WebSphere Business Integration V6 Performance Tuning

Learn valuable tips for tuning

Get the latest best practices

Try the example settings

IBM WebSphere Process Server, WebSphere Enterprise Service Bus, WebSphere Adapters, WebