is fairly common. Use events to track when long-running activities complete, such as when human tasks change state.
- ▶ For short-running flows that complete within seconds, it is sufficient to know that a flow completes, perhaps with the associated data. It usually makes no sense to distinguish events within a microflow that are only milliseconds or seconds apart. Therefore, two events (start and end) are sufficient for a microflow.

2.2.2 Process Designer architecture best practices

This section presents best practices for the Process Designer.

Using a fast connection between Process Designer and Process Center

The Process Designer interacts frequently with the Process Center for authoring tasks. For this reason, minimize network latency to provide optimal response times. Place the Process Center in the same physical location as the Process Designer users. If you cannot relocate the Process Center, you can remotely connect to the environment where the Process Center is physically located and use the Process Designer through that mechanism.

Developing efficient tasks

Here are guidelines for developing efficient tasks:

- ▶ Where possible, call system lane tasks with Service No Task so that the tasks are not written to the database.
- ▶ Limit business data searches to only those fields you know that you need to be searchable in the process portal.

Managing variable usage

Variables are persisted to the database when execution contexts are saved, which happens fairly frequently (that is, when changing from BPD to service execution and when running each coach). These persistence operations are expensive. Minimize the persistence cost in the following ways:

- ▶ Minimize the number of variables used.
- ▶ Minimize the size of each variable.
- ▶ Set variables (like DB result sets) to null when no longer needed.
- ▶ Minimize the number and size of variables passed to each task.
- ▶ If industry standard schema are being used (for example, ACORD or HIPAA), recognize that these schema contain many fields and use only the variables that are required from the schema. If necessary, convert to or from the industry standard schema. However, only on the edges of the application to avoid unnecessary persistence costs in the BPD processing.

Turning off Auto Tracking in business process diagrams if not required

Auto Tracking is enabled by default for BPDs. This capability is important for many BPDs because it enables the gathering, tracking, and reporting of key business metrics. However, there is some cost due to Auto Tracking because the events are processed by the Performance Data Warehouse and persisted in the database. Disable Auto Tracking for BPDs that do not require tracking and reporting business metrics.

Avoiding business process diagrams that run forever

Some BPDs run forever. Although certain business processes must run perpetually, design this type of BPDs only if the capability is strictly required. BPDs that run perpetually continually poll for new events, which uses server processor resources. Consider using other communication mechanisms (such as Java Message Service queues) instead of polling. If polling is necessary, then use an undercover agent (UCA) instead of a BPD to do the polling. Also, disable Auto Tracking for these BPDs to avoid excessive traffic to the Performance Data Warehouse.

Developing efficient coaches

Here are some guidelines for developing well-performing coaches:

- ▶ Use custom visibility sparingly
- ▶ Avoid large, complex coaches
- ▶ Avoid large, repeating tables, page the results instead

Minimizing use of large JavaScript scripts

Avoid large JavaScript blocks because JavaScript must be interpreted and therefore is slower to process than other mechanisms such as Java code. Large JavaScript blocks can also produce very large Document Object Model (DOM) trees, which are expensive for browsers to process and render. Finally, large JavaScript blocks are often indicative of too much logic being placed in the BPM layer. Thus, refactoring the solution to use less JavaScript is better.

2.2.3 Integration Designer best practices

This section describes best practices for using Integration Designer.

Choosing microflows where possible

Use macroflows only where required (for example, for long-running service invocations and human tasks). Microflows exhibit much better performance at run time. A non-interruptible microflow instance is run in one J2EE transaction with no persistence of state. However, an interruptible macroflow instance is typically run in several J2EE transactions, requiring that state persist in a database at transaction boundaries.

Where possible, use synchronous interactions for non-interruptible processes. A non-interruptible process is more efficient than an interruptible process because it does not must use state or persistence in the backing database system.

To determine whether a process is interruptible, in the Integration Designer, click **Properties** → **Details**. A process is interruptible if the **Process is long-running** check box is selected.

If interruptible processes are required for some capabilities, separate the processes so that non-interruptible processes can handle the most frequent scenarios and interruptible processes handle exceptional cases.

Choosing query tables for task list and process list queries

Query tables are designed to provide good response times for high-volume task lists and process list queries. Query tables offer improved query performance in the following ways:

- ▶ Improved access to work items, reducing the complexity of the database query
- ▶ Configurable high-performance filters for tasks, process instances, and work items
- ▶ Composite query tables to bypass authorization through work items.
- ▶ Composite query tables that allow query table definitions reflecting information shown in task lists and process lists

For more information, see the following references:

- ▶ PA71: Business Process Manager Advanced - Query Table Builder
<http://www.ibm.com/support/docview.wss?uid=swg24021440>
- ▶ Query tables in Business Process Choreographer in the IBM BPM 7.5 Information Center
http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r5mx/index.jsp?topic=%2Fcom.ibm.wbpm.bpc.doc%2Ftopics%2Fc6bpe1_querytables.html

Choosing efficient metadata management

This section describes best practices for metadata usage.

Following Java language specification for complex data type names

The WebSphere Process Server allows characters in business object (BO) type names that are permissible in Java class names (the underscore, '_', for example). However, the internal data representation of complex data type names uses Java types. As such, performance is better if BO types follow the Java naming standards because valid Java naming syntax requires no additional translation.

Avoiding use of anonymous derived types in XML schema definitions

Some XML Schema Definition (XSD) features (restrictions on the primitive string type, for example) result in modifications to the type that require a new subtype to be generated. If these types are not explicitly declared, a new subtype (a derived type) is generated at run time. Performance is generally better if this situation can be avoided. Avoid adding restrictions to elements of primitive type where possible. If a restriction is unavoidable, consider creating a new, concrete SimpleType that extends the primitive type to include the restriction. Then XSD elements might use that type without degraded performance.

Avoiding references to elements in one XML schema definition from another

Avoid referencing an element in one XSD from another.

For example, if A.xsd defines an element, AElement (shown in Example 2-1), it might be referenced from another file, B.xsd (shown in Example 2-2).

Example 2-1 AElement XSD

```
<xs:element name="AElement">
  <xs:simpleType name="AElementType">
    <xs:restriction base="xs:string">
      <xs:minLength value="0" />
      <xs:maxLength value="8" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Example 2-2 AElement referenced from another file

```
<xs:element ref="AElement" minOccurs="0" />
```

This practice often performs poorly. It is better to define the type concretely and make any new elements use this type. Thus, A.xsd takes the form shown in Example 2-3.

Example 2-3 AElementType XSD

```
<xs:simpleType name="AElementType">
  <xs:restriction base="xs:string">
    <xs:minLength value="0" />
    <xs:maxLength value="8" />
  </xs:restriction>
</xs:simpleType>
```

B.xsd takes the form shown in Example 2-4.

Example 2-4 BElement XSD

```
<xs:element name="BElement" type="AElementType" minOccurs="0" />
```

Reusing data object type metadata where possible

Within application code, it is common to refer to types, for example, when creating a BO. It is possible to refer to a BO type by name, for example in the method `DataFactory.create(String URI, String typeName)`.

You can also refer to the type by a direct reference, such as in the method `DataFactory.create(Type type)`. In cases where a type is likely to be used more than once, it is faster to retain the type, for example, through `DataObject.getType()`, and reuse that type for future use.

Choosing between business state machines

Business state machines (BSMs) provide an attractive way of implementing business flow logic. For some applications, it is more intuitive to model the business logic as a state machine, making the resultant artifacts easier to understand. However, a BSM is implemented using the business process infrastructure, so there is always a performance impact when choosing BSM over business processes.

If an application can be modeled using either BSM or business processes, and performance is a differentiating factor, choose business processes. Also, more options are available for optimizing business process performance than for BSM performance.

Minimizing state transitions in business state machines

Where possible, minimize external events to drive state transitions in BSMs. External event-driven state transitions are costly from a performance perspective. In fact, the total time it takes to run a BSM is proportional to the number of state transitions that occur during the life span of the BSM.

For example, if a state machine transitions through states $A \rightarrow B \rightarrow B \rightarrow B \rightarrow C$ (four transitions), it is twice as time-consuming as making transitions through states $A \rightarrow B \rightarrow C$ (two transitions). Also, automatic state transitions are much less costly than event-driven state transitions. Take these principles into consideration when designing a BSM.

2.3 Topology

In this section, we provide information about choosing an appropriate topology for your solution.

2.3.1 Deploying appropriate hardware

It is important to pick a hardware configuration that contains the resources necessary to achieve high performance in a BPM environment. The following factors are key considerations in picking a hardware configuration:

- ▶ Cores

Ensure that BPM servers (Process Servers and Process Centers) are installed on a modern server system with multiple cores. Each product scales well, both vertically in terms of symmetric multiprocessing (SMP) scaling, and horizontally, in terms of clustering.

- ▶ Memory

BPM servers benefit from both a robust memory subsystem and an ample amount of physical memory. Ensure that the chosen system has server-class memory controllers and as large as possible L2 and L3 caches (optimally, use a system with at least a 4 MB L3 cache). Make sure that there is enough physical memory for all the combined applications (JVMs) that are expected to run concurrently on the system. For 64-bit JVMs (the published suggested configuration), 4 GB per JVM is needed if the maximum heap size is 3 GB or less. Add additional physical memory for heap sizes larger than 3 GB. When using a 32-bit JVM, a rough guideline is 2 GB per JVM instance.

- ▶ Disk

Ensure that the systems hosting the message and data stores (typically, the database tiers) have fast storage. Fast storage means using Redundant Array of Independent Disks (RAID) adapters with writeback caches and disk arrays with many physical drives.

- ▶ Network

Ensure that the network is sufficiently fast enough not to create a system bottleneck. For example, a dedicated Gigabit Ethernet network is a good choice.

- ▶ Virtualization

Be careful when using virtualization such as IBM AIX® dynamic logical partitioning or VMware virtual machines. Ensure sufficient processor, memory, and I/O resources are allocated to each virtual machine or logical partition (LPAR). Avoid overcommitting resources.

2.3.2 Deploying local modules in the same server

If you plan to deploy modules on the same physical server, you can achieve better performance by deploying the modules to the same application server JVM. The server can then take advantage of this locality.

2.3.3 Best practices for clustering

See IBM Redbooks publication *IBM Business Process Manager V7.5 Production Topologies*, SG24-7976, for information about best practices for clustering. This document provides comprehensive guidance on selecting appropriate topologies for both scalability and high availability. You can retrieve this document from the following location:

<http://www.redbooks.ibm.com/redbooks/pdfs/sg247976.pdf>

It is not the intent of this section to repeat content from that book. Rather, this paper distills some of the key considerations when scaling up a topology for maximum performance.

Using the remote messaging and remote support deployment pattern

Use the remote messaging and remote support deployment environment pattern for maximum flexibility in scaling. For more information, see the section “Topologies and Deployment environment patterns” in the *IBM Business Process Manager Advanced Installation Guide* at the following location:

ftp://ftp.software.ibm.com/software/integration/business-process-manager/library/imuc_ebpm_dist_pdf.pdf

This topology (formerly known as the “Gold Topology”) prescribes the use of separate clusters for applications and messaging engines. The topology also prescribes how clusters are used for support applications servers (such as the Common Event Infrastructure (CEI) server or the Business Rules Manager). This topology allows independent control of resources to support the load on each of these elements of the infrastructure.

Flexibility and cost: As with many system choices, flexibility comes with some costs. For example, synchronous event emission between an application and the CEI server in this topology is a remote call, which is heavier than a local call. The benefit of this configuration is the independent ability to scale the application and support cluster. We assume that the reader is familiar with these kinds of system tradeoffs as they occur in most server middleware.

Considering single server versus clustered topology

In general, there are two primary issues to consider when evaluating whether to move to a clustered topology from a single server configuration:

- ▶ Scalability and load balancing to improve overall performance and throughput
- ▶ High availability by using failover to another cluster member to prevent loss of service due to hardware or software failures

Although not mutually exclusive, each requires certain considerations. This paper focuses on the performance (throughput) aspects of clustering and not on the high availability aspects. Most single server workloads that are driving resources to saturation can benefit by moving to a clustered topology.

2.3.4 Evaluating service providers and external interfaces

One typical usage pattern for BPM V7.5 is as an integration layer between incoming requests and back-end systems for the business (target applications or service providers). In these scenarios, the throughput is limited by the layer with the lowest throughput capacity.

Consider the simple case where there is only one target application. The BPM V7.5-based integration solution cannot achieve throughput rates higher than the throughput capacity of the target application. This inability to increase throughput applies regardless of the efficiency of the BPM V7.5-based implementation or the size or speed of the system hosting it. Thus, it is critical to understand the throughput capacity of all target applications and service providers and apply this information when designing the end-to-end solution.

There are two key aspects of the throughput capacity of a target application or service provider:

- ▶ Response time, both for typical cases and exceptional cases
- ▶ Number of requests that the target application can process at the same time (concurrency)

If you can establish performance aspects of the target applications, you can calculate a rough estimate of the maximum throughput capacity. Similarly, if average throughput is known, you can also roughly calculate either one of these two aspects as well. For example, a target application that can process 10 requests per second with an average response time of one second, can process approximately 10 requests at the same time (throughput / response time = concurrency).

The throughput capacity of target applications is critical to projecting the end-to-end throughput of an entire application. Also, consider the concurrency of target applications when tuning the concurrency levels of the upstream BPM V7.5-based components. For example, if a target application can process 10 requests at the same time, tune the process server components that start this application. By tuning these components, the simultaneous request from BPM V7.5 at least matches the concurrency capabilities of the target.

Additionally, avoid overloading target applications because such configurations do not result in any increase in overall application throughput. For example, if 100 requests are sent to a target application that can process only 10 requests at the same time, throughput does not improve. However, throughput does improve by tuning such that the number of requests made matches the concurrency capabilities of the target.

Service providers that might take a long time to reply, either as part of mainline processing or in exception cases, do not use synchronous invocations that require a response. Not using synchronous invocations avoids tying up the business process and its resources until the service provider replies.

2.4 Business Space

This section documents best practices for building, developing, and deploying Business Space solutions.

2.4.1 Optimizing topology

The performance of Asynchronous JavaScript and XML (AJAX) applications like Business Space can be broken down to a four-tiered model, as shown in Figure 2-1. Each of these tiers must be optimized to deliver a high-performing solution. Several details for optimizing the topology are described later in this paper, but a context for such a description is presented here.

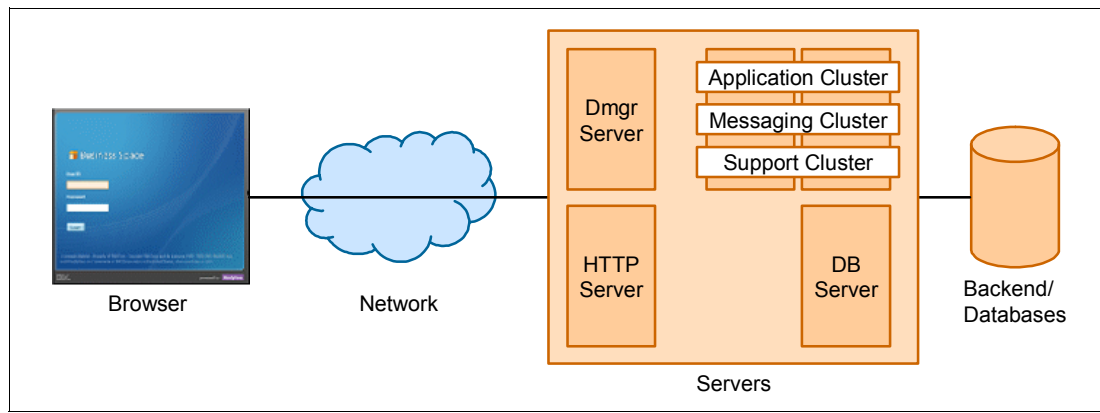


Figure 2-1 Four tiers of AJAX performance

The four tiers are listed here:

► **Browser**

AJAX applications are, by definition, applications that perform work (to some extent) on the client side, inside the browser. All typical client work, like building the UI, is done on the client side, which differentiates these applications from classical web applications, where only the rendering of HTML is done in the browser. As described in the following subsections of 2.4, “Business Space” on page 16, optimizing the browser and client system are key to delivering excellent performance.

► **Network**

In contrast to static web pages, AJAX applications load the content of a page dynamically as needed, instead of loading the complete page immediately. Instead of a few (or one) requests with significant payloads, AJAX applications often send many requests with smaller payloads. Hence, delays in the network can have a significant impact on Business Space response times because they add time to each message. In the end, network delays add up to a bigger delay for the overall page response time.

Figure 2-1 is simplified because the network also plays a role in communications between the servers and the databases and even between servers in a clustered setup. However, due to the nature of AJAX applications, the first point to analyze for delays is generally between the browser and the servers.

► Servers

The server infrastructure is responsible for handling the requests from the clients that are attached to it, running business applications (processes, state machines, Web services, and other applications) and integrating back-end services. The setup of this infrastructure heavily depends on the chosen topology.

The following servers play an important role for Business Space:

– HTTP server

An HTTP server is not part of all topologies. However, for environments that aim to serve thousands of users, an HTTP server is indispensable. All static and dynamic requests from Business Space clients come to the HTTP server. This server can cache and then send the static (cached) content back to the client and route dynamic requests to the WebSphere Process Server REST API. Furthermore, an HTTP server can provide load balancing and support high-availability scenarios.

– BPM V7.5 server

BPM V7.5 servers execute various business logic (such as BPEL and BPMN business processes). However, in this section we focus only on actions that play a role for the Business Space widgets, such as querying task lists, creating and claiming tasks, and other functions.

► Databases

Two databases often play a key role for BPM V7.5 Business Space powered by WebSphere solutions:

– Business Space powered by WebSphere database

The Business Space database holds configuration-related information and is infrequently accessed by each user.

– Business Process Choreographer database (BPEDB)

All types of process and task information are stored in the BPEDB, so the human task widgets frequently update and read this database.

2.4.2 Using a network with low latency and high bandwidth

Again, Business Space V7.5 is an AJAX application. As such, it uses multiple HTTP requests per user action, so minimizing the network delay per request is crucial. Network bandwidth is also crucial because some HTTP requests return a significant amount of data.

Where possible, use a high-speed and high-bandwidth (1 Gb or faster) dedicated network for connectivity between Business Space clients and servers.

2.4.3 Using a high-performing browser

The choice of browser technology is crucial to Business Space performance. The choice of browsers is important in the case of Internet Explorer (IE); more recent versions of IE typically perform much better than older versions of IE.

2.4.4 Enabling browser caching

Browsers generally cache static data after it is initially retrieved from the server, which can significantly improve response time for scenarios after the cache is primed. This improvement is especially true for networks with relatively high latency. Ensure that the browser cache is active and is effective. Cache settings are browser-specific; see 4.9.1, “Ensuring browser cache is enabled in Internet Explorer 8” on page 71 for further details.

2.4.5 Locating servers physically near clients

One factor that influences network latency is the physical distance between servers and clients and also the distance between servers in a cluster (for example, between the support cluster and the BPM V7.5 application servers). When practical, locate BPM V7.5 servers physically close to each other and to the Business Space clients to minimize the latency for requests.

2.4.6 Using modern desktop hardware

For Business Space solutions, much of the processing is done on the client system (through browser rendering). Thus it is imperative to deploy modern desktop hardware with sufficient physical memory and high-speed processors with large caches and fast front-side buses. Monitor your client systems with performance tools (Windows Task Manager or vmstat) to ensure that the client system has sufficient processor and memory resources to ensure high performance.

2.5 Large objects

One issue frequently encountered by field personnel is identifying the largest object size that the BPM V7.5 server and corresponding WebSphere adapters can effectively and efficiently process. A number of factors affect large object processing in each of these products. This section presents both a description of the factors involved and practical guidelines for the current releases of these products. The issue of identifying the largest object size primarily applies to 32-bit JVMs because of the constraints on heap sizes in that environment.

The JVM is the single most important factor affecting large object processing. JVM technology changed starting with Java 5, which was first used by the BPM products in V6.1.0. BPM V7.5 uses the Java 6 JVM, which is the follow-on release to Java 5. The suggestions and best practices in this document differ from those in BPM V6.0.2 and earlier, which used JVM 1.4.2.

In general, objects 5 MB or larger might be considered “large” and require special attention. Objects 100 MB or larger are “very large” and generally require significant tuning to be processed successfully.

2.5.1 Factors affecting large object size processing

In general, the object size capacity for any installation depends on the size of the Java heap and the load placed on that heap (that is, the live set) by the current level of incoming work. The larger the heap, the larger the BO that can be successfully processed.

To apply this principle, you must first understand that the object size limit is based on three fundamental implementation facts about JVMs:

- ▶ Java heap size limitations

The limit to the size of the Java heap is operating system-dependent, but it is not unusual to have a heap size limit of around 1.4 GB for 32-bit JVMs. The heap size limit is much higher on 64-bit JVMs and is typically less of a gating factor on modern hardware configurations than the amount of available physical memory. Further details about heap sizes are described in “Increasing the Java heap size to its maximum” on page 56.

- ▶ Size of in-memory BOs

BOs, when represented as Java objects, are much larger in size than when represented in wire format. For example, a BO that uses 10 MB on an input Java Message Service (JMS) message queue might result in allocations of up to 90 MB on the Java heap. This condition results from many allocations of large and small Java objects occurring as BO flows through the adapters and BPM V7.5. A number of factors affect the in-memory expansion of BOs:

- The single-byte binary wire representation is generally converted to multi-byte character representations (for example, Unicode), resulting in an expansion factor of two.
- The BO might contain many small elements and attributes, each requiring a few unique Java objects to represent its name, value, and other properties.
- Every Java object, even the smallest, has a fixed need for resources due to an internal object header that is 12 bytes long on most 32-bit JVMs, and larger on 64-bit JVMs.
- Java objects are padded to align on 8-byte or 16-byte address boundaries.
- As the BO flows through the system, it might be modified or copied, and multiple copies might exist at any time during the end-to-end transaction. Having multiple copies of the BO means that the Java heap must be large enough to host all these BO copies for the transaction to complete successfully.

- ▶ Number of concurrent objects being processed

The size of an object that can be successfully processed decreases as the number of requests being processed simultaneously increases because each request has its own memory usage profile (liveset) as it makes its way through the system. Simultaneously processing multiple large objects dramatically increases the amount of memory required because the total of livesets for the request must fit into the configured heap.

2.5.2 Large object design patterns

The following sections describe the two proven design patterns for processing large objects successfully. In cases where neither design pattern can be applied, consider 64-bit mode. See 2.6, “64-bit considerations” on page 20 for details.

Also, for IBM WebSphere ESB, a developerWorks article is available detailing best practices and tuning for large messages at the following website:

<http://www.ibm.com/developerworks/webservices/library/ws-largemessaging/>

Batched inputs: Sending large objects as multiple small objects

If a large object needs to be processed, the solutions engineer must find a way to limit the number of allocated large Java objects. The primary technique for limiting the number of objects involves decomposing large BOs into smaller objects and submitting them individually.

If the large objects are actually collections of small objects, the solution is to group the smaller objects into conglomerate objects less than 1 MB in size. Several customer sites have consolidated small objects in this way, producing good results. If temporal dependencies or an all-or-nothing requirement for the individual objects exists, the solution becomes more complex. Implementations at customer sites demonstrate that dealing with this complexity is worth the effort as demonstrated by both increased performance and stability.

Certain WebSphere adapters (such as the Flat Files adapter) can be configured to use a SplitBySize mode with a SplitCriteria set to the size of each individual object. In this case, a large object is split in chunks (of a size specified by SplitCriteria) to reduce peak memory usage.

Claim check pattern: Only a small portion of an input message is used

When the input BO is too large to be carried around in a system and that process or mediation needs only a few attributes, you can use a pattern called the *claim check pattern*. Using the claim check pattern, as applied to a BO, involves the following steps:

1. Detach the data payload from the message.
2. Extract the required attributes into a smaller control BO.
3. Persist the larger data payload to a data store and store the “claim check” as a reference in the control BO.
4. Process the smaller control BO, which has a smaller memory footprint.
5. When the solution needs the whole large payload again, check out the large payload from the data store using the key.
6. Delete the large payload from the data store.
7. Merge the attributes in the control BO with the large payload, taking into account the changed attributes in the control BO.

The claim check pattern requires custom code and snippets in the solution. A less developer-intensive variant is to use custom data bindings to generate the control BO. This approach is limited to certain export and import bindings. The full payload still must be allocated in the JVM.

2.6 64-bit considerations

You can run BPM applications using either 32-bit or 64-bit JVMs. However, we suggest 64-bit mode for both Process Server and Process Center because in 32-bit mode, the maximum heap size is limited by the 4 GB address space size. In most 32-bit operating systems, the practical limit varies between 1.5 GB and 2.5 GB. In contrast, although maximum heap size is limitless in 64-bit mode, standard Java best practices still apply.

The sum of the maximum heap sizes and native memory use of all the Java processes running on a system should not exceed the physical memory available on the system. This total also includes additional memory required for the operating system and other applications. Java processes include threads, stacks, and Just In Time (JIT) compiled code.

BPM V7.5 servers run most efficiently on a 64-bit JVM instance because of the much larger amount of memory that is accessible in this mode. The performance and memory footprint of a 64-bit runtime server is about the same as the 32-bit version. However, the footprint for 64-bit JVMs is not as good as the 32-bit performance and memory footprint in BPM V6.1 and V6.2.

The following list details the factors to consider when determining in which of these modes to run:

- ▶ 64-bit mode is an excellent choice for applications whose liveset approaches or exceeds the 32-bit limits. Such applications either experience `OutOfMemoryExceptions` or suffer excessive time in garbage collection (GC). We consider anything greater than 10% of the time in GC as excessive. These applications exhibit much better performance when allowed to run with the larger heaps they need. However, sufficient physical memory on the system must exist to back the Java heap size.
- ▶ 64-bit mode is also an excellent choice for applications that, though well-behaved in 32-bit mode, can be algorithmically modified to perform better with larger heaps. An example might be an application that frequently persists data to a data store to avoid maintaining a very large in-memory cache, even if such a cache greatly improves throughput. Recoding such an application to trade off the more space available in 64-bit heaps for less execution time yields better performance.
- ▶ Moving to 64-bit can cause some degradation in throughput if a 32-bit application fits well with a 1.5 GB to 2.5 GB heap and the application is not expected to grow significantly. For these situations where the memory limitation is not a significant factor, using 32-bit JVMs might be a better choice than 64-bit.

2.7 Business Monitor

This section describes best practices for performance with Business Monitor V7.5.

2.7.1 Event processing

A major factor in event processing performance is tuning the Business Monitor database. Ensure adequate buffer pool sizes to minimize disk reading activity and placement of the database logs, which ideally is on a physically separate disk subsystem from the database table spaces.

By default, events are delivered from CEI to the monitor database, bypassing an intermediate queue. We suggest using this default delivery style for better performance as it avoids an additional persistence step in the flow.

For additional background, see “Receiving events using table-based event delivery” in the IBM Business Monitor V7.5 Information Center:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r5mx/index.jsp?topic=/com.ibm.wbpm.mon.imuc.doc/inst/cfg_qb.html

2.7.2 Dashboard

The platform requirements of the Business Space and its Business Monitor widgets are relatively modest compared to the widgets of Business Monitor server and the database server. The most important consideration for good dashboard performance is to size and configure the database server correctly. Be sure that it has enough processor capacity for anticipated data mining queries, enough memory for buffer pools, and plenty of disk arms.

2.7.3 Database server

Both event processing and dashboard rely on a fast, well-tuned database server for good performance. The design of the Business Monitor assumes that any customer using it has strong on-site database administrator skills. It is important to apply the database tuning suggestions beginning in 4.10, “General database tuning” on page 78.



Development best practices

This chapter presents best practices that are relevant to the solution developer. It addresses modeling, design, and development choices that are made while designing and implementing a BPM V7.5 solution. Business Process Manager (BPM) Advanced Edition offers two authoring environments. These authoring environments both interact with the Process Center, which is a shared repository and runtime environment.

- ▶ In the *Process Designer* environment, you can model and run high-level business processes, which often have human interactions. The Process Designer is the only authoring tool for BPM V7.5 Standard Edition.
- ▶ In the *Integration Designer* environment, you can build and implement services that are automated or that start other services, such as Web services, enterprise resource applications, or applications running in CICS and IMS. These services and applications exist in the enterprise. It is also the tool to use to author Business Process Execution Language (BPEL) business processes.

Two individuals with separate roles and skill sets work together when developing BPM applications. These roles correspond to the Process Designer and Integration Designer environments.

- ▶ The *business author* is responsible for authoring all business processes. The business author is able to use services but is not interested in the implementation details or how they work. The business author uses Process Designer to create business process diagrams (BPDs), and advanced integration services (AISs) to collaborate with the integration programmer.
- ▶ The *integration programmer* is responsible for doing all of the integration work necessary to support the processes the business author creates. For example, the integration programmer implements all the AISs and produces mappings between back-end formats and the requirements of current applications. The integration programmer uses Integration Designer.

The rest of this chapter is organized based on the type of user, with separate sections describing Process Designer (business author) and Integration Designer (integration programmer) best practices. Additionally, the chapter provides developer considerations for Business Space solutions, WebSphere InterChange Server migration, and Authoring environment.

3.1 Process Designer development best practices

The following best practices pertain to the development of high-performance business processes using the Process Designer.

3.1.1 Clearing variables in exposed human services not intended to end

Data from a taskless human service is not garbage-collected until the service reaches the endpoint. If a human service is developed that is not intended to reach an endpoint, such as a single page or redirect, then memory is not garbage-collected until the Enterprise JavaBeans (EJB) timeout occurs (two hours by default). To reduce memory use, set variables in the coach to null in a custom HTML block.

3.1.2 Not using multi-instance loops in the system lane or for batch activities

Use caution when using sub-BPDs as the activity of a multi-instance loop (MIL). There is no issue if the first activity is a user task instead of a system lane. However, do not try to use MILs to run batch or system lane activities. This pattern can generate an excessive number of tokens for the BPD to process. Also, activities in MILs in the system lane are run on a single thread, which is clearly not optimal on multi-processor core servers.

Figure 3-1 shows an example of a poor BPD design pattern.

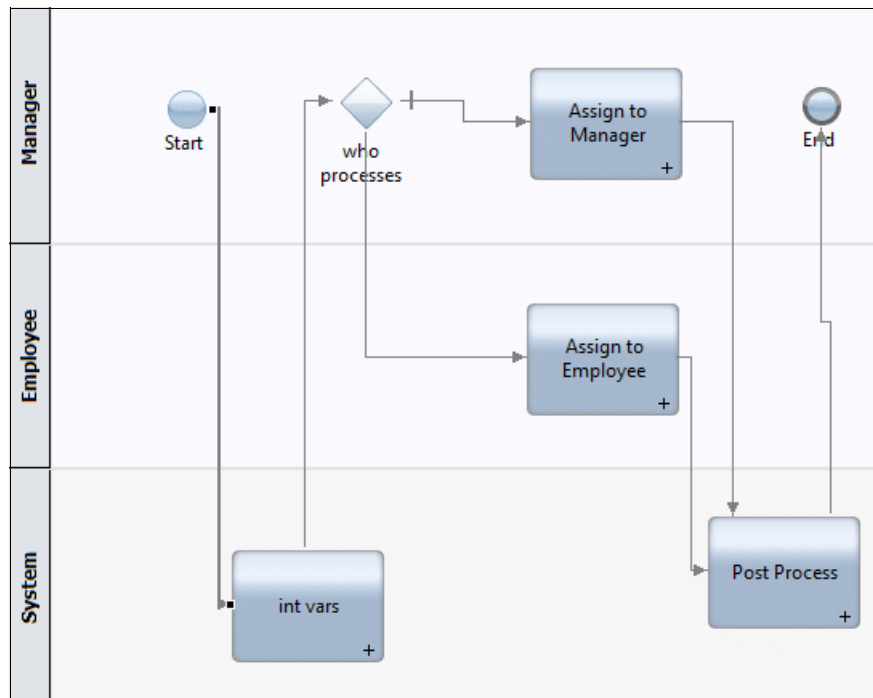


Figure 3-1 Poor BPD design pattern

Figure 3-2 shows an example of a good BPD design pattern.

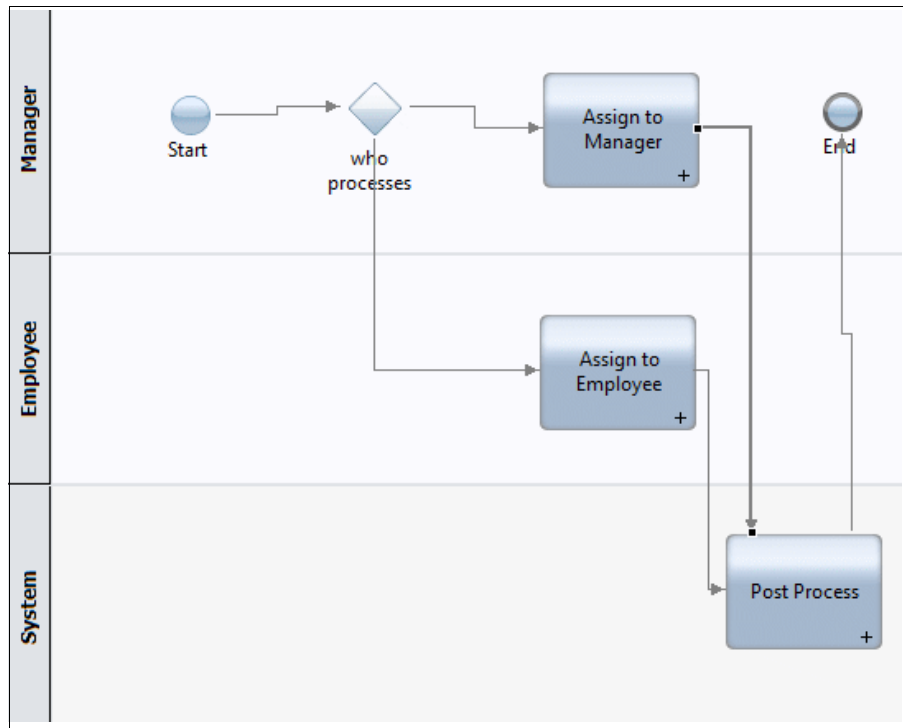


Figure 3-2 Good BPD design pattern

3.1.3 Using conditional joins only when necessary

Simple joins use an “and” condition; all lines heading into the join must have an active token for the tokens to continue forward.

By contrast, for conditional joins, all possible tokens must reach the join before they proceed. Thus, if you have a conditional join with three incoming lines, but only two of them currently have tokens (or *might* have tokens by looking upstream), then those two tokens must arrive at the join to proceed. To determine this condition, the BPD engine must evaluate all possible upstream paths to determine whether the tokens can arrive at the join. This evaluation can be expensive for large, complex BPDs. Use this capability judiciously in these cases.

3.1.4 Guidelines for error handling

Avoid global error handling in a service, which can use an excessive amount of server processor and can even result in limitless loops in coaches.

When catching errors on an activity in a BPD, do not route the error back to the same activity. Doing so causes the server to thrash between the BPD and service engine, using a large amount of processor and database processing.

3.1.5 Using sequential system lane activities efficiently

Each system lane activity is considered a new Event Manager task, which adds a task transition in the service engine. These task transitions are expensive. If your BPD contains multiple system lane service tasks in a row, use one system lane task that wraps the others to minimize the extra resources needed for the transition. Using one system lane also applies for multiple consecutive tasks in a participant lane, although that pattern is much less common because generally, an action is only necessary between each task in a participant lane.

Figure 3-3 demonstrate a poor usage pattern (with multiple consecutive system lane activities).

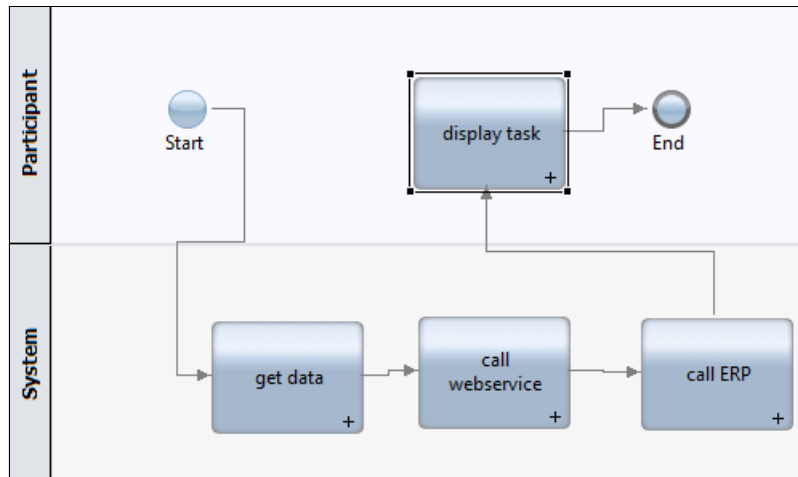


Figure 3-3 Poor BPD usage pattern

Figure 3-4 shows a more optimal usage pattern (one system lane activity that incorporates the multiple activities in Figure 3-3).

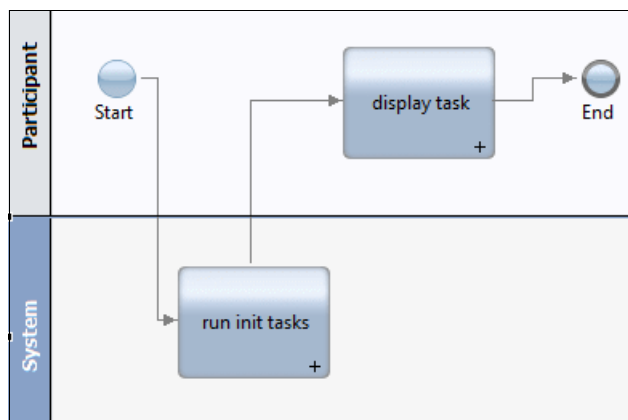


Figure 3-4 Good BPD usage pattern

3.1.6 Ensuring the Process Center is tuned

At a minimum, tune the parameters for the Process Center in the following ways:

- ▶ Use a 64-bit JVM.
- ▶ Use the generational concurrent garbage collector through the `-Xgcpolicy:gencon` JVM argument.
- ▶ Ensure that the Java heap size is sufficiently large to handle the peak load. For more information about setting the Java heap size, see 4.3.2, “Java tuning parameters” on page 52.
- ▶ Set the object request broker (ORB) thread pool size to at least 20.
- ▶ Set the maximum Java Database Connectivity (JDBC) connections for the Process Server data source (`jdbc/TeamWorksDB`) to at least double the maximum number of threads in the object request broker (ORB) thread pools for each node in the cell. For example, if two Process Center nodes exist, and the ORB service thread pool of each node is set to a maximum of 20 threads, then set the Process Server data source to accept at least 80 connections.
- ▶ Tune the Process Center database. For more information, see 4.10, “General database tuning” on page 78.

3.1.7 Using a fast connection between Process Designer and Process Center

The Process Designer interacts frequently with the Process Center for authoring tasks. Take steps to expedite interaction between these components. See “Using a fast connection between Process Designer and Process Center” on page 8 to learn more about the connection between Process Designer and the Process Center.

3.1.8 Preventing Web Services Description Language validation from causing slow web service integrations

The Process Designer Web service integration connector goes through several steps at run time to start a Web service. The system generates the SOAP request from the metadata and the business objects (BOs). The system then validates the request against the Web Services Description Language (WSDL), makes the actual SOAP call over HTTP, and parses the results back into the BOs. Each of these steps potentially has some latency. Therefore, it is important to not only make sure that the actual web service response time is fast, but also that the request can be quickly validated against the WSDL. Speed is especially important for Web services that might be started frequently.

There are two major causes of delays in validation:

- ▶ Slow responses in retrieving the WSDL
- ▶ Deeply nested WSDL includes

If the source of the WSDL is a remote location, the latency of retrieving that WSDL over HTTP adds to the overall latency. Thus, a slow connection, a slow proxy, or a slow server can all potentially increase the latency of the complete web service call. If that WSDL also nests additional WSDL or XML Schema Definition (XSD) files by imports and includes, then after the main WSDL is retrieved, the subfiles must also be retrieved. The validation continues to recurse through all WSDLs or XSDs that are nested within the subfiles. Therefore, in a case where there are multiple levels of nesting, many HTTP calls must make many calls to retrieve the complete WSDL document, and the overall latency can become very high.

To alleviate this type of latency, you can store a local copy of the WSDL either in the local file system or somewhere where HTTP response is fast. For even better performance, this local copy can be "flattened," removing the nesting by manually replacing all of the include statements with the actual content.

3.2 Integration Designer best practices

The following best practices pertain to the development of well-performing solutions through the Integration Designer.

3.2.1 Business object parsing mode considerations

This section describes how and why to choose the BO parsing mode. Two options for this mode exist: *eager* parsing mode and *lazy* parsing mode.

Overview

WebSphere BPM V7.0.0.1 introduced BO lazy parsing mode. In this mode, when BOs are created their properties are not populated until they are accessed by the application. Also, when reading XML input, the stream is incrementally parsed and its properties do not materialize until they are accessed. In contrast, with eager parsing mode, the input XML stream is immediately and fully parsed (with some exceptions for specific mediation primitives) and materializes into a complete in-memory data object representation.

Lazy parsing uses the XML Cursor Interface (XCI) implementation of Service Data Objects (SDOs) provided by the WebSphere Application Server Feature Pack for Service Component Architecture (SCA), which is delivered as part of BPM V7.5. In this mode, the input XML stream remains in memory throughout the lifetime of the corresponding BO. The cursor-based XML parser creates nodes in memory only when the nodes are traversed. Properties and attributes are evaluated only when the application requests them. Parsing the XML stream in this fashion delivers better performance than complete parsing for many applications in eager mode. This improvement is especially evident if only a small portion of the BO is accessed during the application execution.

Lazy parsing also reuses the XML stream when BOs are serialized, for example, for outbound service calls. This results in faster serialization because it is not necessary to serialize the entire in-memory BO into XML strings. Lazy parsing automatically detects namespaces in the output stream that are different from the namespaces in the original input stream and corrects them. This form of serialization is used for all bindings and internal BPM runtime components, such as the Business Process Choreographer (BPC) container, and when custom code calls BO serialization.

To further optimize performance, lazy parsing also employs a technology called *copy-on-write*, where properties in BOs are lazily copied. When the copy operation initially takes place, the target BO just points to node tree of the source BO. Lazy parsing copies no properties. Subsequent modifications of either the source or target BO trigger a split of the affected node in the BO tree, copying BO properties only as needed. Modifications occur in the local copy of the corresponding node. Copy-on-write is transparent to the application.

Setting parsing mode

In BPM V7.5, the default parsing mode is the lazy mode for newly developed applications in Integration Designer V7.5. Applications moved from WPS V7.0.0.3 and earlier releases continue to run in the eager parsing mode. However, for these older applications, you can manually enable lazy parsing mode through Integration Designer, which provides a property for each module and library to configure the XML parsing mode to either eager or lazy.

Figure 3-5 shows the Integration Designer panel that the system displays when creating a library. At this point, you can specify whether it can be included in an eager parsing or lazy parsing module, or both. The default is set to lazy parsing when the library is initially created. When both parsing modes are selected for the library, the module in which the library is located determines the actual parsing mode at run time.

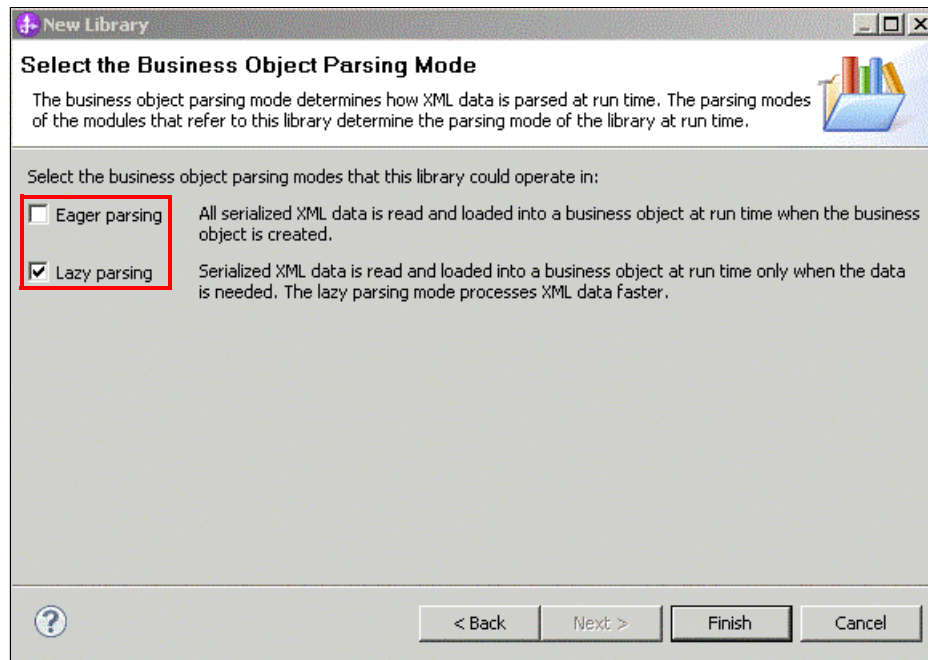


Figure 3-5 Lazy parsing for new library

When creating a module, you must choose either lazy or eager parsing mode. The default selection is lazy parsing mode. Figure 3-6 shows the Integration Designer panel for creating modules.

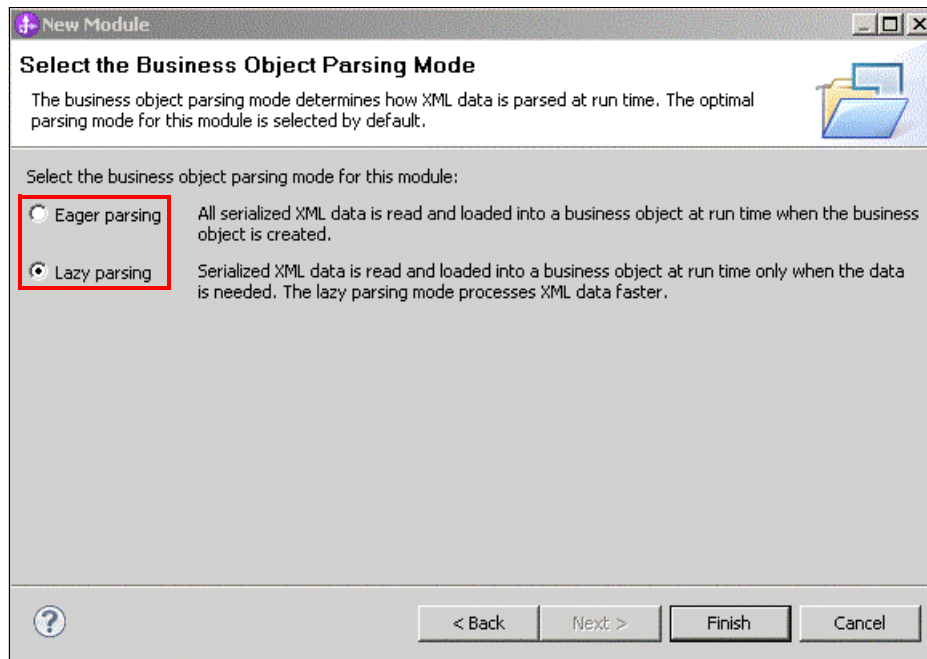


Figure 3-6 Lazy parsing for new module

In BPM V7.5, lazy parsing delivers excellent performance for XML-based applications that use BOs or messages approximately 3 KB or larger. The performance of lazy parsing is generally better than or about equal to eager parsing for other scenarios.

Applications benefiting from lazy parsing

This section provides general guidance for choosing the right parsing mode, and best practices to follow when choosing one mode over the other. However, we suggest that you benchmark your application in both parsing modes to determine which mode best suits the specific characteristics of your application.

The primary performance benefits derived from lazy parsing are delayed parsing, and parsing only the properties accessed by the application. Applications exhibiting any of the following characteristics are candidates for taking advantage of lazy parsing for better performance. As demonstrated in the measurements, the performance improvement when using lazy parsing can be significant.

- ▶ Only applications that use XML data streams can benefit from lazy parsing. For example, applications that use Web services bindings, synchronous or asynchronous SCA bindings, or the Java Message Service (JMS) binding with the XML data handler can potentially benefit from lazy parsing mode.
- ▶ Applications with medium to large XML data streams are likely to see performance improvements when lazy parsing is used. The performance benefit increases as the size of the XML data stream increases. In the workloads we studied, lazy parsing outperformed eager mode when the BOs were approximately 3 KB or larger. The BO size differs depending on the composition of your application, so use these findings as a rule of thumb.

- ▶ Applications that access a small portion of an XML data stream are likely to see greater performance benefits from lazy parsing. Access to a BO is defined as reading the value of a property, updating a property value, or adding a property to a BO. Lazy parsing allows the unmodified portion of the data stream to pass through the system with minimal required resource usage. As opposed to eager parsing, the less an application accesses its BOs, the higher performance improvement it achieves by enabling lazy parsing.
- ▶ WebSphere Enterprise Service Bus (ESB) mediations with multiple mediation primitives are likely to see performance improvements when you use lazy parsing. When using eager parsing, BOs are often serialized before starting the next mediation primitive and then deserialized after that mediation primitive is started. Lazy parsing mode reduces the cost of each serialization and deserialization by preventing unmodified properties from unnecessarily materializing in the data stream.

Applications that fall within these parameters likely benefit from lazy parsing. Applications that do not have those attributes produce little difference in performance and can run in either lazy or eager parsing modes. For example, applications that parse non-XML data streams, such as WebSphere adapters and non-XML data handlers, should perform similarly, whether you choose lazy or eager parsing mode in BPM V7.5. This similarity is also true of applications that create BOs directly using the BOFactory service and applications that use small XML messages (as a rule of thumb, greater than 3 KB).

WebSphere ESB mediation flows that only access the message header or contain a single mediation primitive, might see slightly better performance when using eager parsing. For example, a ServiceGateway with a single Filter mediation routing based on the header content might perform better. However, adding multiple mediation primitives to a flow performance is likely to perform better in lazy parsing mode.

Minimizing use of mixed parsing modes between modules

Sometimes it is necessary to direct some modules to use eager parsing while others use lazy parsing. For better performance, however, avoid using this mixed mode for application modules that frequently interact with each other. For example, if module A starts module B through the synchronous SCA binding on each transaction, it is better to set modules A and B to both use either eager parsing or lazy parsing.

The interaction between modules with different parsing modes takes place through serialization and deserialization, which can negatively affect performance. If using a mixed mode is unavoidable, starting an application from an eager parsing module to a lazy parsing module is more efficient than the other way around. The output from the eager parsing module is serialized into an XML string, and the target module using lazy parsing still benefits fully from delayed parsing, producing this efficiency.

Using share-by-reference libraries where possible

Often, definitions of interfaces, BOs, data maps, mediation subflows, relationships, roles, and web service ports must be shared so that resources in several modules can use them. These resources are stored in the library. Libraries can be deployed in several ways:

- ▶ With a process application
- ▶ With the dependent module
- ▶ Globally

You can make your choice in the dependency editor. If you associate a library with a process application, then you select the application in the editor. The library is shared within the deployed process application, meaning only one copy is in memory. We call this type of library a *share-by-reference library*. The share-by-reference library is a new feature in BPM V7.5.

If you choose to deploy a library with the module, the deployment action creates a copy of the library for each module when the module is deployed. We call this type of library a *share-by-value* library. Deploying the library with the module is the default setting when the library is not associated with a process application.

More details about BPM V7.5 library types can be found in “Libraries” in the Information Center for IBM Business Process Manager, Version 7.5, All platforms:

<http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r5mx/index.jsp?topic=%2Fcom.ibm.wbpm.wid.main.doc%2Fnewapp%2Ftopics%2Fclibrary.html>

Lazy parsing is optimized to use share-by-reference libraries. When a BO travels from one module to another as a parameter of a synchronous SCA service invocation, a lazy copy is used if the source and target modules share the BO schema through a share-by-reference library. Without the share-by-reference optimization, the caller serializes the BO and the callee deserializes the BO.

Differences in behavior for lazy parsing and eager parsing

If you are changing an application that was originally developed using eager parsing to use lazy parsing, be aware of the differences in behavior between each mode, as summarized here;. Also consider the behavior differences between each node if you are planning to switch an application between lazy and eager parsing mode

Error handling

If the XML byte stream being parsed is ill-formed, parsing exceptions occur:

- ▶ With eager parsing, the exceptions occur as soon as the BO is parsed from the inbound XML stream.
- ▶ With lazy parsing, the parsing exceptions occur latently when the application accesses the BO properties and parses the portion of the XML that is ill-formed.

To properly handle ill-formed XML for either parsing mode, select one of the following options:

- ▶ Deploy a mediation flow component on the application edges to validate inbound XML.
- ▶ Author lazy error-detection logic at the point where BO properties are accessed.

Exception stacks and messages

Because eager and lazy parsing have different underlying implementations, stack traces generated by the BO programming interfaces and services have the same exception class name. However, stack traces might not contain the same exception message or wrapped set of implementation-specific exception classes.

XML serialization format

Lazy parsing provides a fast serialization optimization that attempts to copy unmodified XML from the inbound data stream to the outbound data stream upon serialization. The fast serialization improves performance, but the resultant serialization format of the outbound XML data stream might be syntactically different than if the entire BO were updated in lazy parsing, or if eager parsing were used.

The output XML serialization format might not be precisely syntactically equivalent across these cases. However, the semantics of the XML data streams are equivalent, regardless of the parsing mode, and the resultant XML can be safely passed between applications running in different parsing modes with semantic equivalence.

Business object instance validator

The lazy parsing instance validator provides a higher fidelity validation of BOs, particularly for facet validation of property values. Because of these improvements in fidelity, the lazy parsing instance validator catches functional issues not detected in eager parsing mode and provides more detailed error messages.

Migration of XSL style sheets

When an application authored using WebSphere Integration Developer V7.0.0.1 or earlier, the application is migrated using Integration Designer V7.5 and rebuilt. Extensible Stylesheet Language Transformation (XSLT) style sheets are regenerated from the associated map files. Extensible Style Sheet Language (XSL) style sheets built by Integration Designer V7.5 no longer contain whitespace or indent directives. This change improves performance in lazy parsing mode. However, if an XSLT primitive refers directly to manually edited style sheet, the style sheet is not regenerated when the application is built. In this case, performance improves by removing the whitespace directive (`<xsl:strip-space elements="*" />`) if it appears and by ensuring that indentation is disabled (`indent="no"`) unless required.

Private APIs

The BO programming model provides support for BO application programming interfaces (APIs) and a set of Service Data Object (SDO) APIs. In some cases, applications might use additional implementation-specific APIs that are not part of the supported APIs for BOs. For example, an application might use the Eclipse Modeling Framework (EMF) APIs. Although these APIs might have worked in prior BPM releases and in eager parsing mode, they are not supported APIs for accessing BOs. EMF APIs are considered private to the implementation of BOs and should not be used by applications.

For these reasons, remove private APIs from the application and ensure that all BO access takes place using the supported API set.

Service Message Object Eclipse Modeling Framework APIs

A mediation flow component makes it possible to manipulate message content using the Java classes and interfaces provided in the `com.ibm.websphere.sibx.smobo` package. For lazy parsing, the Java interfaces in the `com.ibm.websphere.sibx.smobo` package can still be used. However, methods that refer directly to Eclipse Modeling Framework (EMF) classes and interfaces or that are inherited from EMF interfaces are likely to fail. Also, the Service Message Object (SMO) and its contents cannot be cast to EMF objects when using lazy parsing.

Migration

All applications developed before BPM V7.5 used eager parsing mode by default. When BPM runtime migration moves these applications, they continue to run in eager parsing mode.

To enable an application originally developed using eager parsing to use lazy parsing, first use Integration Designer V7.5 to move the artifacts of the application. After migration, use Integration Designer V7.5 to configure the application to use lazy parsing.

For information about moving artifacts in the Integration Designer, see “Migration source artifacts” in the IBM Business Process Manager Information Center:

<http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r5mx/index.jsp?topic=%2Fcom.ibm.wbpm.wid.imuc.doc%2Ftopics%2Ftmigrsrcartwid.html>

For information about setting the parsing mode, see “Configuring the business object parsing mode of modules and libraries” in the IBM Business Process Manager Information Center:

<http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r5mx/index.jsp?topic=%2Fcom.ibm.wbpm.wid.main.doc%2Fnewapp%2Ftopics%2Ftconfigbo.html>

3.2.2 Service Component Architecture considerations

This section describes the SCA authoring considerations for BMP performance.

Cache results of ServiceManager.locateService()

When writing Java code to locate an SCA service, either within a Java component or a Java snippet, consider caching the result for future use, as service location is a relatively expensive operation. Code generated in Integration Designer does not cache the locateService result, so editing is required.

Reducing the number of Service Component Architecture modules when appropriate

When BPM V7.5 components are assembled into modules for deployment, many factors come into play. Performance is a key factor, but you must consider maintainability, version requirements, and module ownership as well. In addition, more modules can allow for better distribution across servers and nodes. It is important to recognize that modularization also has a cost. When components are placed together in a single server instance, it is best to package them within a single module for best performance.

Using synchronous Service Component Architecture bindings across local modules

For cross-module invocations, where the modules are likely to be deployed locally (that is, within the same server JVM), we suggest using the synchronous SCA binding because this binding is optimized for module locality and outperforms other bindings. Synchronous SCA is as expensive as other bindings when invocations are made between modules in different BPM V7.5 server instances.

Using multi-threaded Service Component Architecture clients to achieve concurrency

Synchronous components that are started locally (that is, from a caller in the same server JVM) run in the context of the thread of the caller. Thus the caller must provide concurrency, if desired, in the form of multiple threads.

Adding quality of service qualifiers at appropriate level

You can add quality of service (QoS) qualifiers, such as Business Object Instance Validation, at the interface level or at an operation level within an interface. Because additional processor usage is associated with QoS qualifiers, do not apply a qualifier at the interface level if it is not needed for all operations of the interface.

3.2.3 Business Process Execution Language business process considerations

This section presents considerations for authoring performance specific to BPEL business processes.

Modeling best practices for activities in a business process

For best performance, use the following guidelines when modeling a BPEL business process:

- ▶ Use the **Audit Logging property for Business Processes only** setting if you need to log events in the Business Process Choreographer database (BPEDB). You can set this property at the activity or process level. If you set it at the process level, the setting is inherited by all activities.
- ▶ For long-running processes, in Integration Designer, click **Properties** → **Server**, and clear the **Enable persistence and queries of business-relevant data** property for both the process and for each individual BPEL activity. Enabling this flag causes details of the execution of this activity to be stored in the BPC database, which increases the load on the database and the amount of data stored for each process instance. Use this setting only if you must retrieve specific information later.
- ▶ For long-running processes, a setting of **Participates** on all activities generally provides the best throughput performance. See “Avoiding two-way synchronous invocation of an asynchronous target” on page 38 for more details.
- ▶ Human tasks can be specified in business processes (for example, process administrators), start activities, and receive activities. Specify human tasks only if needed. When multiple users are involved, use group work items (people assignment criterion: Group) instead of individual work items for group members (people assignment criterion: Group Members).

Avoiding two-way synchronous invocation of long-running processes

When designing long-running business process components, ensure that callers of a two-way (request/response) interface do not use synchronous semantics because this function ties up the caller's resources (such as threads and transactions) until the process completes. Instead, start such processes either asynchronously, or through a one-way synchronous call, where no response is expected. In addition, calling a two-way interface of a long-running business process synchronously introduces difficulties when exceptions occur. Suppose that a non-interruptible process calls a long-running process using the two-way request/response semantics, and the server fails after the long-running process completes, but before the caller's transaction is committed. The following results occur:

- ▶ If the caller was started by a persistent message, upon server restart, the caller's transaction is rolled back and tried again. However, the result of the execution of the long-running process on the server is not rolled back because it was committed before the server failure. As a result, the long-running process on the server runs twice. This duplication causes functional problems in the application unless corrected manually.
- ▶ If the caller was not started by a persistent message, and the response of the long-running process was not submitted yet, the process ends in the failed event queue.

Minimizing the number and size of Business Process Execution Language variables and business objects

Follow these guidelines when defining BPEL variables and BOs:

- ▶ Use as few variables as possible and minimize the size and the number of BOs used. In long-running processes, each commit saves modified variables to the database (to save context), and multiple variables or large BOs make this process costly. Smaller BOs are also more efficient to process when emitting monitor events.
- ▶ Specify variables as data type variables. This specification improves runtime performance.
- ▶ Use transformations (maps or assigns) to produce smaller BOs by only mapping fields necessary for the business logic.

3.2.4 Human task considerations

Follow these guidelines when developing human tasks for BPEL business processes:

- ▶ Use group work items for large groups (people assignment criterion: Group) instead of individual work items for group members (people assignment criterion: Group Members).
- ▶ Use native properties on the task object rather than custom properties where possible. For example, use the **priority** field instead of creating a custom property priority.
- ▶ Set the transactional behavior to `commit` after if the task is not part of a page flow. This setting improves the response time of task complete API calls.

3.2.5 Business process and human tasks client considerations

The following list details considerations for developing effective BPEL business process and human task clients:

- ▶ Do not frequently call APIs that provide task details and process details, such as `htm.getTask()`. Use these methods only when required to show the task details of a single task, for example.
- ▶ Do not put too much work into a single client transaction.
 - In servlet applications, a global transaction is typically not available. If the servlet calls the Human Task Manager (HTM) and Business Flow Manager (BFM) APIs directly, transaction size is typically not a concern.
 - In Enterprise JavaBeans (EJB) applications, make sure that transactions are not too time-consuming. Long-running transactions create long-lasting locks in the database, which prevent other applications and clients to continue processing.
- ▶ Choose the protocol that best suits your needs.
 - In a J2EE environment, use the HTM and BFM EJB APIs. If the client application is running on a BPM V7.5 server, use the local EJB interface.
 - In a Web 2.0 application, use the REST API.
 - In an application that runs remote to the process container, the Web services API is an option.

Clients that follow a page-flow pattern should consider the following issues:

- ▶ Use the `completeAndClaimSuccessor()` API if possible. This API use provides optimal response time.

- ▶ Applications that assign the next available task to the user can use the `claim(String queryTableName, ...)` method on the Human Task Manager EJB interface. This method implements a performance-optimized mechanism to handle claim collisions.
- ▶ Do not put asynchronous invocations between two steps of a page flow because the response time of asynchronous services increases as the load on the system increases.
- ▶ Where possible, do not start long-running subprocesses between two steps of a page flow because long-running subprocesses are started using asynchronous messaging.

Clients that present task lists and process lists should consider the following factors:

- ▶ Use query tables for task list and process list queries.
- ▶ Do not loop over the tasks shown in the task or process list and run an additional remote call for each object. This practice prevents the application from providing good response times and good scalability.
- ▶ Design the application so that all information is retrieved from a single query table during task list and process list retrieval. For example, do not make calls to retrieve the input message for task list or process list creation.

3.2.6 Transactional considerations

One of the strengths of the BPM V7.5 platform is the precise control it provides for specifying transactional behavior. We suggest that when modeling a process or mediation assembly, the modeler should carefully design their desired transaction boundaries as dictated by the needs of the application because boundaries are expensive in terms of system resources. The objective of this section is to guide the modeler to avoid unnecessary transaction boundaries. The following list details some guiding principles:

- ▶ The throughput of a particular usage scenario is inversely related to the number of transaction boundaries traversed in the scenario, so fewer transactions is faster.
- ▶ In user-driven scenarios, improving response time might require more granular transaction boundaries, even at the cost of throughput.
- ▶ Transactions can span across synchronous invocations but cannot span asynchronous invocations.
- ▶ Avoid synchronous invocation of a two-way asynchronous target. The failure recovery of a caller transaction can be problematic.

Taking advantage of Service Component Architecture transaction qualifiers

In an SCA assembly, you can reduce the number of transaction boundaries by allowing transactions to propagate across components. For any pair of components where you want to reduce the number of transaction boundaries, we suggest using the following settings:

- ▶ `SuspendTransaction= false` for the reference of the calling component
- ▶ `joinTransaction= true` for the interface of the called component
- ▶ `Transaction= any|global` for implementation of both components

These settings assume that the first component in such a chain either starts or participates in a global transaction.

Avoiding two-way synchronous invocation of an asynchronous target

If the target component must be started asynchronously and its interface is of two-way request/response style, then the target cannot be safely started through synchronous SCA calls. After the caller sends the request to the target, it waits for response from the target. Upon receiving the request, the asynchronous target starts a new transaction. Upon processing the request, the target returns the response asynchronously to the caller through the response queue. If a system failure occurs after the caller successfully sends the request but before receiving the response, the caller transaction is rolled back and tried again. As a result, the target is started a second time.

Taking advantage of transactional attributes for activities in long-running processes

Although SCA qualifiers control component-level transactional behavior, additional transactional considerations in long-running business processes can cause activities to run in multiple transactions. You can change the scope of those transactions and the number of transactions using the transactional behavior settings on Java Snippet, Human Task, and start activities. For a detailed description of these settings, see “Transactional behavior of business processes” in the WebSphere Process Server for Multiplatforms, Version 7.0 Information Center:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp?topic=/com.ibm.websphere.bpc.doc/doc/bpc/cprocess_transaction.html

You have four setting choices for transactional behavior:

- ▶ Commit before
- ▶ Commit after
- ▶ Participates
- ▶ Requires own

Use the `Commit before` setting in parallel activities that start new branches to ensure parallelism. As noted in the Information Center, you must consider other constraints.

Use `Commit after` for inline human tasks to increase responsiveness to human users. When a human user issues a task completion, the thread/transaction handling that action, is used to resume navigation of the human task activity in the process flow. The user’s task completion action does not complete until the process engine commits the transaction.

Starting with WebSphere Process Server (WPS) 6.2.0, Receive and Pick activities in BPEL flow are allowed to define their own transactional behavior property values. If not set, the default value of initiating a Receive or Pick activity is `Commit after`. Consider using `Participates` where possible because that behavior performs better.

If the `Participates` setting is used, the commit is delayed and forces a longer response time for the user. Only the `Participates` setting does not require a new transaction boundary. The other three settings require the process flow container to start a new transaction before executing the activity, after executing the activity, or both.

In general, the `Participates` attribute provides the best throughput and should be used where possible. This suggestion is true for both synchronous and asynchronous activities. In the two-way asynchronous case, it is important to understand that the calling transaction always commits after sending the request. The `Participates` setting refers to the transaction started by the process engine for the response. When set, this setting allows the next activity to continue on the same transaction.

In special cases, the transaction settings other than `Participates` might be preferable. See “Transactional behavior of business processes” in the WebSphere Process Server for Multiplatforms, Version 7.0 Information Center for details, as indicated in “Taking advantage of transactional attributes for activities in long-running processes” on page 38.

Requires own requires that a new transaction be started.

3.2.7 Invocation style considerations

This section explains invocation style considerations.

Using asynchronicity judiciously

Components and modules might be wired to each other either synchronously or asynchronously. The choice of interaction style can have a profound impact on performance. Exercise care when making this choice.

Setting the Preferred Interaction Style to Synchronous when possible

Many BPM V7.5 server component types (such as interface maps or business rules) start their target components based on the **Preferred Interaction Style** setting of the target interface. Because synchronous cross-component invocations perform better, we suggest setting the **Preferred Interaction Style** to Synchronous when possible. Change this setting to Asynchronous only in specific cases. Such cases might include starting a long-running business process, or more generally, where the target component requires asynchronous invocation.

Starting with WebSphere Integration Developer V6.2 (now Integration Designer), when a new component is added to an assembly diagram, its **Preferred Interaction Style** is set to Synchronous, Asynchronous, or Any, based on the component. In previous releases of the Integration Designer (then called WebSphere Integration Developer), the default initial setting of **Preferred Interaction Style** was set to Any unless explicitly changed by the user. If the **Preferred Interaction Style** of a component is set to Any, how the component is started is determined by the caller's context. If the caller is a long-running business process, a **Preferred Interaction Style** setting of Any is treated as asynchronous. If the caller is a non-interruptible business flow, a **Preferred Interaction Style** setting of Any is treated as synchronous.

See “Taking advantage of transactional attributes for activities in long-running processes” on page 38 for more information about the invocation logic of processes

The following list details additional considerations for invocation styles:

- ▶ When the **Preferred Interaction Style** of an interface is set to Asynchronous, it is important to realize the downstream implications. Any components started downstream inherit the asynchronous interaction style unless they explicitly set the **Preferred Interaction Style** to Synchronous.
- ▶ At the input boundary to a module, exports that represent asynchronous transports such as IBM WebSphere MQ, JMS, or Java EE Connector Architecture (with asynchronous delivery set), set the interaction style to Asynchronous. This setting can cause downstream invocations to be asynchronous if the **Preferred Interaction Style** is Any.

- ▶ For an SCA import, you can use **Preferred Interaction Style** to specify whether the cross-module call should be Synchronous or Asynchronous.
- ▶ For other imports that represent asynchronous transports such as WebSphere MQ or JMS, it is not necessary to set the **Preferred Interaction Style** to Asynchronous. Doing so introduces an unnecessary asynchronous hop between the calling module and the invocation of the transport.

Minimizing cross-component asynchronous invocations within a module

Asynchronous invocations are intended to provide a rich set of qualities of service, including transactions, persistence, and recoverability. Hence, think of an asynchronous invocation as a full messaging hop to its target. When the intended target of the invocation is in the same module, a synchronous invocation yields better performance.

Some qualities of services (such as event sequencing and store-and-forward) can only be associated with asynchronous SCA calls. Consider the performance impact of asynchronous invocations when setting these qualities of service.

Avoiding asynchronous invocation of synchronous services in a FanOut/FanIn block

Do not select asynchronous (deferred response interaction) service invocations for services with synchronous bindings (for example, Web services) unless there is an overriding need and the non-performance implications for this style of invocation are understood.

Apart from the performance implications of calling a synchronous service asynchronously, you must consider reliability and transactional aspects. Generally, use asynchronous callouts only for idempotent query type services. If you need to guarantee that the service is only called once, do not use asynchronous invocation. It is beyond the scope of this paper to provide complete guidance on the functional applicability of using asynchronous callouts in your mediation flow.

More information can be found in the Integration Designer help documentation and BPM V7.5 Information Center:

<http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r5mx/index.jsp>

Assuming that asynchronous callouts are functionally applicable for your configuration, there might be a performance reason for starting a service in this style. However, understand that asynchronous processing is inherently more expensive in terms of processor cycles due to the additional messaging resources incurred by calling a service this way.

Additional operational considerations in choosing synchronous or asynchronous mode might apply. For example, asynchronous invocations use the service integration bus messaging infrastructure, which in turn uses a database for persistence. Synchronous invocations perform well with basic tuning of the JVM heap size and thread pools, but for asynchronous invocations, SCA artifacts require review and tuning. This requirement includes tuning the SCA messaging engine (see 4.3.7, “Messaging engine properties” on page 54), data sources (see 4.3.6, “Java Database Connectivity data source parameters” on page 53), and the database itself. For the data source, the tuning for JMS bindings in this paper can provide guidance as the considerations are the same.

If multiple synchronous services with large latencies are called, asynchronous invocations can reduce the overall response time of the mediation flow, but at the expense of increasing the internal response time of each individual service call. This reduction of overall response time and increase of internal response time assume that asynchronous callouts are configured with parallel waiting in the FanOut section of the flow under the following conditions:

- ▶ With iteration of an array when configuring the FanOut to check for asynchronous responses after all/N messages are fired
- ▶ With extra wires/FlowOrder primitive (by default)

If a number of services in a FanOut section of a mediation flow exists, calling these services synchronously results in an overall response time equal to the sum of the individual service response times.

Calling the services asynchronously (with parallel waiting configured) results in a response time of at least the largest individual service response time in WebSphere ESB. The response time also includes the sum of the time taken by WebSphere ESB to process the remaining service callout responses on the messaging engine queue.

For a FanOut/FanIn block, the processing time for any primitives before or after the service invocations must be added in both cases.

To optimize the overall response time when calling services asynchronously in a FanOut/FanIn section of a mediation flow, start the services in the order of expected latency (highest latency first), if known.

You must consider the tradeoff between parallelism and additional asynchronous processing. The suitability of asynchronous processing depends on the size of the messages being processed, the latency of the target services, the number of services being started, and any response time requirements expressed in service level agreements (SLAs). We suggest running performance evaluations on mediation flows, including FanOuts with high latency services, if you are considering asynchronous invocations.

The default quality of service-on-service references is `Assured Persistent`. You can gain a substantial reduction in asynchronous processing time by changing this setting to `Best Effort` (non-persistent). This change eliminates I/O to the persistence store, but the application must tolerate the possibility of lost request or response messages. This level of reliability for the service integration bus can discard messages under load and might require tuning.

3.2.8 Large object considerations

This section presents best practices for performance (particularly memory use) when using large objects. Very large objects put significant pressure on Java heap use, particularly for 32-bit JVMs. The 64-bit JVMs are less susceptible to this issue due to the large heap sizes that you can be configure, although throughput, response time, and overall system performance can still be impacted by excessive memory usage. Follow these best practices to reduce the memory pressure and avoid `OutOfMemory` exceptions and improve overall system performance.

Avoiding lazy cleanup of resources

Lazy cleanup of resources adds to the liveset required when processing large objects. If you can clean up any resources (for example, by dropping object references when no longer required), do so as soon as is practical.

Avoiding tracing when processing large business objects

Tracing and logging can add significant memory resources. A typical tracing activity is to dump the BO payload. Creating a string representation of a large BO can trigger allocation of many large and small Java objects in the Java heap. For this reason, avoid turning on tracing when processing large BO payloads in production environments.

Also, avoid constructing trace messages outside of conditional guard statement. For example, the sample code in Example 3-1 creates a large string object even if tracing is disabled:

Example 3-1 Creating a large string object

```
String boTrace = bo.toString();
```

Although this pattern is always inefficient, it impacts performance even more if the BO size is large. To avoid unnecessarily creating a BO when tracing is disabled, move the String construction inside an if statement, as shown here:

```
if (tracing_on) System.out.println(bo.toString());
```

Avoiding buffer-doubling code

Study the memory implications of using Java data structures that expand their capacity based on input (for example, StringBuffer and ByteArrayOutputStream). Such data structures usually double their capacity when they run out of space. This doubling can produce significant memory pressure when processing large objects. If possible, always assign an initial size to such data structures.

3.2.9 Mediation flow considerations

This section describes mediation flow considerations for performance.

Using Extensible Stylesheet Language Transformation primitives versus business object maps

The XSLT and the Business Object Map primitives offer two alternate transformation approaches in a mediation flow. If no XSLT-specific function is required, then it is generally better to use the Business Object Map primitive, which can be faster. The exception is when a mediation flow component is trivial in nature and the transformation is taking place at the /body level of the service message object (SMO). In this case, XSLT is faster as the native XML bytes can be passed straight to the XSLT engine. Native XML bytes are available if the XSLT transform primitive is the first in the flow or only preceded by one or more of the following primitives:

- ▶ Route on Message Header (Message Filter primitive)
- ▶ XSLT primitive (Transforming on /body as the root)
- ▶ EndpointLookup without Xpath user properties.
- ▶ Event emitter (event header only)

Aggregation blocks (FanOut / FanIn)

Aggregation is an important WebSphere Enterprise Service Bus (ESB) pattern. It enables a single inbound request to map into multiple outbound service invocations, the responses from which can be aggregated into a single response to the original request. Before you develop mediations using aggregation design patterns, you need to consider several performance factors. See the developerWorks topic on best practices, “Aggregation design patterns and performance considerations in WebSphere Enterprise Service Bus V7.5” at this site:

http://www.ibm.com/developerworks/websphere/library/techarticles/1111_norris/1111_norris.html

Configuring WebSphere ESB resources

When creating resources using Integration Designer, the application developer might choose to use pre-configured WebSphere ESB resources or let the tool generate the mediation flow-related resources that it requires. Both approaches have their advantages and disadvantages.

Pre-configured resources help in the following circumstances:

- ▶ Existing resources are to be used
- ▶ External creation and tuning scripts are to be applied

Pre-configured resources also allow easier post-deployment adjustment.

Tooling-created resources are suitable if no further need exists for creating resources using scripts or the administrative console. Most performance tuning options can be changed as they are now exposed in the tooling.

In our performance tests, we used pre-configured resources because segregating the performance tuning from the business logic makes it possible to maintain the configuration for different scenarios in a single script.

The only cases that did not follow this pattern in our tests were those cases using generic JMS bindings. In such scenarios, resources have already been configured by the third-party JMS provider software (WebSphere MQ V6.0.2.2 for all instances in this paper). The tooling-created resources were used to locate the externally defined resources.

3.3 WebSphere InterChange Server migration considerations

The following considerations pertain to those migrating from WebSphere InterChange Server (WICS) to BPM V7.5:

- ▶ Migrated workloads using custom IBM WebSphere Business Integration (WBI) adapters or older WBI adapters result in interaction with BPM V7.5 through JMS, which is slower than the JCA adapters. Use WebSphere adapters to replace WBI adapters when possible.
- ▶ Some technology adapters (such as HTTP and Web services) are migrated by the WebSphere InterChange Server migration wizard into native BPM V7.5 SCA binding, which performs better. For adapters that are not migrated automatically to available SCA bindings, development effort spent on migrating manually to an SCA binding removes the dependency on an older adapter and provides better performance.
- ▶ The WebSphere InterChange Server Migration wizard in Integration Designer offers a feature to merge the connector and collaboration module. Enable this option, if possible, as it increases performance by reducing cross-module SCA calls.

- ▶ WebSphere InterChange Server collaborations are migrated into BPM V7.5 BPEL processes. You can further customize the resultant BPEL processes and make them more efficient.
 - Migrated BPEL processes enable support for compensation by default. If the migrated workload does not use compensation, you can disable this support to gain performance. The relevant flag can be found in Integration Designer by selecting the process name and then clicking **Properties** → **Details** → **Require a compensation sphere context to be passed in**.
 - The generated BPEL flows still use the WICS APIs to perform BO and collaboration-level tasks. Development effort spent cleaning up the migrated BPEL to use BPM APIs instead of the ICS APIs results in better performance and better maintainability.
 - You might be able to replace BPEL processes produced by migration with other artifacts. All WebSphere InterChange Server collaborations are currently migrated into BPEL processes. For certain scenarios, other BPM V7.5 server artifacts (for example, business rules) might be better choices. Investigate the BPEL processes produced by migration to ensure that the processes are the best fit for your scenario.
- ▶ Disable Message Logger calls in migration-generated mediation flow components (MFCs). The WebSphere InterChange Server Migration wizard in Integration Designer V7.5 generates an MFC to deal with the mapping details of a connector. This MFC contains the code for handling synchronous and asynchronous calls to maps that transform application-specific BOs to generic BOs and generic BOs to application-specific objects. The generated MFC contains embedded MessageLogger calls that log the message to a database. Disable these calls if they are not required in your business scenario to reduce writes to the database and thus improve performance. (Select the MessageLogger instance, choose the Details panel, and clear the **Enabled** check box.)
- ▶ Reduce memory pressure by splitting the shared library generated by the migration wizard. The migration wizard creates a single shared library and puts all migrated BOs, maps, and relationships into it. All the migrated modules share this library by copy. This sharing can cause memory bloat for cases where the shared library is large and many modules are present. The solution is to manually refactor the shared library into multiple libraries based on functionality or usage and modify modules to reference only the shared libraries that they need.
- ▶ If the original WebSphere InterChange Server maps contain many custom map steps, the development effort spent in rewriting those map steps results in better performance. The WebSphere InterChange Server Migration wizard in Integration Designer V7.5 generates maps that use the ICS APIs at a translation layer above BPM V7.5 server technologies. Removing this layer by making direct use of BPM V7.5 server APIs avoids the cost of translation and produces better performance.

3.4 Authoring environment considerations

This section describes suggestions for improving the performance of activities commonly encountered by application developers during the development of an enterprise application, primarily the import, build, and publish operations of an application workspace. The focus of this section contrasts that of previous sections, which concentrated on using the authoring tools to produce high-performance runtime applications.

Using hardware advantages

Importing and building an enterprise application is a resource-intensive activity. Recent improvements in desktop hardware architecture greatly improve the responsiveness of import and build activities. Also, for I/O intensive activities (such as Import), a faster disk drive or disk subsystem reduces total response time.

Sharing Business Integration libraries across modules

BPM V7.5 includes the IBM Process Center, which makes it possible for Business Integration modules to become part of a BPM Process Application, a new feature in this version. One of the benefits of a Process Application is the ability to scope sharing of Business Integration libraries among the various modules in a Process Application instead of being duplicated for each module. This benefit saves server memory for large libraries that were previously module scoped because it eliminates duplicate copies. Lazy BO parsing mode is required to use the Process Application scope.

Although the Process Application scope is the default for new libraries in Process Applications for BPM V7.5, existing libraries retain the module scope. Whenever possible, change these libraries to use the Process Application scope. Memory savings are correlated to the number of modules that reference the library. The greater the number of modules that use the library, the greater the potential memory savings for using Process Application scoped libraries.

Placing libraries in Process Application Scope to save memory

The Process Center makes it possible for Business Integration modules to become part of a BPM Process Application, a new feature for the Integration Designer in this version. One of the benefits of a Process Application is the ability to scope sharing of Business Integration libraries among the various modules in a Process Application instead of being duplicated for each module. This benefit saves server memory for large libraries that were previously module scoped because it eliminates duplicate copies. In BPM V7.5, lazy BO parsing mode is required to use the Process Application scope.

Although the Process Application scope is the default for new libraries in Process Applications of BPM V7.5, existing libraries retain the module scope. Whenever possible, change these libraries to use the Process Application scope. Memory savings are correlated to the number of modules that reference the library. The greater the number of modules that use the library, the greater the potential memory savings for using Process Application scoped libraries.

3.5 Business Space developer considerations

While developing a Business Space solution, developers often alter the configuration of the browser to perform debugging, tracing, and other browser functions. However, some of these configuration changes can dramatically affect the performance of the Business Space solution. Before rolling out a solution to production or taking performance measurements, review this section and observe its guidelines.

Disabling or uninstalling add-ins or extensions for production use

An important consideration is using add-ins or extensions for both Internet Explorer (IE) and Firefox. For IE, enabling Developer Tools dramatically decreases performance. To some extent, Firebug and HttpWatch do as well, but not to the same degree. If you experience problems, always make sure you disable or uninstall all add-ins or extensions unless they are necessary.

Avoiding Developer Tools render or compatibility mode in Internet Explorer for production

In IE, it is possible to place the browser in compatibility mode or in a render mode equivalent to an older browser version by using the Developer Tools. This option can be useful for developing or debugging solutions (for example, making an IE 8 browser behave as an IE 7 browser for compatibility testing). However, make sure that you do not use these settings because more recent versions of IE typically perform better than older versions.

Using browser tools

The following observations pertain to browser tools that are available for obtaining response time measurements, counting requests, and analyzing caching.

- ▶ The **Net** tab in Firebug is useful for obtaining request timings, analyzing request/response headers, and counting the number of requests. However, it reports requests that are satisfied from the browser cache as code 200 responses. You can still determine that the request is cached from the **Cache** tab shown on the request, which indicates that the request was satisfied from the browser cache. If you copy and paste the results into another document (for example, into an email), the **Cache** tab is not copied. Thus, it is possible to be misled by the 200 responses and draw an incorrect conclusion that caching is not effective when it actually is.
- ▶ Fiddler is another powerful tool and has the advantage of supporting both IE and FireFox browsers. However, because Fiddler acts as a proxy between the browser and the server and cached requests are handled internally by browsers, these requests never shown in Fiddler. This absence of result reporting prevents you from determining which requests are fulfilled from the browser cache, but Fiddler is still useful for analyzing the requests that actually are sent to the server.
- ▶ HttpWatch does not have the limitations of Fiddler because it is supported on both IE and FireFox browsers. Its results copy and paste easily into either a spreadsheet or a document, and it displays cached requests in a straightforward manner.



Performance tuning and configuration

To optimize performance, it is necessary to configure a system differently than the default settings. This chapter lists several areas to consider during system tuning, including Business Process Manager (BPM) products and other products in the system, like DB2. Documentation for each of these products can answer questions about performance, capacity planning, and configuration and offers guidance about achieving high performance in various operational environments.

A number of configuration parameters are available to the system administrator. Although this chapter identifies several specific parameters observed to affect performance, it does not address all available parameters. For a complete list of configuration parameters and possible settings, see the relevant product documentation.

The first section of this chapter offers a methodology for tuning a deployed system. Following this section is a basic tuning checklist that enumerates the major components and their associated tuning concepts. Subsections address tuning in more detail, describing several tuning parameters and their suggested settings (where appropriate) and providing advanced tuning guidelines for key areas of the system. Representative values for many of the tuning parameters are shown in Chapter 5, “Initial configuration settings” on page 95.

Important: Although there is no guarantee that following the guidance in this chapter provides acceptable performance, but if you set these parameters incorrectly, you can expect degraded performance.

“Related publications” on page 105 contains references to related documentation that might prove valuable when tuning a particular configuration.

4.1 Performance tuning methodology

We suggest a system-wide approach to performance tuning the BPM environment. System performance tuning, which requires training and experience, is not exhaustively described here. Rather, we highlight some key aspects of tuning that are important.

Tuning encompasses every element of the deployment topology:

- ▶ Physical hardware topology choices
- ▶ Operating system parameters
- ▶ BPM server, WebSphere Application Server, and messaging engine settings

The methodology for tuning can be stated as an iterative loop:

1. Pick a set of reasonable initial parameter settings and run the system.
2. Monitor the system to obtain metrics that indicate whether performance is being limited.
3. Use monitoring data to guide further tuning changes.
4. Repeat until done.

The following subsections describe these steps.

4.1.1 Picking a set of reasonable initial parameter settings

Use the tuning checklist in 4.2, “Tuning checklist” on page 49 for a systematic way to set parameters.

For specific initial values, see Chapter 5, “Initial configuration settings” on page 95 for settings for the various workloads used in internal performance evaluation. You might consider these values for initial values.

4.1.2 Monitoring the system

We suggest monitoring the system components to determine system health and the need for further tuning as follows:

- ▶ For each physical machine in the topology, including front-end and back-end servers such as web and database servers, monitor the following processes using the relevant OS tools (such as `vmstat`, `iostat`, `netstat`, or their equivalents):
 - Processor core use
 - Memory use
 - Disk use
 - Network use
- ▶ Complete the following actions for each Java virtual machine (JVM) process started on a physical machine (for example, process server, messaging engine server, and other components):
 - Use tools such as `ps` or their equivalents to get core and memory usage per process.
 - Collect verbose garbage collection (`verbosegc`) statistics to obtain information about Java memory usage.
- ▶ For each BPM server or messaging engine JVM, use IBM Tivoli® Performance Viewer to monitor these types of data:
 - Data connection pool use for each data source.
 - Thread pool use for each thread pool (Web container, default, and Work Managers).

4.1.3 Using monitoring data to guide further tuning changes

Correctly using monitoring data to determine how to tune the BPM requires skill and experience. In general, this phase of tuning requires the analyst to look at the collected monitoring data, detect performance bottlenecks, and do further tuning. The key characteristic of this phase of tuning is that it is driven by the monitoring data collected in the previous phase.

Performance bottlenecks might include, but are not limited to, these situations:

- ▶ Excessive use of physical resources, such as processor cores, disk, and memory. These issues can be resolved either by adding more physical resources or rebalancing the load across the available resources.
- ▶ Excessive use of virtual resources. Examples of these resources include heap memory, connection pools, thread pools, and other resources. In this case, use tuning parameters remove the bottlenecks.

4.2 Tuning checklist

This checklist serves as a guide when tuning a BPM solution. Each of these topics is covered in more detail in the remainder of this chapter.

- ▶ Common tunables
 - Use a 64-bit JVM for all servers.
 - Disable tracing and monitoring when possible.
 - Ensure that all databases are on a high-performance database management system (DBMS) such as DB2, and that the databases are well-tuned.
 - If security is required, use Application security, not Java2 security.
 - Use an appropriate hardware configuration for performance measurement (for example, notebooks and desktops are not appropriate for realistic performance evaluations).
 - If hardware virtualization is used, ensure that adequate processor, memory, and I/O resources are allocated to each virtual machine. Avoid overcommitting resources.
 - Do not run production servers in development mode or with a development profile.
 - Tune external service providers and external interfaces to ensure that they do not cause a system bottleneck.
 - Configure message-driven bean (MDB) activation specifications.
 - Configure for clustering (where applicable).
 - Configure thread pool sizes.
 - Configure settings of data sources for connection pool size and prepared statement cache size. Consider using non-XA data sources for Common Event Infrastructure data when that data is non-critical.
 - Increase the maximum number of connections in the data pool to greater than or equal to the sum of all maximum thread pool sizes.

- ▶ Business Process Choreographer (for BPC business processes)
 - Use Work Manager-based navigation for long running processes, and optimize the message pool size and intertransaction cache size.
 - Use query tables to optimize query response time.
 - Optimize Business Flow Manager resources:
 - Database connection (Business Process Choreographer database)
 - Activation specification (**BPEInternalActivationSpec**)
 - Java Message Service (JMS) data source connection (BPECF and BPECFD)
 - Optimize the database configuration for the Business Process Choreographer database (BPEDB).
 - Optimize indexes for SQL statements that result from task and process list queries using database tools such as the DB2 design advisor.
 - Turn off state observers that are not needed (for example, audit logging).
- ▶ Business Processing Modeling Notation (BPMN) business processes
 - Increase log file size for twproc database (to 16,384 pages).
 - Enable file system caching for twproc database:


```
db2 alter tablespace userspace1 file system caching
```
 - Exclude the table SIBOWNER from automatic runstats execution.
 - Ensure that database statistics are up to date.
 - Create new indexes as described in the *Saved search performance degradation with WebSphere Lombardi Edition* technote:

<http://www-01.ibm.com/support/docview.wss?uid=swg21474536>
- ▶ Messaging and message bindings
 - Optimize activation specification (JMS, MQJMS, WebSphere MQ).
 - Optimize queue connection factory (JMS, MQJMS, WebSphere MQ).
 - Configure connection pool size (JMS, MQJMS, WebSphere MQ).
 - Configure service integration bus data buffer sizes.
- ▶ Database
 - Place database table spaces and logs on a fast disk subsystem.
 - Place logs on a separate device from the table space containers.
 - Maintain current indexes on tables.
 - Update database statistics.
 - Set log file sizes correctly.
 - Optimize buffer pool size (DB2) or buffer cache size (Oracle).
- ▶ Java
 - Set the heap and nursery sizes to manage memory efficiently.
 - Choose the appropriate garbage collection policy (generally, **-Xgcpolicy:gencon**)
- ▶ Business Monitor
 - Configure Common Event Infrastructure.
 - Set message consumption batch size.
 - Enable key performance indicator (KPI) caching.
 - Use table-based event delivery.
 - Enable the data movement service.

4.3 Common tuning parameters

This section lists performance tuning parameters commonly used for tuning BPM V7.5 solutions, for both BPMN and Business Process Execution Language (BPEL) business processes.

4.3.1 Tracing and logging flags

Tracing and logging are often necessary when setting up a system or debugging issues. However, these capabilities require performance resources that are often significant. Minimize their use when evaluating performance or in production environments. This section lists tracing parameters used in the products covered in this paper. Some settings are common to all or a subset of the products, while others are specific to a particular product. Unless stated otherwise, you can set all of these parameters using the administrative console.

To enable or disable tracing, Click **Troubleshooting** → **Logs and Trace** in the properties of the subscription. Select the server on which you want to change the log detail levels and click **Change Log Detail Levels**. Set both the **Configuration** and **Runtime** fields to * =all=disabled.

To change the Performance Monitoring Infrastructure (PMI) level, click **Monitoring and Tuning** → **Performance Monitoring Infrastructure**. Select the server on which you want to change the log detail levels and click **none**.

In addition, Cross-Component Tracing (XCT) is useful for problem determination, enabling correlation of Service Component Architecture (SCA) component information with log entries. However, do not use XCT in production or while obtaining performance data. Two levels of XCT settings are possible:

- ▶ Enable
- ▶ Enable with data snapshot

Both incur significant performance resource usage. Enable with data snapshot is costly because of the additional I/O involved in saving snapshots in files.

To enable or disable XCT, click **Troubleshooting** → **Cross-Component Trace**. Select the **XCT** setting from three options under the **Configuration or Runtime** tab:

- ▶ Enable
- ▶ Disable
- ▶ Enable with data snapshot

Changes made on the **Runtime** tab take effect immediately. Changes made on the **Configuration** tab require a server restart to take effect.

Further information is provided in the *Managing Log Level Settings in TeamWorks* technote:

<http://www-01.ibm.com/support/docview.wss?uid=swg21439659>

4.3.2 Java tuning parameters

In this section, we list a few frequently used Java Virtual Machine (JVM) tuning parameters. For a complete list, see the JVM tuning guide offered by your JVM supplier.

To change the JVM parameters, go to the JVM administrative window by first clicking **Servers** → **Application** → **Performance Monitoring Infrastructure**. Select the server on which you want to change the JVM tuning parameters. Then click **Server Infrastructure** → **Java and Process Management** → **Process Definition** → **Additional Properties** → **Java Virtual Machine**. Change the JVM parameters on this panel.

Java garbage collection policy

The default garbage collection (GC) algorithm on platforms with an IBM JVM is a generational concurrent collector (specified with `-Xgcpolicy:gencon` under the Generic JVM arguments on the JVM administrative console panel). Our internal evaluation shows that this garbage collection policy usually delivers better performance with a tuned nursery size, as described in the next section.

Java heap sizes

To change the default Java heap sizes, set the initial heap size and maximum heap size explicitly on the JVM window in the administrative console. Note that 64-bit JVMs (the suggested mode for BPM V7.5 servers) support much larger heap sizes than 32-bit JVMs. Use this capability to relieve memory pressure in the Java heap, but always ensure that there is sufficient physical memory to back the JVM heap size and all other memory requirements.

If you click **Generational Concurrent Garbage Collector**, the Java heap is divided into a new area (*nursery*), where new objects are allocated, and an old area (*tenured space*), where longer-lived objects are located. The total heap size is the sum of the new area and the tenured space. You can set the new area size independently from the total heap size. Typically, set the new area size between 1/4 and 1/2 of the total heap size. The relevant parameters are:

- ▶ `Xmns<size>`: Initial new area size
- ▶ `Xmnx<size>`: Maximum new area size
- ▶ `Xmn<size>`: Fixed new area size

4.3.3 Message-driven bean ActivationSpec

There is more than one shortcut you can take in the administrative console to access the **MDB ActivationSpec** tuning parameters:

- ▶ Click **Resources** → **Resource Adapters** → **J2C activation specifications**. Select the name of the application specification you want to access.
- ▶ Click **Resources** → **Resource Adapters** → **Resource adapters**. Select the name of the resource adapter you want to access. Then click **Additional properties** → **J2C activation specifications**. Select the name of the activation specification you want.

The following custom properties in the message-driven bean (MDB) ActivationSpec have considerable performance implications. These properties are described further in “Tuning message-driven bean ActivationSpec properties” on page 57.

- ▶ **maxConcurrency**
- ▶ **maxBatchSize**

4.3.4 Thread pool sizes

BPM servers use thread pools to manage concurrent tasks. You can set the Maximum Size property of a thread pool in the administrative console by clicking **Servers** → **Application servers** and selecting the server name whose thread pool you want to manage. Click **Additional Properties** → **Thread Pools** and then the thread pool name.

You typically must tune the following thread pools:

- ▶ Default
- ▶ ORB.thread.pool
- ▶ WebContainer

In addition, thread pools used by Work Managers for BPEL processes are configured separately in the console by clicking **Resources** → **Asynchronous beans** → **Work managers**. Select the Work Manager name and then click **Thread pool properties**

You typically must tune the following Work Managers:

- ▶ DefaultWorkManager
- ▶ BPENavigationWorkManager

4.3.5 Java Message Service connection pool sizes

You can access the JMS connection factories and JMS queue connection factories from the administrative console in several ways:

- ▶ Click **Resources** → **Resource Adapters** → **J2C connection factories** and select the factory name.
- ▶ Click **Resources** → **JMS** → **Connection factories** and select the factory name.
- ▶ Click **Resources** → **JMS** → **Queue connection factories** and select the factory name.
- ▶ Click **Resources** → **Resource Adapters** → **Resource adapters** and select the resource adapter name (for example, SIB JMS Resource Adapter). Then click **Additional priorities** → **J2C connection factories** and select the factory name.
- ▶ From the connection factory admin panel, click **Additional Properties** → **Connection pool properties**. Set the **Maximum connections** property for the maximum size of the connection pool.

4.3.6 Java Database Connectivity data source parameters

Data sources can be accessed through either of these paths:

- ▶ Click **Resources** → **JDBC** → **Data sources** and select the data source name
- ▶ Click **Resources** → **JDBC Providers** and select the Java Database Connectivity (JDBC) provider name, followed by clicking **Additional Properties** → **Data sources** and selecting the data source name

Connection pool size

The maximum size of the data source connection pool is limited by the value of the Maximum connections property, which can be configured by clicking the **Additional Properties** → **Connection pool properties** in the data source window.

Increase the maximum number of connections in the data pool to greater than or equal to the sum of all maximum thread pool sizes.

The following data sources typically need to be tuned:

- ▶ Business Process Choreographer (BPC) data sources for the BPEDB and associated message engine database (for BPEL business processes)
- ▶ BPMN data sources for the BPMN databases and associated message engine databases (for BPMN business processes)
- ▶ SCA application bus messaging engine data source
- ▶ SCA system bus messaging engine data source
- ▶ Common Event Infrastructure (CEI) bus messaging engine data source

Prepared statement cache size

You can configure the cache size of the data source prepared statement from the data source. Click **Additional properties** → **WebSphere Application Server data source properties**.

For BPM servers, tune the BPEDB data source to a higher value. Start by setting this value to 300.

4.3.7 Messaging engine properties

Two messaging engine custom properties might affect the messaging engine performance:

- ▶ `sib.msgstore.discardableDataBufferSize`
 - The property stays in the data buffer for best effort nonpersistent messages.
 - The default value is 320 K.
 - After the buffer is full, messages are discarded to allow newer messages to be written to the buffer.
- ▶ `sib.msgstore.cachedDataBufferSizeCachedDataBufferSize`
 - The property stays in memory cache for messages other than best effort nonpersistent.
 - The default is 320 K.

These properties can be accessed in the console by first clicking **Service Integration** → **Buses** and selecting the bus name. Then, click **Messaging Engines** and select the messaging engine name. Finally, click **Additional properties** → **Custom properties**.

4.3.8 Running production servers in production mode

BPM servers can be run in development mode to reduce startup time for the server by using JVM settings to disable bytecode verification and reduce Just-In-Time (JIT) compilation time. Do not use this setting on production servers because it is not designed to produce optimal runtime performance. Make sure the **Run in development mode** check box for the server is cleared. This setting is found in the configuration window of the server in the administrative console. Click **Servers** → **Application Servers**. Click the server whose setting you want to change and click **Configuration**.

You might also create server profiles with production or development templates. Use production profile templates for production servers.

4.4 Advanced tuning

This section contains advanced tuning tips.

4.4.1 Tracing and monitoring considerations

The ability to configure tracing and monitoring at different levels for various system components is valuable during periods of system analysis or debugging. The BPM product set provides rich monitoring capabilities, both in terms of business monitoring through the CEI and audit logging, and system performance monitoring through the PMI and the Application Response Measurement (ARM) infrastructure. Although these capabilities provide insight into the performance of the running solution, these features can degrade overall system performance and throughput.

Tracing and monitoring effect on performance: Use tracing and monitoring judiciously. When possible, turn them off to ensure optimal performance.

Most tracing and monitoring are controlled through the administrative console. Validate that the appropriate level of tracing and monitoring is set for the PMI Monitoring, Logging, and Tracing settings through the administrative console.

Use the administrative console to validate that the **Audit logging** and **Common Event Infrastructure logging** check boxes are cleared in the Business Flow Manager and the Human Task Manager, unless these capabilities are required for business reasons.

Integration Designer is also used to control event monitoring. Check the **Event Monitor** tab for your components and business processes to ensure that event monitoring is applied judiciously.

4.4.2 Tuning for large objects

In this section, we describe tuning for performance when using large objects.

Increasing the Java heap size to its maximum

One of the key factors affecting large object processing is the maximum size of the Java heap. In this section, we describe how to set the heap size as large as possible on two commonly used platforms, Windows and AIX.

- ▶ Windows (32-bit)

Because of address space limitations in the Windows 32-bit operating system, the largest heap that can be obtained is 1.4 GB to 1.6 GB for 32-bit JVMs.

- ▶ AIX (32-bit)

On AIX 32-bit systems, the Java 5 and Java 6 JVM typically supports heaps in the range of 2 GB to 2.4 GB. Because the 4 GB address space allowed by the 32-bit system is shared with other resources, the actual limit of the heap size depends on the memory used by these resources. These resources include thread stacks, JIT compiled code, loaded classes, shared libraries, and buffers used by OS system services. A large heap squeezes address space reserved for other resources and might cause runtime failures.

Setting maximum heap sizes applies only to 32-bit JVMs; when using 64-bit JVMs, the heap size is only constrained by the amount of physical memory available. The suggested configuration for BPM V7.5 servers is 64-bit.

For more comprehensive heap setting techniques, see 4.13, “Advanced Java heap tuning” on page 89.

Reducing or eliminating other processing while processing a large object

One way to allow for larger object sizes is to limit the concurrent processing within the JVM. Do not expect to process a steady stream of the largest objects possible concurrently with other BPM server and WebSphere Adapters activities. The operational assumption when considering large objects is that not all objects are large or very large and that large objects do not arrive often, perhaps only once or twice per day. If more than one very large object is being processed concurrently, the likelihood of failure increases dramatically.

The size and number of the normally arriving smaller objects affect the amount of Java heap memory consumption in the system. Generally speaking, the heavier the load on a system when a large object is being processed, the more likely you encounter memory problems.

For adapters, the amount of concurrent processing can be influenced by setting the **pollPeriod** and **pollQuantity** parameters. To allow for larger object sizes, set a relatively high value for **pollPeriod** (for example, 10 seconds) and low value for **pollQuantity** (for example, 1 second) to minimize the amount of concurrent processing that occurs. These settings are not optimal for peak throughput, so if an adapter instance must support both high throughput for smaller objects interspersed with occasional large objects, you must make trade-offs.

4.4.3 Tuning for maximum concurrency

For most high-volume deployments on server-class hardware, many operations present themselves to take place simultaneously. Tuning for maximum concurrency ensures that the server accepts enough load to saturate its core. One sign of an inadequately tuned configuration is when additional load does not result in additional core use while the cores are not fully used. To optimize these operations for maximum concurrency, the general guideline is to follow the execution flow and remove bottlenecks one at a time.

Higher concurrent processing means higher resource requirements (memory and number of threads) on the server. High concurrent processing must be balanced with other tuning objectives, such as the handling of large objects, handling large numbers of users, and providing good response time.

Tuning edging components for concurrency

The first step in tuning edging components for concurrency is to ensure that business objects are handled concurrently at the edge components of BPM solutions. If the input business objects come from the adapter, ensure that the adapter is tuned for concurrent delivery of input messages.

If the input business objects come from WebServices export binding or direct invocation from Java Server Pages (JSPs) or servlets, make sure the WebContainer thread pool is sized right. For example, to allow for 100 in-flight requests to be handled concurrently by a BPM server, the maximum size of the WebContainer thread pool must be set to 100 or larger.

If the input business objects come from messaging, you must tune the ActivationSpec (MDB bindings) and Listener ports (WebSphere MQ or MQJMS bindings).

Tuning message-driven bean ActivationSpec properties

For each JMS export component, there is an MDB and its corresponding ActivationSpec in the Java Naming and Directory Interface (JNDI name `module name/export component name_AS`). The default value for `maxConcurrency` of the JMS export MDB is 10, meaning up to 10 business objects from the JMS queue can be delivered to the MDB threads concurrently. Change it to 100 if a concurrency of 100 is wanted.

The Tivoli Performance Viewer can be used to monitor the `maxConcurrency` parameter. For each message being processed by an MDB, there is a message on the queue marked as being locked inside a transaction (which is removed after the `onMessage` completes). These messages are classed as unavailable. The PMI metric `UnavailableMessageCount` gives you the number of unavailable messages on each queue point. Check this value by selecting the resource name and then clicking **SIB Service** → **SIB Messaging Engines**. Select the bus name and click **Destinations** → **Queues**.

If any queue has `maxConcurrency` or more unavailable messages, this condition implies that the number of messages on the queue is currently running above the concurrency maximum of the MDB. If this situation occurs, increase the `maxConcurrency` setting for that MDB.

The maximum batch size in the activation specification also has an impact on performance. The default value is 1. The maximum batch size value determines how many messages are taken from the messaging layer and delivered to the application layer in a single step. This batch size value does not mean that this work is done within a single transaction, and thus this setting does not influence transactional scope. Increase this value (for example, to 8) for activation specs associated with SCA modules and long-running business processes to improve performance and scalability, especially for large multi-core systems.

Configuring thread pool sizes

The sizes of thread pools have a direct impact on the ability of a server to run applications concurrently. For maximum concurrency, you must set the thread pool sizes to optimal values. Increasing the **maxConcurrency** or **Maximum sessions** parameters only enables the concurrent delivery of business objects from the JMS or WebSphere MQ queues. For a BPM server to process multiple requests concurrently, you must increase the corresponding thread pool sizes to allow higher concurrent execution of these MDB threads.

MDB work is dispatched to threads allocated from the default thread pool. All MDBs in the application server share this thread pool unless a different thread pool is specified. This condition means that the default thread pool size needs to be larger, probably significantly larger, than the **maxConcurrency** of any individual MDB.

Threads in the WebContainer thread pool are used for handling incoming HTTP and Web services requests. This thread pool is shared by all applications deployed on the server, and you must tune it, likely to a higher value than the default.

Object request broker (ORB) thread pool threads are employed for running ORB requests (for example, remote EJB calls). The thread pool size needs to be large enough to handle requests coming through the interface, such as certain human task manager APIs.

Configuring dedicated thread pools for message-driven beans

The default thread pool is shared by many WebSphere Application Server tasks. It is sometimes desirable to separate the execution of JMS MDBs to a dedicated thread pool. Complete the following steps to change the thread pool used for JMS MDB threads:

1. Create a thread pool (for example, MDBThreadPool) on the server by clicking **Servers** → **Server Types** → **WebSphere application servers** → **server** → **Thread pools**. Then click **New**.
2. Open the service integration bus (SIB) JMS Resource Adapter administrative console with server scope by clicking **Resources** → **Resource Adapters** → **Resource adapters**. If the adapter is not visible, go to **Preferences**, and select the **Show built-in resources** check box.
3. Change the thread pool alias from **Default** to **MDBThreadPool**.
4. Repeat steps 2 and 3 for SIB JMS Resource Adapters at the node and cell scope.
5. Restart the server for the changes to become effective.

SCA Module MDBs for asynchronous SCA calls use a separate resource adapter, the Platform Messaging Component SPI Resource Adapter. Follow the same steps to change the thread pool to a different one if you want.

Even with a dedicated thread pool, all MDBs associated with the resource adapter still share a thread pool. However, they do not have to compete with other WebSphere Application Server tasks that also use the Default thread pool.

Configuring Java Message Service and Java Message Service queue connection factories

Multiple concurrently running threads might bottleneck on resources such as JMS and database connection pools if such resources are not tuned properly. **Maximum Connections pool size** specifies the maximum number of physical connections that can be created in this pool. These physical connections interface with back-end resources (for example, a DB2 database). After the thread pool limit is reached, the requester cannot create new physical connections and must wait until a physical connection that is currently in use is returned to the pool, or a **ConnectionWaitTimeout** exception is issued.

For example, if you set the **Maximum Connections** value to 5, and there are five physical connections in use, the pool manager waits for the amount of time specified in **Connection Timeout** for a physical connection to become free. The threads waiting for connections to the underlying resource are blocked until the connections are freed up and allocated to them by the pool manager. If no connection is freed in the specified interval, a `ConnectionWaitTimeout` exception is issued.

If you set **Maximum Connections** to 0, the connection pool is allowed to grow infinitely. This setting has the side effect of causing the **Connection Timeout** value to be ignored.

The general guideline for tuning connection factories is that their maximum connection pool size needs to match the number of concurrent threads multiplied by the number of simultaneous connections per thread.

For each JMS, WebSphere MQ, or MQJMS Import, a connection factory exists that was created during application deployment. Make the **Maximum Connections** property of the connection pool associated with the JMS connection factory large enough to provide connections for all threads concurrently running in the import component. For example, if 100 threads are expected to run in a given module, set the **Maximum Connections** property to 100. The default is 10.

From the connection factory configuration panel, click **Additional Properties** → **Connection pool properties**. Set the **Maximum Connections** property to the maximum size of the connection pool.

Configuring data source options

Make the **Maximum Connections** property of data sources large enough to allow concurrent access to the databases from all threads. Typically, a number of data sources are configured in BPM servers (for example, the BPEDB data source, the TWPROC data sources, the TWPERFDB data sources, the WPSDB data source, and the messaging engine database data sources). Set the **Maximum Connections** property of each data source to match the maximum concurrency of other system resources as described in 4.4.3, “Tuning for maximum concurrency” on page 57.

Setting data source prepared statement cache size

The BPC container uses prepared statements extensively. Set the statement cache sizes to large enough to avoid repeatedly preparing statements for accessing the databases.

Set the prepared statement cache for the BPEDB to at least 300.

4.4.4 Messaging tuning

This section describes performance tuning for messaging.

Choosing data store or file store for messaging engines

Messaging engine persistence is backed by a database. However, a stand-alone BPM configuration might have the persistence storage of BPE and SCA buses backed by the file system (file store). You must choose a file store at profile creation time. Use the Profile Management Tool to create a new stand-alone enterprise service bus profile or stand-alone process server profile.

To do so, click **Profile Creation Options** → **Advanced profile creation** → **Database Configuration**, and select the **Use a file store for Messaging Engine (MEs)** check box. When this profile is used, file stores are used for BPE and SCA service integration buses.

Setting data buffer sizes (discardable or cached)

The **DiscardableDataBufferSize** is the size in bytes of the data buffer used when processing best-effort non-persistent messages. The purpose of the discardable data buffer is to hold message data in memory because this data is never written to the data store for this quality of service. Messages that are too large to fit into this buffer are discarded.

The **CachedDataBufferSize** is the size in bytes of the data buffer used when processing all messages other than best-effort non-persistent messages. The purpose of the cached data buffer is to optimize performance by caching in memory data that might otherwise need to be read from the data store.

You can set the **DiscardableDataBufferSize** and **CachedDataBufferSize** in the administrative console by clicking **Service Integration-Buses** and selecting the bus name. Click **Messaging Engines** and select the messaging engine name. Click **Additional properties** → **Custom properties** and enter the values for **DiscardableDataBufferSize** and **CachedDataBufferSize**.

Using a high-performance database management system for messaging engine data stores

For better performance, use production-quality databases, such as DB2, for the messaging engine data stores. You can choose the database at profile creation time using the advanced profile creation option.

Creating the DB2 database and loading the data store schema

Instead of having one DB2 database per messaging engine, we put all messaging engines into the same database, using different schema to separate them, as shown in Table 4-1.

Table 4-1 Messaging engine schemas

Schema	Messaging engine
SCASYS	box01-server1.SCA.SYSTEM.box01Node01Cell.Bus
SCAAPP	box01-server1.SCA.APPLICATION.box01Node01Cell.Bus
CEIMSG	box01-server1.CommonEventInfrastructure_Bus
BPCMSG	box01-server1.BPC.box01Node01Cell.Bus

Follow these steps to place all messaging engines into the same database:

1. Create one schema definition for each messaging engine with the following command:

```
WAS Install\bin\sibDDLGenerator.bat -system db2 -version 8.1 -platform windows  
-statementend ; -schema BPCMSG -user user >createSIBSchema_BPCMSG.dd1
```

In this syntax (used on a Windows operating system), **WAS Install** represents the BPM Installation directory and **user** represents the user name.

2. Repeat the command for each schema or messaging engine.

3. To distribute the database across several disks, edit the created schema definitions, and put each table in a table space named after the schema used. For example, SCAAPP becomes SCANODE_TS, CEIMSG becomes CEIMSG_TS, and so on. After editing, the schema definition should look like Example 4-1.

Example 4-1 Schema definition

```
CREATE SCHEMA CEIMSG;
CREATE TABLE CEIMSG.SIBOWNER (
  ME_UUID VARCHAR(16),
  INC_UUID VARCHAR(16),
  VERSION INTEGER,
  MIGRATION_VERSION INTEGER
) IN CEIMSG_TB;
CREATE TABLE CEIMSG.SIBCLASSMAP (
  CLASSID INTEGER NOT NULL,
  URI VARCHAR(2048) NOT NULL,
  PRIMARY KEY(CLASSID)
) IN CEIMSG_TB;
...

```

It is possible to separate table spaces for the various tables here. Optimal distribution depends on application structure and load characteristics. In this example, one table space per data store is used.

4. After creating all schema definitions and defined table spaces for the tables, create a database named SIB.
5. Create the table spaces and distribute the containers across available disks by issuing the following command for a system managed table space:

```
DB2 CREATE TABLESPACE CEIMSG_TB MANAGED BY SYSTEM USING( '<path>\CEIMSG_TB' )
```

Place the database log on a separate disk, if possible.

6. Create the schema of the database by loading the four schema definitions into the database.

For more information about database and DB2-specific tuning, see the following sections:

- ▶ 4.10, “General database tuning” on page 78
- ▶ 4.11, “DB2-specific database tuning” on page 80

Creating the data sources for the messaging engines

Create a data source for each messaging engine and configure each messaging engine to use the new data store using the administrative console.

To do so, follow these steps:

1. Create a JDBC provider of type DB2 Universal JDBC Driver Provider for the non-XA data sources if it does not exist. The XA DB2 JDBC Driver Provider should exist if BPC was configured correctly for DB2.
2. Create four new JDBC data sources, one for CEI as an XA data source and the remaining three as single-phase commit (non-XA) data sources.

Table 4-2 provides new names for the data sources.

Table 4-2 New data sources

Name of data source	JNDI Name	Type of JDBC provider
CEIMSG_sib	jdbc/sib/CEIMSG	DB2 Universal (XA)
SCAAPP_sib	jdbc/sib/SCAAPPLICATION	DB2 Universal
SCASYSTEM_sib	jdbc/sib/SCASYSTEM	DB2 Universal
BPCMSG_sib	jdbc/sib/BPCMSG	DB2 Universal

When creating a data source, complete the following tasks.

1. Clear the **Use this Data Source in container managed persistence (CMP)** check box.
2. Set a component-managed authentication alias.
3. Set the database name to the name used for the database created earlier for messaging (for example, Service Integration Bus).
4. Select a driver type of type 2 or type 4. It is suggested that DB2 use the JDBC Universal Driver type 2 connectivity to access local databases and type 4 connectivity to access remote databases. A type 4 driver requires a host name and valid port to be configured for the database.

Changing the data stores of the messaging engines

Use the administrative console to change the data stores of the messaging engines:

1. In the Navigation panel, click **Service Integration** → **Buses** and change the data stores for each bus/messaging engine that is displayed.
2. Put in the new JNDI and schema name for each data store. Clear the **Create Tables** check box because the tables have already been created.

The server immediately restarts the messaging engine. The `SystemOut.log` file shows the results of the change and indicates whether the messaging engine starts successfully.

3. Restart the server and validate that all systems come up using the updated configuration.

The last remaining task is tuning the database. For more information about database and DB2-specific tuning, see the following sections:

- ▶ 4.10, “General database tuning” on page 78
- ▶ 4.11, “DB2-specific database tuning” on page 80

4.4.5 Clustered topology tuning

One reason for deploying a clustered topology is to add more resources to system components that are bottlenecked due to increasing load. Ideally, you can scale up a topology arbitrarily to match the required load. The BPM Network Deployment (ND) infrastructure provides this capability. However, effective scaling still requires standard performance monitoring and bottleneck analysis techniques.

The following list details some considerations and tuning guidelines for when you expand or tune a clustered topology. In the description, we assume that additional cluster members also imply additional server hardware.

- ▶ If deploying more than one cluster member (JVM) on a single physical system, it is important to monitor the resource use (core, disk, network, and other components) of the system as a whole. Also monitor the resources used by each cluster member. Monitoring makes it possible for you to detect a system bottleneck due to a particular cluster member.
- ▶ If all members of a cluster are bottlenecked, you can scale by adding one or more members to the cluster, backed by appropriate physical hardware.
- ▶ If a singleton server or cluster member is the bottleneck, you must consider additional factors:
 - A messaging engine in a cluster with a “One of N” policy (to preserve event ordering) might become the bottleneck. The following scaling options are available to fix this issue:
 - Host the active cluster member on a more powerful hardware server or remove extraneous load from the existing server.
 - If the messaging engine cluster is servicing multiple buses, and messaging traffic is spread across these buses, consider employing a separate messaging engine cluster per bus.
 - If a particular bus is a bottleneck, consider whether destinations on that bus can tolerate out-of-order events. In this case, the cluster policy can be changed to allow workload balancing with partitioned destinations. Partitioning a bus also includes considerations for balancing work across the messaging engine cluster members.
 - A database server might become the bottleneck. Consider the following approaches to fix this issue:
 - If the database server is hosting multiple databases that are active (for example, BPEDB, twproc, twperfdb, and MEDB), consider hosting each database on a separate server.
 - If a single database is driving load, consider a more powerful database server.
 - Beyond these items, you can use database partitioning and clustering capabilities.

4.4.6 Web services tuning

If the target of the Web services import binding is hosted locally in the same application server, you can further improve the performance by taking advantage of the optimized communication path provided by the Web container. Normally, requests from the Web services clients are sent through the network connection between the client and the service provider. For local Web services calls, however, WebSphere Application Server offers a direct communication channel bypassing the network layer completely. Complete the following steps to enable this optimization. Use the administrative console to change these values.

1. Click **Application servers** and then the name of the desired server. Click **Container Settings** → **Web Container Settings** → **Web container** → **Additional Properties** → **Custom Properties**. Set the Web container custom property `enableInProcessConnections` to `true`.

Do not use wildcards (*) for the host name of the Web container port. Replace it with the host name or IP address. The property can be accessed by clicking the following path:

Application servers → **messaging engine** → **Container Settings** → **Web Container Settings** → **Web container** → **Additional Properties** → **Web container transport chains** → **WCInboundDefault** → **TCP inbound channel (TCP_2)** → **Related Items** → **Ports** → **WC_defaulthost** → **Host**.

2. Use the `localhost` instead of host name in the Web services client binding. Using the actual host name (even if it is aliased to `localhost`), disables this optimization. The host name can be accessed by first clicking **Enterprise Applications** and selecting the application name. Next, click **Manage Modules** and select the application EJB jar. Finally, click **Web services client bindings** → **Preferred port mappings** and select the binding name. Use the `localhost` (for example, `localhost:9080`) in the URL.
3. Make sure that there is not an entry for the server host name and IP address hosts file of your server for name resolution. An entry in the hosts file inhibits this optimization by adding name resolution processor usage.

4.4.7 Power management tuning

Power management is becoming common in processor technology. Both Intel and Power core processors now have this capability. This capability delivers obvious benefits, but it can also decrease system performance when a system is under high load, so consider whether to enable power management. For example, with IBM POWER6® hardware, ensure that Power Saver Mode is not enabled unless wanted. One way to modify or check that this setting on AIX is through the Power Management window on the Hardware Management Console (HMC).

4.4.8 Setting AIX threading parameters

The IBM JVM threading and synchronization components are based upon the AIX POSIX-compliant threads implementation. The following environment variable settings improve Java performance in many situations. The variables control the mapping of Java threads to AIX Native threads, turn off mapping information, and allow for spinning on mutually exclusive (mutex) locks:

- ▶ `export AIXTHREAD_COND_DEBUG=OFF`
- ▶ `export AIXTHREAD_MUTEX_DEBUG=OFF`
- ▶ `export AIXTHREAD_RWLOCK_DEBUG=OFF`
- ▶ `export AIXTHREAD_SCOPE=S`
- ▶ `export SPINLOOPTIME=2000`

4.5 Business Process Choreographer tuning for Business Process Execution Language business processes

This section provides advanced tuning tips for Business Process Choreographer.

4.5.1 Tuning Work Manager-based navigation for business processes

Work Manager-based navigation is the default navigation mode for BPEL business processes (as opposed to JMS-based navigation). Work Manager-based navigation provides two methods to optimize performance, keeping the quality of service of (JMS-based) process navigation consistent with persistent messaging:

- ▶ **Work Manager-based navigation**

WorkManager is a thread pool of J2EE threads. WorkManager process navigation takes advantage of an underlying capability of WebSphere Application Server to start the processing of ready-to-navigate business flow activities without using messaging as provided by JMS providers.

- ▶ **InterTransactionCache**

This cache is a part of the Work Manager-based navigation mode that holds process instance state information in memory, reducing the need to retrieve information from the BPE database.

Several parameters control usage of these two optimizations. We find the first set of these parameters in the administrative console by clicking **Application Servers** and selecting the server name you want. Then, click **Business Integration** → **Business Process Choreographer** → **Business Flow Manager** → **Business Process Navigation Performance**.

Enabling this capability at the cluster level overrides the settings for a specific server. In other words, in a clustered environment, it is easiest to enable this capability at the cluster level.

Key parameters are as follows:

- ▶ **Enable advanced performance optimization**

Select this property to enable both the Work Manager-based navigation and InterTransactionCache optimizations.

- ▶ **Work-Manager-Based Navigation Message Pool Size**

This property specifies the size of the cache used for navigation messages that cannot be processed immediately, provided Work Manager-based navigation is enabled. The cache defaults to the message size (computed by $10 * \text{thread pool size of the BPENavigationWorkManager}$). If this cache reaches its limit, JMS-based navigation is used for new messages. For optimal performance, ensure that this Message Pool size is set to a sufficiently high value.

- ▶ **InterTransaction Cache Size**

This property specifies the size of the cache used to store process state information that has also been written to the BPE database. Set this value to twice the number of parallel running process instances. The default value for this property is the thread pool size of the BPENavigationWorkManager.

In addition, you can customize the number of threads for the Work Manager using these settings by clicking **Resources** → **Asynchronous Beans** → **Work Managers** → **BPENavigationWorkManager**.

Increase the minimum number of threads from its default value of 5, and increase the maximum number of threads from its default value of 12, using the methodology outlined in 4.4.3, “Tuning for maximum concurrency” on page 57. If the thread pool size is modified, also modify the work request queue size and set to be twice the maximum number of threads.

4.5.2 Tuning the business process container for Java Message Service navigation

If JMS-based navigation is configured, you must optimize the following resources for efficient navigation of business processes:

- ▶ Activation specification `BPEInternalActivationSpec`

The **Maximum Concurrent Endpoints** parameter specifies the parallelism that is used for process navigation across all process instances. Increase the value of this parameter to increase the number of business processes run concurrently. This resource can be found in the administrative console by clicking **Resources** → **Activation Specifications** → **BPEInternalActivationSpec**.

- ▶ JMS connection factory `BPECFC`

Set the connection pool size to the number of threads in the `BPEInternalActivationSpec` plus 10%. This resource can be found in the administrative console by clicking **Resources** → **JMS** → **Connection factories** → **BPECFC** → **Connection pool properties**. This connection factory is also used when Work Manager-based navigation is in use, but only for error situations or if the server is highly overloaded.

4.5.3 Tuning task list and process list queries

The programmer creates Task list and process list queries in BPM applications using the standard query APIs, that is, `query()` and `queryAll()`, and related REST and Web services interfaces. Task list and process list queries are also created by the query table APIs `queryEntities()` and `queryRows()`. All task list and process list queries result in SQL queries against the BPC database. These SQL queries might need special tuning to provide optimal response times, as follows:

- ▶ Up-to-date database statistics are key for good SQL query response times.
- ▶ Databases offer tools to tune SQL queries. In most cases, additional indexes improve query performance with some potential impact on process navigation performance. For DB2, the DB2 design advisor can be used to guide in choosing indexes.

4.5.4 Tuning Business Process Choreographer API calls

BPC API calls are triggered by requests external to the BPM server run time. Examples include remote EJB requests, web service requests, web requests over HTTP, requests that come through the SCA layer, or JMS requests. The connection pools associated with each of these communication mechanisms might need tuning. Consider the following hints when tuning the connection pools:

- ▶ API calls for task list and process list queries might take more time to respond, depending on the tuning of the database and the amount of data in the database.
- ▶ Ensure that concurrency (parallelism) is sufficiently high to handle the load and to use the processor. However, increasing the parallelism of API call execution beyond what is necessary can negatively influence response times. Also, increased parallelism can put excessive load on the BPC database. When tuning the parallelism of API calls, measure response times before and after tuning, and adjust the parallelism if necessary.

4.5.5 Tuning intermediate components for concurrency

If the input business object is handled by a single thread from end to end, the tuning for the edge components is normally adequate. In many situations, however, multiple thread switches exist during the end-to-end execution path. It is important to tune the system to ensure adequate concurrency for each asynchronous segment of the execution path.

Asynchronous invocations of an SCA component use an MDB to listen for incoming events that arrive in the associated input queue. Each SCA module defines an MDB and its corresponding activation specification (JNDI name *sca/module name/ActivationSpec*). The SCA module MDB is shared by all asynchronous SCA components within the module, including SCA export components. Take this shared state into account when configuring the `maxConcurrency` property value of `ActivationSpec`. SCA module MDBs use the same default thread pool as those for JMS exports.

The asynchronicity in a long-running business process occurs at transaction boundaries (see 3.2.6, “Transactional considerations” on page 37 for more details about settings that affect transaction boundaries). BPE defines an internal MDB and its `ActivationSpec` as `BPEInternalActivationSpec`. The `maxConcurrency` parameter must be tuned following the same guideline as for SCA module and JMS export MDBs (as described in 4.4.3, “Tuning for maximum concurrency” on page 57). The only catch is that only one `BPEInternalActivationSpec` exists for a single BPM server.

4.6 Business Processing Modeling Notation business process tuning

In addition to the common tuning guidance, this section provides tuning guidance specific to BPMN business processes. Much of this tuning applies to the Performance Server and SIB databases.

4.6.1 Database tuning

The following list indicates that ways databases can be tuned for better performance.

- ▶ Increase log file size for twproc database (to 16,384 pages)
- ▶ Enable file system caching for twproc database as follows:
db2 alter table space userspace1 file system caching
- ▶ Exclude the table SIBOWNER from automatic runstats execution as described in the following technote:
<http://www-01.ibm.com/support/docview.wss?uid=swg21452323>
- ▶ Ensure that database statistics are up to date.

If database utilization is running at high throughput rates, consider disabling some or all database auto-maintenance tasks to avoid impacting peak throughput. However, if you disable these capabilities, be sure to perform runstats regularly to update database statistics

- ▶ Create new indexes as described in the following technote:

<http://www-01.ibm.com/support/docview.wss?uid=swg21474536>

Create the indexes using the following commands (if using DB2; other databases have similar commands):

```
db2 "CREATE INDEX IDX_SOAB_BPD_INSTANCE ON LSW_BPD_INSTANCE (SNAPSHOT_ID ASC,
BPD_INSTANCE_ID ASC) ALLOW REVERSE SCANS COLLECT SAMPLED DETAILED STATISTICS"
```

```
db2 "CREATE INDEX IDX_SOAB_BPD_INSTANCE_VAR ON LSW_BPD_INSTANCE_VARIABLES
(BPD_INSTANCE_ID ASC, VARIABLE_TYPE ASC, ALIAS ASC) ALLOW REVERSE SCANS COLLECT
SAMPLED DETAILED STATISTICS"
```

4.6.2 Business Process Definition Queue Size and Worker Thread Pool

When running at high throughput rates, you might need to set the **BPD Queue Size** and **Worker Thread Pool** parameters to larger values than their defaults (20 for BPD Queue Size and 40 for Worker Thread Pool). These values are defined in the file `80EventManager.xml` in the configuration directory for the process server. The specific configuration directory is as follows:

```
%BPM%/profiles/<profileName>/config/cells/<cellName>/nodes/<nodeName>/servers/
server1/process-center or process-server/config/system
```

To change the values, directly edit that file. Here are some guidelines for tuning these parameters:

- ▶ As a rule of thumb, start with a BPD Queue Size (**bpd-queue-capacity**) of 10 per physical processor core (for example, 40 for a four-processor core configuration). Tune as needed after that, based on the performance of your system.
- ▶ Ensure the Worker Thread Pool size (**max-thread-pool-size**) is always greater than the sum of all four queue sizes (BPD, Sync, Async, and System).

4.7 WebSphere ESB tuning

Following are additional configuration options that are relevant to tuning WebSphere ESB. See 5.2, “WebSphere ESB settings” on page 102 for a suggested set of initial values to use.

4.7.1 Tuning the database if using persistent messaging

If you are using persistent messaging, the configuration of your database is important. Use a remote DB2 instance with a fast disk array as the database server. You might benefit from tuning the connection pooling and statement cache of the data source.

For more information about tuning DB2, see the following sections:

- ▶ 4.10, “General database tuning” on page 78
- ▶ 4.11, “DB2-specific database tuning” on page 80

Note the relevant references in “Related publications” on page 105.

4.7.2 Disabling event distribution for Common Event Infrastructure

The event server that manages events can be configured to distribute events and log them to the event database. Some mediations only require events to be logged to a database. For these cases, performance improves by disabling event distribution. Because the event server might be used by other applications, it is important to check that none of them use event monitoring, which requires event distribution before disabling event distribution.

Event distribution can be disabled from the administrative console by clicking **Service integration** → **Common Event Infrastructure** → **Event service** → **Event services** → **Default Common Event Infrastructure event server**. Clear the **Enable event distribution** check box.

4.7.3 Configuring WebSphere Service Registry and Repository cache timeout

WebSphere Service Registry and Repository (WSRR) is used by WebSphere ESB for endpoint lookup. When accessing the WSRR (for example, using the endpoint lookup mediation primitive), results from the registry are cached in WebSphere ESB. You can configure the lifetime of the cached entries from the administrative console by clicking **Service Integration** → **WSRR Definitions** and entering the WSRR definition name. Then click **Timeout of Cache** and set a value for the cached registry results.

Validate that the timeout is a sufficiently large value. The default timeout is 300 seconds, which is reasonable from a performance perspective. Too low a value results in frequent lookups to the WSRR, which can be expensive (especially if retrieving a list of results) and includes the associated network latency if the registry is on a remote machine.

4.8 Business Monitor tuning

This section provides advanced tuning suggestions for Business Monitor.

4.8.1 Configuring Java heap sizes

The default maximum heap size in most implementations of Java is too small for many of the servers in this configuration. The Business Monitor Launchpad installs the Business Monitor and its prerequisite servers with larger heap sizes, but you might check that these sizes are appropriate for your hardware and workload.

4.8.2 Configuring Common Event Infrastructure

By default, when an event arrives at CEI, it is delivered to the registered consumer (in this case, a particular Monitor Model) and into an additional default queue. Performance improves by avoiding this double-store by removing the All Events event group using the administrative console. Click **Service Integration** → **Common Event Infrastructure** → **Event Service** → **Event Services** → **Default Common Event Infrastructure event server** → **Event Groups** and remove the event group.

Beyond its persistent delivery of events to registered consumers, CEI offers the ability to explicitly store events in a database. Database event storage requires significant processor usage, so avoid storing the events in the database if this additional functionality is not needed. You can also configure the CEI data store in the administrative console by going to **Service Integration** → **Common Event Infrastructure** → **Event Service** → **Event Services** → **Default Common Event Infrastructure event server**. Clear the **Enable Data Store** check box.

4.8.3 Configuring message consumption batch size

Processing events in large batches is much more efficient than one at a time. Up to some limit, the larger the batch size, the higher the event processing throughput. But there is a trade-off: processing events, processing them, and persisting them to the Monitor database is done as a transaction. Although a larger batch size yields better throughput, it costs more if you must roll back. If you experience frequent rollbacks, consider reducing the batch size. You can reduce cache size in the administrative console under the server scope. Click **Applications** → **Monitor Models** and select the version of the batch. Then click **Runtime Configuration** → **Tuning** → **Message Consumption Batch size** and set the batch size you want. The default value is 100.

4.8.4 Enabling key performance indicator caching

The cost of calculating aggregate key performance indicator (KPI) values increases as completed process instances accumulate in the database. A KPI cache is available to reduce the resource usage of these calculations at the cost of some staleness in the results. The refresh interval is configurable in the WebSphere administrative console. Click **Applications** → **Monitor Models** and select the version. Then, click **Runtime Configuration** → **KPI** → **KPI Cache Refresh Interval**.

A value of zero (the default) disables the cache.

4.8.5 Using table-based event delivery

There are two ways in which events can be delivered by CEI to a monitor model:

- ▶ Through a JMS queue
- ▶ Through a database table

You can choose a method at application install time for a monitor model. It is suggested that you choose table-based event delivery (sometimes called *queue bypass*), both for reliability and for performance and scalability reasons.

4.8.6 Enabling the Data Movement Service

By default, the same tables are used for event processing and for dashboard reporting. You can enable an optional scheduled service, called the *Data Movement Service (DMS)* to switch dashboard reporting against a separate set of tables. DMS also periodically copies the data from the event processing tables for the server to the dashboard reporting tables. This processing and reporting mode optimizes the following components:

- ▶ Indexes for each table
- ▶ Event processing tables for the server for quick insert/update/delete
- ▶ Dashboard reporting tables for common dashboard-related queries

We suggest that you enable DMS in any production scenario.

4.9 Business Space tuning

Because Business Space is an Asynchronous JavaScript and XML (AJAX) application, much of its code is run in the browser through JavaScript. Its performance is therefore highly dependent on the browser and the client system it is running on. To improve response times in Business Space, you might need to adjust server and client settings. This section covers the key tuning parameters for Business Space solutions.

4.9.1 Ensuring browser cache is enabled in Internet Explorer 8

By default, Business Space allows the browser to cache static information like style sheets and JavaScript files for 24 hours. However, if the browser is configured in a way that it does not cache data, the same resources is loaded over and over, again resulting in long page load times. To enable caching, follow these steps for your browser:

- ▶ Internet Explorer
 - a. Go to **Tools** → **Internet Options** → **General** → **Browsing History**
 - b. Set the cache size, indicated by **Disk space to use**, to at least 50 MB.
- ▶ Firefox
 - a. Go to **Tools** → **Options** → **Advanced** → **Network** → **Offline Storage**
 - b. Set the offline storage to at least 50 MB.

Ensuring browser cache is not cleared on logout

Modern browsers have options to clear the cache on exit; make sure that this setting is not enabled. Instructions for IE8 and Firefox follow:

Use the following procedure to disable cache clearing for IE8:

1. Click **Tools** → **Internet Options**.
2. Under the **General** tab, make sure the **Delete browser history on exit** check box is cleared, as shown in Figure 4-1.

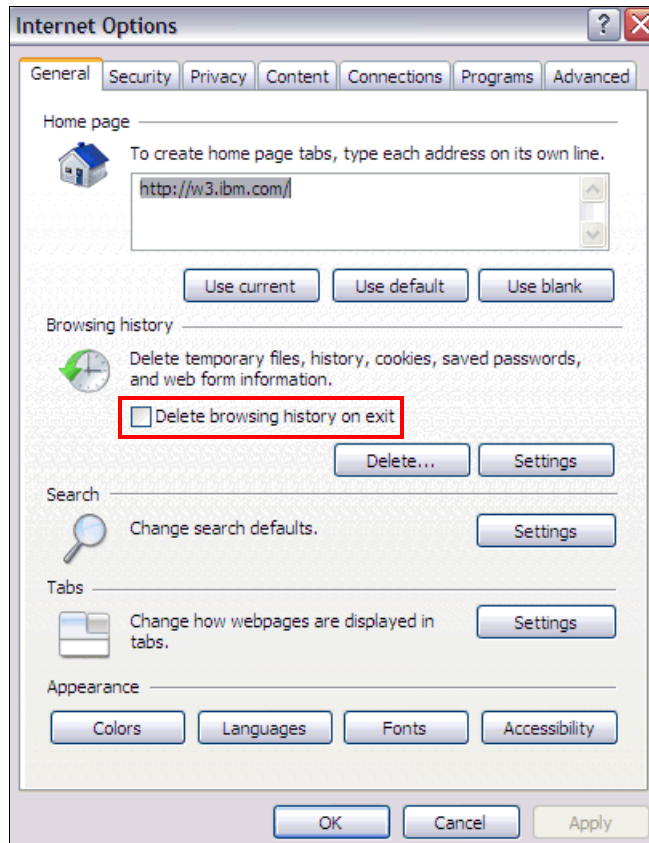


Figure 4-1 Deleting browsing history in Internet Explorer 8

3. Click the **Advanced** tab and make sure the **Empty Temporary Internet Files folder when browser is closed** check box is cleared, as shown in Figure 4-2. Click **OK** to save the settings.

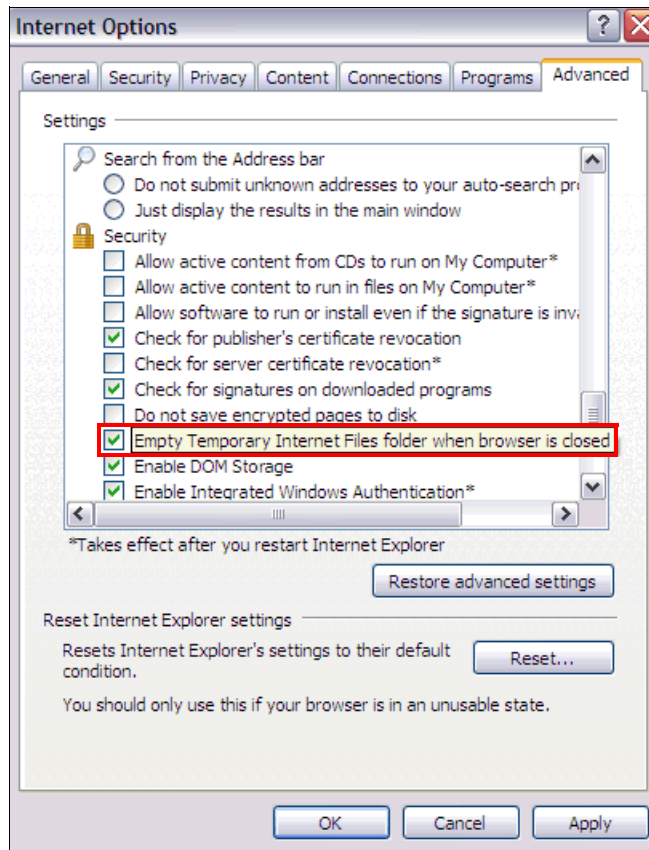


Figure 4-2 Empty Temporary Internet Files folder in Internet Explorer 8

4. Restart your browser to make sure that the changes have taken effect.

4.9.2 Ensuring browser cache is enabled in Firefox 3.6

Use the following procedure to disable cache clearing for Firefox 3.6:

1. Click **Tools** → **Options**.
2. Click the **Privacy** tab and make sure that it reads Firefox will: Remember History, as shown in Figure 4-3. Click **OK**. This setting enables caching.

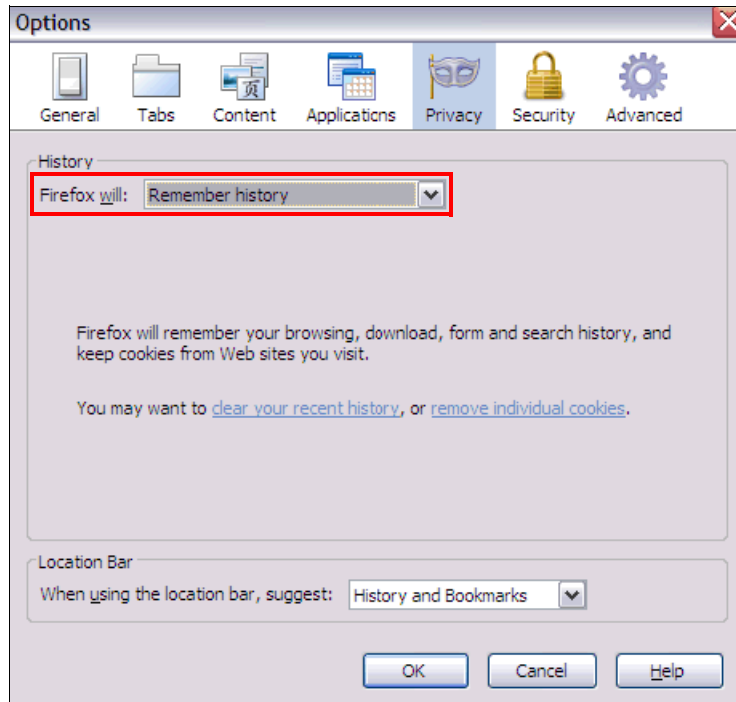


Figure 4-3 Remember history in Firefox 3.6

If you click **Never remember history**, caching is disabled and you must change the setting.

If you click **Use custom settings for history**, you have options that still allow caching, as shown in Figure 4-4.

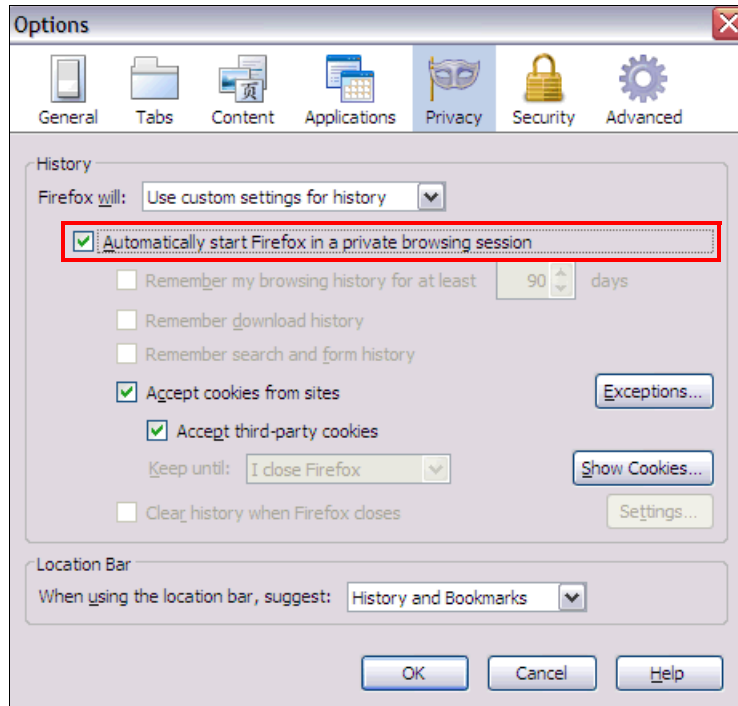


Figure 4-4 Using custom settings for history in Firefox 3.6

These options still allow caching in Firefox:

- If you check **Automatically start Firefox in a private browsing session**, caching is enabled. However, after the browser is closed, everything is erased (not just cache, but also browsing history) as though you were never using the browser.
- If private browsing is not selected, then make sure the **Clear history when Firefox closes** check box is cleared, as shown in Figure 4-5.

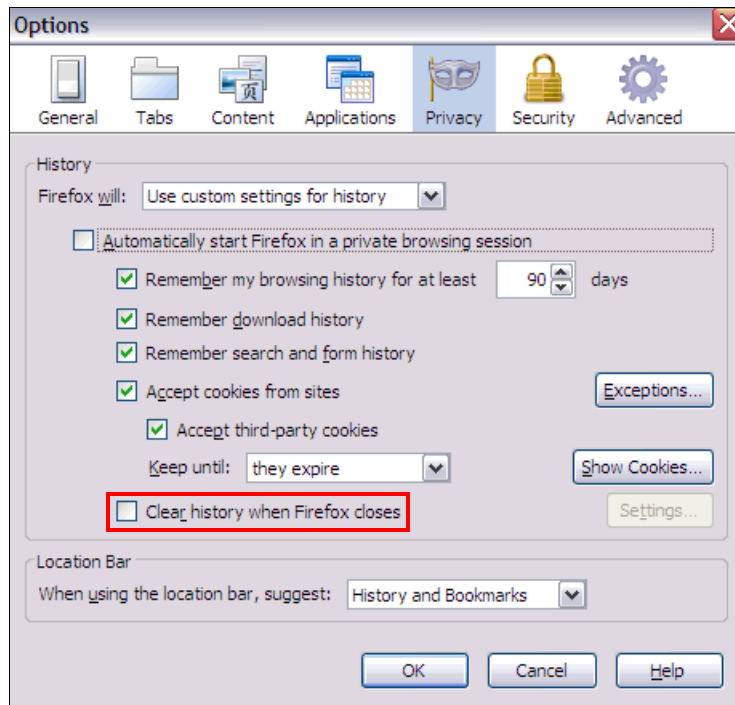


Figure 4-5 Clear history in Firefox 3.6

3. Restart the browser to make sure that changes have taken effect.

4.9.3 Optimizing complex task message performance

When opening tasks with large and complex input messages through the Task Information widget, your browser might take a long time to render and show the task. If you are facing this issue, refer to the following technote:

<http://www-01.ibm.com/support/docview.wss?uid=swg21468931&wv=1>

4.9.4 Tuning the HTTP server

Business Space in production scenarios is generally deployed using a topology which includes an HTTP server or an HTTP server plug-in. As an AJAX client, Business Space naturally sends multiple requests to the server, which are then handled by this HTTP server. In order to ensure good user response times, tune your HTTP server to efficiently handle the expected load.

For more information about tuning the IBM HTTP Server, which is included as part of BPM V7.5, see the *IBM HTTP Server Performance Tuning* page:

http://publib.boulder.ibm.com/httserv/ihsdiag/ihs_performance.html

For more information about tuning Business Space, see *Scalability and Performance of Business Space and Human Task Management Widgets in WebSphere Process Server v7*:

<http://www-01.ibm.com/support/docview.wss?uid=swg27020684&wv=1>

4.9.5 Optimizing performance when not using federation mode

Those familiar with the performance characteristics of Human Workflow widgets with WebSphere Process Server (WPS) V7.0 might experience an initial performance degradation when using Human Workflow widgets such as Tasks, Processes, and Task Definitions in BPM V7.5. This degradation is only apparent if using Business Space on a new (not a migrated) installation of BPM V7.5. These performance considerations apply if you use the Human Task Management widgets with only Business Process Definition (BPD) processes and human services on a single application server, or on a cluster.

This performance degradation is caused by the addition of a federation service layer and the higher number of items to be retrieved. Addition of the federated service layer is a usability enhancement in BPM V7.5 that enables the display of all tasks in a single unified task list.

WPS performance with BPM upgrade: WPS performance degradation with BPM V7.5 does not apply to an upgrade or migration from WPS V7.

To check the Business Space target services configuration in the administration console, complete the following steps.

1. Open the IBM Business Process Manager Integrated Solutions Console.
2. Click the **Servers** tab in the navigation bar.
3. Click **WebSphere**.
4. Click **Application Servers** server type and choose the application server on which you want to run Business Space.
5. In the Business Integration section on the **Configuration** tab, click the **Business Space Configuration** entry.
6. On the Business Space Configuration page, in the Additional Properties section, click the **REST service endpoint registration** entry.
7. On the REST service endpoint registration page, check the Service Endpoint Target values for the Process services and Task services REST endpoint types. The default initial setting is Federated REST services.

To improve performance by using the BPC REST End Point instead of the Federated REST Endpoint, complete the following steps.

1. Change the Business Space target services configuration in the administration console by following the steps..
2. Change the Process services to the Business Process Choreographer REST services endpoint of the application server that is running your processes and human tasks.
3. Change the Tasks services to the Business Process Choreographer REST services endpoint of the application server that is running your processes and human tasks.
4. Click **OK** to save the configuration changes.
5. Log out of Business Space and then log in again.

Using BDP processes and human services for a single server or cluster: To use only the BPD processes and human services of a single application server or cluster, change the Process services and Task services to the Business process definition engine REST services of the application server that is running your BPD processes and human services.

4.10 General database tuning

This section provides general tuning hints for databases.

4.10.1 Providing adequate statistics for optimization

Databases usually offer a wide variety of available choices when determining the best approach to accessing data. Statistics, which describe the shape of the data, are used to guide the selection of a low-cost data access strategy. Statistics are maintained in tables and indexes. Examples of statistics include the number of rows in a table and the number of distinct values in a certain column.

Gathering statistics can be expensive, but fortunately, for many workloads, a set of representative statistics allows for good performance over a large span of time. It might be necessary to refresh statistics periodically if the data population shifts dramatically.

4.10.2 Placing database log files on a fast disk subsystem

Databases are designed for high availability, transactional processing, and recoverability. For performance reasons, changes made to table data might not be written immediately to disk. These changes can be recovered if they are written to the database log. Updates are made to database log files when the log buffer fills, at transaction commit time, and for some implementations after a maximum interval of time.

As a result, database log files might be heavily used. More importantly, the log writes hold commit operations pending, meaning that the application is synchronously waiting for the write to complete. Therefore, the performance of write access to the database log files is critical to overall system performance. For this reason, we suggest that database log files be placed on a fast disk subsystem with write-back cache.

4.10.3 Placing logs on a separate device from the table space containers

A basic strategy for all database storage configurations is to place the database logs on dedicated physical disks, ideally on a dedicated disk adapter. This placement reduces disk access contention between I/O to the table space containers and I/O to the database logs and preserves the mostly sequential access pattern of the log stream. Such separation also improves recoverability when log archival is employed.

4.10.4 Providing sufficient physical memory

Accessing data in memory is much faster than reading it from disk. Because 64-bit hardware is readily available and memory prices continue to fall, it makes sense to provision enough memory to avoid most disk reads in steady state for many performance-critical workloads.

Take care to avoid virtual memory paging in the database machine. The database manages its memory with the assumption that it is never paged and does not cooperate well if the operating system swaps some of its pages to disk.

4.10.5 Avoiding double buffering

Because the database attempts to keep frequently accessed data in memory, in most cases, using file system caching offers no benefit. However, performance typically improves by using direct I/O, when files read by the database bypass the file system cache and only one copy of the data is held in memory. Using direct I/O makes it possible for the system to allocate more memory to the database and avoids resource usage in the file system as it manages its cache.

A further advantage can be gained on some operating systems, such as AIX, by using concurrent I/O. Using concurrent I/O bypasses per-file locking, shifting responsibility for concurrency control to the database and, in some cases, makes it possible to offer more useful work to the adapter or the device.

An important exception to this guideline occurs for large objects (LOB, BLOB, CLOB, and others) that are not buffered by the database itself. In this case, it can be advantageous to arrange for file system caching, preferably only for files that back large objects.

4.10.6 Refining table indexes as required

BPM products typically provide a reasonable set of indexes for the database tables they use. In general, creating indexes involves a tradeoff between the cost of queries and the cost of statements which insert, update, or delete data. For query-intensive workloads, it makes sense to provide a rich variety of indexes as required to allow rapid access to data. For update-intensive workloads, it is often helpful to minimize the number of indexes defined as each row modification might require changes to multiple indexes. Indexes are kept current even when they are infrequently used.

Index design therefore involves compromises. The default set of indexes might not be optimal for the database traffic generated by a BPM product in a specific situation. If database processor or disk use is high or there are concerns with database response time, it might be helpful to consider changes to the indexes.

DB2 and Oracle databases provide assistance in this area by analyzing indexes in the context of a given workload. These databases offer suggestions to add, modify, or remove indexes. One caveat is that if the workload does not capture all relevant database activity, a necessary index might appear unused, leading to a suggestion that it be dropped. If the index is not present, future database activity can suffer as a result.

4.10.7 Archiving completed process instances

Over time, completed process instances accumulate in the database of the servers. This accumulation can alter the performance characteristics of the solution being measured. It is helpful to archive completed process instances to ensure that database population is controlled. If you do not archive completed process instances and Performance Data Warehouse events, they grow unbounded over time, which impacts overall system performance.

One symptom of this problem is long query times on Business Process Definition (BPD) and TASK tables. Another symptom is your process server database tables occupy too much disk space. Both of these symptoms occur because completed BPD instances are not deleted from the system automatically. After a BPD instance is completed, the instance is typically no longer needed and therefore can be removed from the Process Server database. BPM provides a stored procedure called LSW_BPD_INSTANCE_DELETE, which you can use to delete old instances. Archiving procedures can be found in the following technote:

<http://www-01.ibm.com/support/docview.wss?uid=swg21439859>

4.11 DB2-specific database tuning

Providing a comprehensive DB2 tuning guide is beyond the scope of this paper. However, a few general rules of thumb can assist in improving the performance of DB2 environments. In the following sections, we describe these rules and provide pointers to more detailed information.

The complete set of current DB2 manuals (including database tuning guidelines) can be found in the DB2 Information Center:

<http://publib.boulder.ibm.com/infocenter/db21uw/v9r7/index.jsp>

Another reference is the *Best practices for DB2 for Linux, UNIX, and Windows* page, available at the following location:

<http://www.ibm.com/developerworks/data/bestpractices/db21uw/>

4.11.1 Updating database statistics

DB2 provides an Automatic Table Maintenance feature which runs the **RUNSTATS** command in the background as required. Using **RUNSTATS** ensures that the correct statistics are collected and maintained. This feature is controlled with the database configuration parameter **auto_runstats** and is enabled by default for databases created by DB2 9.1 and beyond. See the Configure Automatic Maintenance wizard at the database level in the DB2 Control Center.

One approach to updating statistics manually on all tables in the database is to use the **REORGCHK** command. Dynamic SQL, such as that produced by Java Database Connectivity (JDBC), immediately takes the new statistics into account. Static SQL, such as that in stored procedures, must be explicitly rebound in the context of the new statistics. Example 4-2 shows an example of DB2 commands that perform the steps to gather basic statistics on database DBNAME.

Example 4-2 Gathering basic statistics on database DBNAME

```
db2 connect to DBNAME
db2 reorgchk update statistics on table all
db2 connect reset
db2rbind DBNAME all
```

Run **REORGCHK** and **rebind** when the system is relatively idle to ensure that a stable sample might be acquired and to avoid possible deadlocks in the catalog tables.

It is better to gather additional statistics; thus consider also using the following command for every table requiring attention, as follows:

runstats on table <schema>.<table> with distribution and detailed indexes

4.11.2 Setting buffer pool sizes correctly

A *buffer pool* is an area of memory into which database pages are read, modified, and held during processing. Buffer pools improve database performance. If a required page of data is already in the buffer pool, that page is accessed faster than if the page must be read directly from disk. As a result, the size of the DB2 buffer pools is critical to performance.

The amount of memory used by a buffer pool depends upon two factors:

- ▶ Size of buffer pool pages
- ▶ Number of pages allocated

Buffer pool page size is fixed at creation time and might be set to 4, 8, 16, or 32 KB. The most commonly used buffer pool is IBMDEFAULTBP, which has a 4000 MB page size.

All buffer pools are in database global memory, which is allocated on the database machine. The buffer pools must coexist with other data structures and applications, all without exhausting available memory. In general, having larger buffer pools improves performance up to a point by reducing I/O activity. Beyond that point, allocating additional memory no longer improves performance.

DB2 9.1 and later provide self-tuning memory management, which includes managing buffer pool sizes. The **self_tuning_mem** database level parameter, which is on by default, globally controls memory management. You can enable self-tuning for individual buffer pools using **SIZE AUTOMATIC** at CREATE or ALTER time.

To choose appropriate buffer pool size settings manually, monitor database container I/O activity by using system tools or by using DB2 buffer pool snapshots. Be careful to avoid configuring large buffer pool size settings that lead to paging activity on the system.

4.11.3 Maintaining correct table indexing

The DB2 Design Advisor provides suggestions for schema changes, including changes to indexes. To open the Design Advisor, complete the following steps:

1. From the Control Center, right-click a database in the left column
2. Click **DB2 Design Advisor** on the resulting menu.

4.11.4 Sizing log files appropriately

When using circular logging, it is important that the available log space permits dirty pages in the buffer pool to be cleaned at a reasonably low rate. Changes to the database are immediately written to the log, but a well-tuned database coalesces multiple changes to a page before eventually writing that modified page back to disk. Naturally, changes recorded only in the log cannot be overwritten by circular logging. DB2 detects this condition and forces the immediate cleaning of dirty pages required to allow switching to a new log file. Although this mechanism protects the changes recorded in the log, it suspends all application logging until the necessary pages are cleaned.

DB2 works to avoid pauses when switching log files by proactively triggering page cleaning under control of the database level **softmax** parameter. The default value of 100 for **softmax** begins background cleaning when the size gap between the current head of the log and the oldest log entry recording a change to a dirty page exceeds 100% of one log file. In extreme cases, this asynchronous page cleaning cannot keep up with log activity, leading to log switch pauses that degrade performance.

Increasing the available log space gives asynchronous page cleaning more time to write dirty buffer pool pages and avoid log switch pauses. A longer interval between cleanings makes it possible for multiple changes to be coalesced on a page before it is written, which reduces the required write throughput by making page cleaning more efficient.

Available logspace is governed by the product of log file size and the number of primary log files, which are configured at the database level. The **logfilsiz** setting is the number of 4 K pages in each log file. The **logprimary** setting controls the number of primary log files. The Control Center also provides a Configure Database Logging wizard.

As a starting point, try using 10 primary log files that are large enough that they do not wrap for at least a minute in normal operation.

Increasing the primary log file size has implications for database recovery. Assuming a constant value for **softmax**, larger log files mean that recovery might take more time. You can lower the **softmax** parameter to counter this effect, but keep in mind that more aggressive page cleaning might also be less efficient. Increasing the **softmax** parameter offers additional opportunities for write coalescing at the cost of longer recovery time.

The default value of **softmax** is 100, meaning that the database manager attempts to clean pages such that a single log file needs to be processed during recovery. For best performance, we suggest increasing this value to 300 as follows, meaning that three log files might need processing during recovery:

```
db2 update db config for <yourDatabaseName> using softmax 300
```

4.11.5 Setting inline length to improve throughput for high volume systems

When tuning database tables, consider setting inline length for large object (LOB) columns to potentially improve performance when running at high throughput rates. This parameter enables DB2 to optimize database access by storing the LOBs on the same page as the database row if the LOB fits within the inline length provided. Inlining LOBs can improve the performance of query and retrieval operations. Log traffic is increased when you set this parameter.

Inlining LOBs can apply to several columns. Of particular importance is setting inline length for the DATA column in the LSW_BPD_INSTANCE_DATA table. IBM internal performance evaluation shows that this setting can significantly improve throughput for high-volume production systems.

4.11.6 Using System Managed Storage for table spaces containing large objects

When creating REGULAR or LARGE table spaces in DB2 9.5 and later that contain performance critical LOB data, we suggest specifying **MANAGED BY SYSTEM** to gain the advantages of cached LOB handling in System Managed Storage (SMS).

This consideration applies to the following BPM-related products:

- ▶ Business Process Choreographer database (BPEDB)
- ▶ BPMN databases (twproc and twperfdb)
- ▶ Databases backing service integration bus messaging engine data stores

For background on using SMS for table spaces with large objects, see 4.10.5, “Avoiding double buffering” on page 79. A detailed explanation follows here.

DB2 table spaces can be configured with **NO FILE SYSTEM CACHING**, which in many cases improves system performance. If a table space is **MANAGED BY SYSTEM**, then it uses SMS, which provides desirable special case handling for LOB data regarding caching. Even if **NO FILE SYSTEM CACHING** is in effect (by default or as specified), access to LOB data still uses the file system cache.

If a table space is **MANAGED BY DATABASE**, it uses Database Managed Storage (DMS), which does not differentiate between LOB and non-LOB data regarding caching. In particular, **NO FILE SYSTEM CACHING** means that LOB access reads and writes directly to disk. Unconditionally reading LOBs from disk can cause high disk use and poor database performance.

Since Version 9.1, DB2 has by default created databases that use automatic storage (**AUTOMATIC STORAGE YES**). Creating databases using automatic storage means that the database manages disk space allocates itself from one or more pools of available file system space called storage paths. If automatic storage is enabled, **CREATE TABLESPACE** uses it by default (**MANAGED BY AUTOMATIC STORAGE**). For non-temporary table spaces, **REGULAR** and **LARGE**, automatic storage is implemented using DMS on files.

Before DB2 9.5, the default caching strategy for table spaces was **FILE SYSTEM CACHING**. In version 9.5, this strategy was changed to **NO FILE SYSTEM CACHING** for platforms where direct I/O or concurrent I/O is available. Taking defaults on version 9.5, we now have a database with **AUTOMATIC STORAGE YES**, and a table space that is **MANAGED BY AUTOMATIC STORAGE**, and in many cases, **NO FILE SYSTEM CACHING**. Such a table space, which is implemented using DMS, does not cache LOBs in the buffer pool or the file system.

4.11.7 Ensuring sufficient locking resources are available

Locks are allocated from a common pool controlled by the `locklist` database level parameter, which is the number of 4000 pages set aside for this use. A second database level parameter, `maxlocks`, bounds the percentage of the lock pool held by a single application. When an application attempts to allocate a lock that exceeds the fraction allowed by `maxlocks`, or when the free lock pool is exhausted, DB2 performs lock escalation to replenish the supply of available locks. Lock escalation involves replacing many row locks with a single table-level lock.

Although lock escalation addresses the immediate problem of lock pool overuse or starvation, it can lead to database deadlocks and thus should not occur frequently during normal operation. In some cases, application behavior can be altered to reduce pressure on the lock pool by breaking up large transactions that lock many rows into smaller transactions. It is simpler to try tuning the database first.

Beginning with version 9, DB2 adjusts the `locklist` and `maxlocks` parameters automatically by default. To tune these parameters manually, observe whether lock escalations are occurring either by examining `db2diag.log` or by using the system monitor to gather snapshots at the database level. If the initial symptom is database deadlocks, consider whether these deadlocks are initiated by lock escalations, as follows:

1. Check the lock escalations count in the output.

```
db2 get snapshot for database <yourDatabaseName>
```

2. Obtain current values for `locklist` and `maxlocks` by examining the output.

```
db2 get db config for <yourDatabaseName>
```

3. If necessary, alter these values, for example, to 100 for `locklist` and 20 for `maxlocks`, as shown in Example 4-3.

Example 4-3 Setting values for locklist and maxlocks

```
db2 update db config for <yourDatabaseName> using locklist 100 maxlocks 20
```

When increasing the locklist size, consider the impacts of the additional memory allocation required. Often, the locklist is relatively small compared with memory dedicated to buffer pools, but the total memory required must not lead to virtual memory paging.

When increasing the `maxlocks` fraction, consider whether a larger value allows a few applications to drain the free lock pool. This drain might lead to a new cause of escalations as other applications needing relatively few locks encounter a depleted free lock pool. Often, it is better to start by increasing locklist size alone.

4.11.8 Setting boundaries on the size of the catalog cache for clustered applications

The catalog cache is used to avoid repeating expensive activities, notably preparing execution plans for dynamic SQL. Thus it is important that the cache is sized appropriately.

By default, several 4 KB pages of memory are allocated for each possible application as defined by the `MAXAPPLS` database parameter. The multiplier is 4 for DB2 9, and 5 for DB2 9.5 and beyond. `MAXAPPLS` is `AUTOMATIC` by default, and its value is adjusted to match the peak number of applications connected at run time.

When running clustered applications, such as those deployed in the BPEL Process Choreographer in BPM V7.5, a value of more than 1000 for `MAXAPPLS` is possible. Such a value means that at least 4000 pages are allocated for the catalog cache, given default tuning. For the same workload, 500 pages are sufficient:

```
db2 update db config for <yourDatabaseName> using catalogcache_sz 500
```

The default behavior assumes heterogeneous use of database connections. A clustered application typically receives more homogeneous use across connections, allowing a smaller package cache to be effective. Bounding the package cache size frees up memory for other more valuable uses.

To tune the `CATALOGCACHE_SZ` database parameter manually, see the suggestions at the following website:

<http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0000338.htm>

4.11.9 Sizing the database heap appropriately before DB2 9.5

DB2 9.5 and later provide **AUTOMATIC** tuning of the database heap by default. We suggest using this value when available.

To tune the `DBHEAP` database parameter manually, see the suggestions documented at the following website:

<http://publib.boulder.ibm.com/infocenter/db2luw/v9/topic/com.ibm.db2.udb.admin.doc/doc/r0000276.htm>

4.11.10 Sizing the log buffer appropriately before DB2 9.7

Before DB2 Version 9.7, the default `LOGBUFSZ` was only eight pages. We suggest setting this value to 256, which is the default in Version 9.7:

```
db2 update db config for <yourDatabaseName> using logbufsz 256
```

4.11.11 Considering disabling current commit in DB2 V9.7 and later

DB2 9.7 supports new query semantics that always return the committed value of the data at the time the query is submitted. This support is active by default for newly created databases. In some cases, performance improves when you disable the new behavior, reverting to the original DB2 query semantics, as follows:

```
db2 update db config for <yourDatabaseName> using cur_commit disabled
```

4.11.12 Suggestions for Business Process Execution Language business processes in Business Process Manager V7.5

The following website explains the specification of initial DB2 database settings and provides examples of creating SMS table spaces for the BPEDB. It also contains useful links for planning and fine-tuning the BPEDB:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp?topic=/com.ibm.websphere.bpc.doc/doc/bpc/t5tuneint_spec_init_db_settings.html

The following website explains how to create DB2 databases for Linux, UNIX, and Windows for Business Process Choreographer. The following website gives details about BPEDB creation, including pointers for creating scripts for a production environment:

<http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp?topic=/com.ibm.websphere.bpc.doc/doc/bpc/t2codbdb.html>

4.11.13 Suggestions for Business Monitor V7.5

This section describes ways to tune and maximize the performance of Business Monitor V7.5.

Improving concurrency by setting registry variables

DB2 allows you to defer row locks for Cursor Stability (CS) or Read Stability (RS) isolation scans in certain cases. This deferral can last until a record is known to satisfy the predicates of a query when row locking is performed during a table or index scan.

To improve concurrency, it might be possible to defer row locking until after determining that a row qualifies for a query. Usually, concurrency is improved by setting the registry variables *DB2_SKIPDELETED* to permit scans to unconditionally skip uncommitted deletes, and *DB2_SKIPINSERTED* to permit scans to unconditionally skip uncommitted inserts. Example 4-4 illustrates how to enable these two registry variables for the database MONITOR:

Example 4-4 Enabling registry values DB2_SKIPDELETED and DB2_SKIPINSERTED

```
db2 connect to MONITOR
```

```
db2set DB2_SKIPDELETED=ON
```

```
db2set DB2_SKIPINSERTED=ON
```

Avoiding full transaction log

DB2 Health Monitor (HMON) regularly captures information about the database manager, database, table spaces, and tables. It calculates health indicators based on data retrieved from database system monitor elements, the operating system, and the DB2 system.

Transactions hang if HMON examines the status for the table *<MONMEBUS_SCHEMA>*.SIBOWNER and the transaction occupies the entire transaction log, resulting in an error with SQLCODE 964.

To prevent this error, you can disable the HMON process using the following command:

```
db2 update dbm cfg using HEALTH_MON OFF
```

Setting lock timeout properly

The parameter `LOCKTIMEOUT` specifies the number of seconds that an application waits to obtain a lock. This parameter helps avoid global deadlocks for applications. If you set this parameter to 0, the application does not wait for locks. In this situation, if the lock is unavailable at the time of the request, the application immediately receives a -911 return code.

If you set this parameter to -1, lock timeout detection is turned off.

A value of 30 seconds might be a good starting value; tune as necessary after setting this value. The following example shows how to set this parameter for the database MONITOR:

Example 4-5 Starting value for LOCKTIMEOUT

```
db2 -v update db cfg for MONITOR using LOCKTIMEOUT 30
```

Limiting event XML to 32 K where possible

Events that are small enough are persisted to a regular VARCHAR column in the incoming events table; large events are persisted to a binary large object (BLOB) column instead. In DB2, the largest VARCHAR column is 32768 bytes. Performance improves considerably when a VARCHAR column is used instead of a BLOB.

Using materialized query tables

When you have a deep history of data (for example, more than 10 million Monitoring Context instances), the response time for dashboard navigation in dimensional reports can degrade significantly. IBM Business Monitor V7.5 uses Cognos Business Intelligence V10.1 for its dimensional reports. BPM V7.5 provides a tool that can be used with DB2 to generate cube summary tables that pre-compute the values of measures for known dimensional member values. Such values include the value of the “average loan amount” measure based on values for the “customer loyalty level” dimension, say, for bronze, silver, and gold customers. DB2 calls these tables *materialized query tables*, and Business Monitor provides a scheduled service to refresh them on a periodic basis.

For details about using this capability, see the following Information Center topic:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r5mx/index.jsp?topic=/com.ibm.wbpm.mon.admin.doc/data/enable_cubesumtable_refresh.html

4.12 Oracle-specific database tuning

As with DB2, providing a comprehensive Oracle database tuning guide is beyond the scope of this paper. However, there are a few general rules of thumb that can assist in improving the performance of Oracle environments when used with BPM products. In the following sections, we describe these rules, and provide pointers to more detailed information. In addition, the following references are useful:

- ▶ Oracle Database 11g Release 1 documentation (includes a Performance Tuning Guide)
<http://www.oracle.com/pls/db111/homepage>
- ▶ *Oracle Architecture and Tuning on AIX v2.20* white paper
<http://www-03.ibm.com/support/techdocs/atmsastr.nsf/WebIndex/WP100883>

4.12.1 Updating database statistics

Oracle provides an automatic statistics gathering facility, which is enabled by default. One approach to updating statistics manually on all tables in a schema is to use the `dbms_stats` utility. For more information, see the Oracle product documentation.

4.12.2 Correctly setting buffer cache sizes

Oracle provides automatic memory management for buffer caches. For more information about configuring automatic memory management and for guidance on manually setting buffer cache sizes, see the following references:

- ▶ For Oracle 10g R2

http://download.oracle.com/docs/cd/B19306_01/server.102/b14211/memory.htm#i29118

- ▶ For Oracle 11g R1

http://download.oracle.com/docs/cd/B28359_01/server.111/b28274/memory.htm#i29118

4.12.3 Maintaining correct table indexing

The SQL Access Advisor, available from the Enterprise Manager, provides suggestions for schema changes, including changes to indexes. You can find the SQL Access Advisor by starting at the database home page, then following the **Advisor Central** link in the Related Links section at the bottom of the page.

4.12.4 Sizing log files appropriately

Unlike DB2, Oracle performs an expensive checkpoint operation when switching logs. The checkpoint involves writing all dirty pages in the buffer cache to disk. Thus, it is important to make the log files large enough that switching occurs infrequently. Applications that generate a high volume of log traffic need larger log files to achieve this goal.

4.12.5 Suggestions for Business Process Execution Language business processes in Business Process Manager V7.5

Web material is available that offers suggestions for tuning BPEL business processes in BPM V7.5.

The following website explains how to specify initial Oracle database settings:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp?topic=/com.ibm.websphere.bpc.doc/doc/bpc/t5tuneint_spec_init_db_oracle.html

The following website explains how to create an Oracle database for Business Process Choreographer. It provides details about BPEDB creation, including pointers to useful creation scripts for a production environment:

<http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp?topic=/com.ibm.websphere.bpc.doc/doc/bpc/t2codbdb.html>

The default Oracle policy for LOBs is to store the data within the row when the size of the object does not exceed a threshold. In some cases, workloads have LOBs that regularly exceed this threshold. By default, such LOB accesses bypass the buffer cache, meaning that

LOB reads are exposed to disk I/O latencies when using the preferred direct or concurrent path to storage.

4.13 Advanced Java heap tuning

Because the BPM product set is written in Java, the performance of the JVM has a significant impact on the performance delivered by these products. JVMs externalize multiple tuning parameters that might be used to improve both authoring and runtime performance. The most important of these parameters are related to garbage collection and setting the Java heap size. This section explains these topics in detail.

The products covered in this paper use IBM JVMs on most platforms (for example, AIX, Linux, and Windows), and the HotSpot JVMs on selected other systems, such as Solaris and HP-UX. Vendor-specific JVM implementation details and settings are described as appropriate. All BPM V7 products in this document use Java 6. It has characteristics similar to Java 5 used in BPM V6.1 and V6.2.0 products but is much different from Java 1.4.2 used by BPM V6.0.2.x and earlier versions. For brevity, only Java 6 tuning is described here. The IBM Java 6 Diagnostics Guide is at the following website:

<http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/index.jsp>

The guide describes many more tuning parameters than those described in this paper, but most are for specific situations and are not of general use. For a more detailed description of IBM Java 6 garbage collection algorithms, see the section on memory management.

The following sites contain additional Sun HotSpot JVM references:

- ▶ Summary of HotSpot JVM options for Solaris
<http://java.sun.com/docs/hotspot/VMOptions.html>
- ▶ FAQs about the Solaris HotSpot JVM
<http://java.sun.com/docs/hotspot/PerformanceFAQ.html#20>
- ▶ Additional tuning information for Sun HotSpot JVM
<http://java.sun.com/docs/performance/>

4.13.1 Monitoring garbage collection

To set the heap correctly, determine how the heap is being used by collecting a verbosegc trace. A verbose garbage collection (verbosegc) trace prints garbage collection actions and statistics to stderr in IBM JVMs and stdout in Sun HotSpot JVMs. The verbosegc trace is activated by using the Java runtime option `-verbose:gc`. Output from verbosegc is different for the HotSpot and IBM JVMs, as shown by Example 4-6 and Example 4-7.

Example 4-6 Example IBM JVM verbosegc trace output

```
<af type="tenured" id="12" timestamp="Fri Jan 18 15:46:15 2008" intervalms="86.539">
  <minimum requested_bytes="3498704" />
  <time exclusiveaccessms="0.103" />
  <tenured freebytes="80200400" totalbytes="268435456" percent="29" >
    <soa freebytes="76787560" totalbytes="255013888" percent="30" />
    <loa freebytes="3412840" totalbytes="13421568" percent="25" />
  </tenured>
  <gc type="global" id="12" totalid="12" intervalms="87.124">
    <refs_cleared soft="2" threshold="32" weak="0" phantom="0" />
    <finalization objectsqueued="0" />
    <timesms mark="242.029" sweep="14.348" compact="0.000" total="256.598" />
    <tenured freebytes="95436688" totalbytes="268435456" percent="35" >
      <soa freebytes="87135192" totalbytes="252329472" percent="34" />
      <loa freebytes="8301496" totalbytes="16105984" percent="51" />
    </tenured>
  </gc>
  <tenured freebytes="91937984" totalbytes="268435456" percent="34" >
    <soa freebytes="87135192" totalbytes="252329472" percent="34" />
    <loa freebytes="4802792" totalbytes="16105984" percent="29" />
  </tenured>
  <time totalms="263.195" />
</af>
```

Example 4-7 Example Solaris HotSpot JVM verbosegc trace output (young and old)

```
[GC 325816K → 83372K(776768K), 0.2454258 secs]
[Full GC 267628K → 83769K <- live data (776768K), 1.8479984 secs]
```

Sun HotSpot JVM verbosegc output can be more detailed by setting additional options:

- ▶ `-XX:+PrintGCDetails`
- ▶ `-XX:+PrintGCTimeStamps`

It can be tedious to parse the verbosegc output using a text editor. Visualization tools that can be used for more effective Java heap analysis are available on the web. The IBM Pattern Modeling and Analysis Tool (PMAT) for Java Garbage Collector is one such tool. It is available for download at IBM alphaWorks® at the following website:

<http://www.alphaworks.ibm.com/tech/pmat>

PMAT supports the verbosegc output formats of JVMs offered by major JVM vendors such as IBM, Sun, and HP.

4.13.2 Setting the heap size for most configurations

This section contains guidelines for determining the appropriate Java heap size for most configurations. If your configuration requires that more than one JVM run concurrently on the same system, see 4.13.3, “Setting the heap size when running multiple JVMs on one system” on page 92. For example, you might need more than one JVM if you run both a BPM server and Integration Designer on the same system. If your objective is designed to support large business objects, read 4.4.2, “Tuning for large objects” on page 56.

When the heap size is too low, Out Of Memory errors occur. For most production applications, the IBM JVM Java heap size defaults are too small, so increase them. When the heap size is too low, Out Of Memory errors occur. In general, the HotSpot JVM default heap and nursery size are also too small so increase them as well.

There are several approaches to setting optimal heap sizes. Here we describe the approach that most applications can use when running the IBM JVM on AIX. The essentials can be applied to other systems. Set the initial heap size (`-Xms` option) to a typical value (for example, 768 MB on a 64-bit JVM, or 256 MB on a 32-bit JVM). Set the maximum heap size (`-Xmx`) option to a typical value, but large (for example, 3072 MB on a 64-bit JVM, or 1024 MB on a 32-bit JVM). The maximum heap size should never force the heap to page. It is imperative that the heap always stays in physical memory. The JVM then tries to keep the GC time within reasonable limits by growing and shrinking the heap. The output from `verbosegc` is used to monitor GC activity.

If Generational Concurrent GC is used (`-Xgcpolicy:gencon`), you can also set the new area size to specific values. By default, the new size is a quarter of the total heap size or 64 MB, whichever is smaller. For better performance, set the nursery size to half of the heap size or larger, and do not cap the value at 64 MB. You can set new area sizes using the following JVM options:

- ▶ `-Xmn<size>`
- ▶ `-Xmns<initialSize>`
- ▶ `-Xmnx<maxSize>`

You can use a similar process to set the size of HotSpot heaps. In addition to setting the minimum and maximum heap size, also increase the nursery size to between 1/4 and 1/2 of the heap size. Never increase the nursery to more than half the full heap.

You can set the nursery size using the `MaxNewSize` and `NewSize` parameters:

- ▶ `-XX:MaxNewSize=128m`
- ▶ `-XX:NewSize=128m`

If you are using a 64-bit IBM JVM, use the `-Xgc:preferredHeapBase` parameter to avoid native OutOfMemory issues due to exhaustion of memory addresses below 4 GB (for example, classes and threads are allocated in this region). The option `-Xgc:preferredHeapBase` can be used to move the Java heap outside of the lower 4 GB address space. See the IBM JVM Information Center for a more detailed description of the solution:

http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/topic/com.ibm.java.doc.diagnostics.60/diag/understanding/mm_compressed_references.html?resultof=%22%63%6f%6d%70%72%65%73%73%22%20

After the heap sizes are set, use `verbosegc` traces to monitor GC activity. After analyzing the output, modify the heap settings accordingly. For example, if the percentage of time in GC is high and the heap has grown to its maximum size, you might improve throughput by increasing the maximum heap size. As a rule of thumb, greater than 10% of the total time spent in GC is generally considered high.

Increasing the maximum size of the Java heap might not always solve this type of problem as it might be a memory overuse problem. Conversely, if response times are too long due to GC pause times, decrease the heap size. If both problems are observed, an analysis of the application heap usage is required.

4.13.3 Setting the heap size when running multiple JVMs on one system

Each running Java program has a heap associated with it. If you have a configuration where more than one Java program is running on a single physical system, setting the heap sizes appropriately is of particular importance. Setting the heap size is also important for 32-bit systems, where the total amount of addressable memory is limited.

An example of one such configuration is when the Integration Designer is on the same physical system as a BPM server using a 32-bit JVM. Each of these applications is a separate Java program that has its own Java heap. If the sum of all of the virtual memory usage (including both Java heaps and all other virtual memory allocations) exceeds the size of addressable physical memory, the Java heaps are subject to paging. Such paging causes total system performance to degrade significantly. To minimize the possibility of total system degradation, use the following guidelines:

First, collect a `verbosegc` trace for each running JVM.

- ▶ Based on the `verbosegc` trace output, set the initial heap size to a relatively low value. For example, assume that the `verbosegc` trace output shows that the heap size grows quickly to 256 MB, and then grows more slowly to 400 MB and stabilizes at that point. Based on this change, set the initial heap size to 256 MB (`-Xms256m`).
- ▶ Also based on the `verbosegc` trace output, set the maximum heap size appropriately. Take care not to set this value too low, or Out Of Memory errors occur. The maximum heap size must be large enough to allow for peak throughput. Using the same example, a maximum heap size of 768 MB might be appropriate (`-Xmx768m`). Correct sizing of maximum heap gives the Java heap room to expand beyond its current size of 400 MB, if required. The Java heap grows only if required (for example, if a period of peak activity drives a higher throughput rate), so setting the maximum heap size higher than current requirements is generally a good policy.
- ▶ Be careful not to set the heap sizes too low, or garbage collections will occur frequently, which might reduce throughput. Again, a `verbosegc` trace assists in determining garbage collection frequency. You must ensure that the heap sizes are large enough that garbage collections do not occur too often. At the same time, you must still ensure that the heap sizes are not cumulatively so large as to cause the heap to page. This balancing act is configuration-dependent.

4.13.4 Reducing or increasing heap size if OutOfMemory errors occur

The `java.lang.OutOfMemory` exception is used by the JVM in various circumstances, making it sometimes difficult to track down the source of the exception. There is no conclusive mechanism for telling the difference between these potential error sources, but a good start is to collect a trace using `verbosegc`. If the problem is a lack of memory in the heap, you can easily see this condition in the output. For more information about `verbosegc` output, see 4.13.1, “Monitoring garbage collection” on page 90. Many garbage collections that produce little free heap space generally occur preceding this exception. If this lack of free heap space is the problem, increase the size of the heap.

If there is enough free memory when the `java.lang.OutOfMemory` exception occurs, the next item to check is the finalizer count from the `verbosegc` (only the IBM JVM provides this information). If this count appears high, a subtle effect might be occurring whereby resources outside the heap are held by objects within the heap and being cleaned by finalizers. Reducing the size of the heap can alleviate this situation by increasing the frequency with which finalizers are run. In addition, examine your application to determine whether the finalizers can be avoided or minimized.

`OutOfMemory` errors can also occur for issues unrelated to JVM heap usage, such as running out of certain system resources. Examples of this problem include insufficient file handles or thread stack sizes that are too small.

In some cases, you can tune the configuration to avoid running out of native heap. Try reducing the stack size for threads (the `-Xss` parameter). Deeply nested methods might force a thread stack overflow if there is insufficient stack size.

Also, if you are using a 64-bit IBM JVM, use the `-Xgc:preferredHeapBase` parameter to avoid native `OutOfMemory` issues due to exhaustion of memory addresses below 4 GB (for example, classes and threads are allocated in this region). You can use the option `-Xgc:preferredHeapBase` to move the Java heap outside of the lower 4 GB address space. See the IBM JVM Information Center for a more detailed description of the solution:

http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/topic/com.ibm.java.doc.diagnostics.60/diag/understanding/mm_compressed_references.html?resultof=%22%63%6f%6d%70%72%65%73%73%22%20

For middleware products, if you are using an in-process version of the JDBC driver, it is possible to find an out-of-process driver that can have a significant effect on the native memory requirements. For example, you can use type 4 JDBC drivers (DB2 Net drivers or Oracle's Thin drivers), or you can switch IBM MQSeries® from Bindings mode to Client mode. See documentation for DB2, Oracle, and MQSeries for more details.

4.14 Tuning for WebSphere InterChange Server migrated workloads

The following tuning suggestions are unique to workloads migrated using the WebSphere InterChange Server migration wizard in the Integration Designer. In addition to these suggestions, see the other BPM server tuning suggestions detailed in this document:

- ▶ For JMS-based messaging used to communicate with older WBI adapters or custom adapters, use non-persistent queues when possible.
- ▶ For JMS-based messaging used to communicate with older WBI adapters or custom adapters, use WebSphere MQ-based queues, if available. By default, the adapters use the WebSphere MQ APIs to connect to the service integration bus destinations through MQ Link. MQ Link is a protocol translation layer that converts messages to and from MQ-based clients. By switching to WebSphere MQ-based queues, MQLink translation costs are eliminated and performance is improved.
- ▶ Turn off server logs for verbose workloads. Some workloads emit log entries for every transaction, causing constant disk writes and reducing overall throughput. Turning off server logs might reduce the throughput degradation for such workloads.



Initial configuration settings

In this chapter, we suggest initial settings for several relevant parameters. These values are not optimal in all cases, but the values work well in internal performance evaluations. They are, at a minimum, useful starting points for many proof-of-concepts and customer deployments. As described in 4.1, “Performance tuning methodology” on page 48, tuning is an iterative process. Follow that procedure and adjust these values as appropriate for your environment.

5.1 Business Process Manager server settings

This section provides settings based on IBM internal performance evaluations of the Business Process Manager (BPM) V7.5 server. These settings were derived using the tuning methodology and guidelines described in Chapter 4, “Performance tuning and configuration” on page 47. Consider these settings useful starting points for your use of this product. For settings that we do not list, use the default settings that are supplied by the product installer as a starting point, and follow the tuning methodology specified in 4.1, “Performance tuning methodology” on page 48.

We describe three settings in this section:

- ▶ A three-tiered setup for Business Process Execution Language (BPEL) business processes with the production database on a separate server.
- ▶ A three-tiered setup for Business Processing Modeling Notation (BPMN) business processes with the production database on a separate server.
- ▶ A two-tiered (client/server) setup for BPEL business processes with the production database collocated on the server.

5.1.1 Three-tiered configuration with Web services and remote DB2 system

Through the WebSphere Application Server, we used a three-tiered configuration in our internal performance work to evaluate the performance of a BPEL business process that models automobile insurance claims processing. This configuration is an example of many production environments where DB2 is on a separate system than the BPM server. The Web services binding was used for communications. The business process has two modes of operation:

- ▶ A BPEL microflow (straight-through process) that processes claims where no human intervention is required
- ▶ A BPEL microflow plus macroflow (long-running process) pattern, where the macroflow is started when a review or approval is required (for example, if the claim amount is above a certain limit)

Three systems were used in this configuration:

- ▶ Request driver
- ▶ BPM V7.5 server
- ▶ DB2 database server

The BPM server and the DB2 database server required extensive tuning to maximize throughput. Some tuning varied due to the operating system (such as AIX and Windows) and the number of processor cores. We present these variations in tabular format after describing common tuning.

For all topologies in this section, we suggest you complete the following actions to tune BPM and DB2 and maximize throughput:

- ▶ Use the production template.
- ▶ Define the Common database as local DB2 type 4.
- ▶ Establish BPEL Business Process support with `bpeconfig.jacl`. Click **Data sources** → **BPEDataSourceDb2** → WebSphere Application Server data source properties statement cache to 300).
- ▶ Disable PMI.

- ▶ Set HTTP `maxPersistentRequests` to -1.
Set GC policy to `-Xgcpolicy:gencon` (see Table 5-1 and Table 5-2 for nursery setting `-Xmn`).
- ▶ Use remote DB2 databases (connection type 4) for the SIB System, BPCDB, and SIB BPEDB.

Table 5-1 shows BPM V7.5 server-related settings to modify from their default value when the BPM V7.5 server is deployed on AIX.

Table 5-1 Three-tiered application cluster settings for AIX

Setting	Value
Java heap Megabytes	1536
Java nursery Megabytes -Xmn	768
Default thread pool max	100
BPEDB Data source → connection pool max	300
BPEDB Data source → WebSphere Application Server data source properties → Statement cache size	300
BPC messaging engine data source → connection pool max	50
SCA SYSTEM messaging engine data source → connection pool max	50
BPM Common Data source → connection pool max	500
J2C activation specifications → SOABenchBPELMod2_AS → Custom properties → maxConcurrency, maxBatchSize	50,
Resources → Asynchronous Beans → Work Managers → BPENavigationWorkManager → Work request queue size, max threads, growable	400, 50, no
Application Cluster → Business Flow Manager → Message pool size, Intertransaction cache size	5000, 400
WebContainer thread pool min, max	100, 100
com.ibm.websphere.webservices.http.maxConnection	50

Table 5-2 shows BPM V7.5 server-related settings to modify from their default value when BPM V7.5 server is deployed on Windows and Linux on Intel systems.

Table 5-2 Three-tiered web service and remote DB2 tuning variations for Windows and Linux on Intel systems

Tuning variations	Microflow: Number of cores			Macroflow: Number of cores	
	1	2	4	1	4
Java heap Megabytes	1280	1280	1280	1280	1280
Java nursery Megabytes -Xmn	640	640	640	768	768
Web container thread pool max	100	150	150	100	300
Default thread pool max	100	200	200	100	200
BPE database connection pool max	150	250	250	150	350

Tuning variations	Microflow: Number of cores			Macroflow: Number of cores	
	1	2	4	1	4
BPC messaging engine database connection pool max	30	30	30	30	150
SYSTEM messaging engine database connection pool max	30	40	40	30	100
Common database connection pool max	80	80	80	80	100
J2C activation specifications → SOABenchBPELMod2_AS → Custom properties → maxConcurrency	40	40	40	160	160
BPEInternalActivationSpec batch size				10	10
SOABenchBPELMod2_AS batch size				32	32
Java custom property com.ibm.websphere.webservic es.http.maxConnection	100	200	200	200	200
Application servers → server1 → Business Flow Manager → allowPerformanceOptimizations				Yes	Yes
Application servers → server1 → Business Flow Manager → interTransactionCache.size				400	400
Application servers → server1 → Business Flow Manager → workManagerNavigation.messa gePoolSize				4000	4000
Resources → Asynchronous Beans → Work Managers → BPENavigationWorkManager → min threads, max threads, request queue size				30, 30, 30	30, 30, 30

The DB2 database server has several databases defined for use by the BPM V7.5 server. Spread the database logs and table spaces across a RAID array to distribute disk use. Tune the SCA.SYSTEM.<cellname>.BUS database and the BPEDB as follows:

- ▶ **db2 update db cfg for sysdb using logbufsz 512 logfilsiz 8000 logprimary 20 logsecond 20 auto_runstats off**
- ▶ **db2 alter bufferpool ibmdefaultbp size 30000**

Create and tune the BPE database using the following DB2 commands and generated scripts:

- ▶ **db2 CREATE DATABASE bpedb ON /raid USING CODESET UTF-8 TERRITORY en-us**
- ▶ **db2 update db cfg for bpedb using logbufsz 512 logfilsiz 10000 logprimary 20 logsecond 10 auto_runstats off**
- ▶ **db2 -tf createTablespace.sql** (BPM V7.5 server generated script)
- ▶ **db2 -tf createSchema.sql** (BPM V7.5 server generated script)
- ▶ **db2 alter bufferpool ibmdefaultbp size 132000**
- ▶ **db2 alter bufferpool bpebp8k size 132000**

5.1.2 Three-tiered configuration using Human Services with Business Processing Modeling Notation processes

We used a three-tiered configuration in our internal performance work to evaluate the performance of a BPMN business process that models automobile insurance claims processing. Human Services were used to process the claims with a Call Center scenario of Query Tasks, Claim Task, Complete Task, and Commit Task.

This configuration is an example of many production environments where DB2 is on a separate system than the BPM server. The Web services open SCA binding was used for communications. Three systems were used in this configuration:

- ▶ Request driver
- ▶ BPM V7.5 server
- ▶ DB2 database server

BPM V7.5 server in three-tiered configuration

The following settings are recommended to use BPM V7.5 in a three-tiered configuration.

- ▶ For a 64-bit JVM, alter Java memory management settings by adding Java command-line parameters **-Xgencon, -Xms1800M, -Xmx1800M, -Xmn800M**
- ▶ Increase size of the WebContainer ThreadPool to a minimum of 200, maximum of 400
- ▶ Increase the maximum size of the Default Thread Pool to 40
- ▶ Disable logging for selected BPM V7.5 components (for example, Web services)

DB2 database server in three-tiered configuration

The following settings are required to use the DB2 database server in a three-tiered configuration.

- ▶ Separate log files and containers onto separate (RAID) disks
- ▶ Increase log file size for twproc database (to 16384 pages)
- ▶ Enable file system caching for twproc database:
db2 alter tablespace userspace1 file system caching
- ▶ Exclude the table SIBOWNER from automatic runstats execution, as described in the following technote:

<http://www-01.ibm.com/support/docview.wss?uid=swg21452323>

- ▶ Ensure that database statistics are up to date.
If running at high throughput rates, consider disabling some or all database auto-maintenance tasks to avoid impacting peak throughput. However, if you disable these capabilities, be sure to perform **runstats** regularly to update database statistics
- ▶ Create new indexes as described in the following technote:
<http://www-01.ibm.com/support/docview.wss?uid=swg21474536>
Create the indexes using the following commands (if using DB2; other databases have similar commands):
 - db2 "CREATE INDEX IDX_SOAB_BPD_INSTANCE ON LSW_BPD_INSTANCE (SNAPSHOT_ID ASC, BPD_INSTANCE_ID ASC) ALLOW REVERSE SCANS COLLECT SAMPLED DETAILED STATISTICS"
 - db2 "CREATE INDEX IDX_SOAB_BPD_INSTANCE_VAR ON LSW_BPD_INSTANCE_VARIABLES (BPD_INSTANCE_ID ASC, VARIABLE_TYPE ASC, ALIAS ASC) ALLOW REVERSE SCANS COLLECT SAMPLED DETAILED STATISTICS"

5.1.3 Two-tiered configuration using file store for Java Message Service

We used a two-tiered configuration to evaluate the performance of a long-running business process that models a typical mortgage application process. This configuration is used with the BPM V7.5 server and DB2 on the same physical system as is common for proofs-of-concept when limited hardware is available. However, the configuration is not a representative production configuration. Java Message Service (JMS) binding is used for communication.

In this configuration, the BPE uses a DB2 database, and the messaging engines are configured to use file stores. To select the file store option, start the Profile Management Tool, click **Advanced Profile Creation**, and in the Database Configuration window, click **Use a file store for Messaging Engines**.

Tuning parameter settings for the BPE database were initially derived using the DB2 Configuration Advisor. A few key parameter settings are modified further:

- ▶ **MAXAPPLS** is enlarged enough to accommodate connections from all possible JDBC Connection Pool threads.
- ▶ The default buffer pool sizes (number of 4 KB pages in IBMDEFAULTBP) for each database are set so that each pool is 256 MB in size.

Table 5-3 shows the parameter settings suggested for this configuration.

Table 5-3 Two-tiered configuration using JMS file store parameter settings

Parameter names	Business Process Choreographer database (BPEDB) settings
APP_CTL_HEAP_SZ	144
APPGROUP_MEM_SZ	13001
CATALOGCACHE_SZ	521
CHNGPGS_THRESH	55
DBHEAP	600
LOCKLIST	500
LOCKTIMEOUT	30
LOGBUFSZ	245
LOGFILSIZ	1024
LOGPRIMARY	11
LOGSECOND	10
MAXAPPLS	90
MAXLOCKS	57
MINCOMMIT	1
NUM_IOCLEANERS	6
NUM_IOSERVERS	10
PCKCACHESZ	915
SOFTMAX	440
SORTHEAP	228
STMHEAP	2048
DFT_DEGREE	1
DFT_PREFETCH_SZ	32
UTIL_HEAP_SZ	11663
IMBDEFAULTBP	65536

In addition to these database-level parameter settings, you must modify several other parameters using the administrative console. These settings primarily affect concurrency (thread settings):

- ▶ The amount of expected concurrency influences the size of the thread pool because more in-flight transactions require more threads. As a possible remedy, you might need to adjust increase the size of the default thread pool beyond the default of 50 threads.
- ▶ Set the maximum concurrency to 50 threads for activation specifications.
- ▶ For the Business Process Choreographer Database (BPEDB), increase the database connection pool size to 60 and the statement cache size to 300.

- ▶ Set the Maximum Connections property for JMS connection pools to 40.
- ▶ Connect to the local database using the DB2 JDBC Universal Driver type 2 driver. Type 2 drivers produce better performance when the BPM V7.5 server and DB2 are on the same physical system.
- ▶ Set the BPM V7.5 server JVM heap size to a fixed size of 1024 MB. In addition, use the gencon garbage collection policy.

5.2 WebSphere ESB settings

This section describes settings used for selected internal performance evaluations of WebSphere ESB. These settings were derived using the tuning methodology and guidelines described in Chapter 4, “Performance tuning and configuration” on page 47. Consider these settings starting points for using WebSphere ESB. For settings that are not listed, use the default settings supplied by the product installer as a starting point. See 4.1, “Performance tuning methodology” on page 48 for a description of tuning methodology.

5.2.1 WebSphere ESB common settings

WebSphere ESB settings are good starting points for tuning a WebSphere ESB solution, regardless of binding choices:

- ▶ Tracing is disabled.
- ▶ If security is required, application security instead of Java2 security is used to reduce processor usage.
- ▶ Java heap size is set to 1280 MB for Windows and 1280 MB for AIX.
- ▶ Gencon garbage collection policy is enabled (`-Xgcpolicy:gencon`), setting the nursery heap size to 1024 MB.

5.2.2 WebSphere ESB settings for Web services

The WebSphere ESB settings for Web services are as follows:

- ▶ PMI monitoring is disabled
- ▶ Value of WebContainer thread pool sizes is set to max 50 and min 10
- ▶ Value of WebContainer thread pool inactivity timeouts for thread pools is set to 3500

5.2.3 WebSphere ESB settings for Java Message Service

The WebSphere ESB settings for WebSphere MQ and JMS are as follows:

- ▶ Activation specification
 - Set maximum concurrent endpoints to 50
- ▶ Queue Connection factory
 - Set the maximum connection pool size to 51
- ▶ DiscardableDataBufferSize
 - Set to 10 MB and CachedDataBufferSize set to 40 MB

5.2.4 DB2 settings for Java Message Service persistent

Set the following values. They are only relevant for JMS persistent configurations as they use the database to persist messages:

- ▶ Place database tablespaces and logs on a fast disk subsystem
- ▶ Place logs on separate device from table spaces
- ▶ Set buffer pool size correctly
- ▶ Set the connection min and max to 30
- ▶ Set the statement cache size to 40
- ▶ Set up a raw partition for DB2 logs

Otherwise, unless noted in the workload description, use the default settings as supplied by the product installer.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this paper.

IBM Redbooks

For information about ordering this publication, see “How to get Redbooks” on page 106. The document referenced here might be available in softcopy only.

IBM Business Process Manager V7.5. Production Topologies, SG24-7976:

<http://www.redbooks.ibm.com/abstracts/sg247976.html>

Online resources

These websites are also relevant as further information sources:

- ▶ WebSphere Application Server Performance
<http://www.ibm.com/software/webservers/appserv/was/performance.html>
- ▶ WebSphere Application Server Information Center (including Tuning Guide)
http://www-306.ibm.com/software/webservers/appserv/was/library/?S_CMP=rnav
- ▶ DB2 Version 9 best practices
http://www.ibm.com/developerworks/data/bestpractices/?&S_TACT=105AGX11&S_CM
- ▶ DB2 Version 9.7 Information Center
<http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/index.jsp>
- ▶ Diagnostics Guide for IBM SDK and Runtime Environment Java Technology Edition, Version 6
<http://publib.boulder.ibm.com/infocenter/javasdk/v6r0/index.jsp>
- ▶ IBM Pattern Modeling and Analysis Tool for Java Garbage Collector
<http://www.alphaworks.ibm.com/tech/pmat>
- ▶ Oracle 11g Documentation Library
<http://www.oracle.com/pls/db111/homepage>
- ▶ IBM Integration Designer V7.5 Information Center
<http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r5mx/topic/com.ibm.wbpm.mai.n.do.homepage-bpm.html>
- ▶ IBM Business Process Management V7.5 Information Center
<http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r5mx/topic/com.ibm.wbpm.mai.n.do.homepage-bpm.html>

- ▶ Performance tuning resources for WebSphere Process Server and IBM BPM solutions
http://www.ibm.com/developerworks/websphere/library/techarticles/1111_herrmann/1111_herrmann.html
- ▶ Understanding and Tuning the Event Manager
<http://www-01.ibm.com/support/docview.wss?uid=swg21439613>
- ▶ Best practices for DB2 for Linux, UNIX, and Windows
<http://www.ibm.com/developerworks/data/bestpractices/db2luw/>
- ▶ DB2 Version 9.5 Information Center
<http://publib.boulder.ibm.com/infocenter/db2luw/v9r5/index.jsp>
- ▶ Extending a J2CA adapter for use with WebSphere Process Server and WebSphere Enterprise Service Bus
<http://www-128.ibm.com/developerworks/library/ws-soa-j2caadapter/index.html?ca=drs->
- ▶ Support home page links for WebSphere Enterprise Service Bus
<http://www-306.ibm.com/software/integration/wsesb/support/>
- ▶ Oracle Architecture and Tuning on AIX V2.2.0
<http://www-03.ibm.com/support/techdocs/atmsastr.nsf/WebIndex/WP100883>
- ▶ Oracle 10g Release 2 Documentation Library (including Performance Tuning Guide)
<http://www.oracle.com/pls/db102/homepage>

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, technotes, draft publications and Additional materials, and order hardcopy Redbooks publications, at this website:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



IBM Business Process Manager V7.5 Performance Tuning and Best Practices



Learn valuable tips for tuning

Get the latest best practices

See example settings

This IBM Redpaper publication provides performance tuning tips and best practices for IBM Business Process Manager (BPM) V7.5 (all editions) and IBM Business Monitor V7.5. These products represent an integrated development and runtime environment based on a key set of service-oriented architecture (SOA) and business process management technologies. Such technologies include Service Component Architecture (SCA), Service Data Object (SDO), Business Process Execution Language for Web services (BPEL), and Business Processing Modeling Notation (BPMN).

Both BPM and Business Monitor build on the core capabilities of the IBM WebSphere Application Server infrastructure. As a result, BPM solutions benefit from tuning, configuration, and best practices information for WebSphere Application Server and the corresponding platform Java Virtual Machines (JVMs).

This paper targets a wide variety of groups, both within IBM (development, services, technical sales, and others) and customers. For customers who are either considering or are in the early stages of implementing a solution incorporating BPM and Business Monitor, this document proves a useful reference. The paper is useful both in terms of best practices during application development and deployment and as a reference for setup, tuning, and configuration information.

This paper introduces many of the issues influencing the performance of each product and can serve as a guide for making rational first choices in terms of configuration and performance settings. Similarly, customers who have already implemented a solution using these products might use the information presented here to gain insight into how their overall integrated solution performance might be improved.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks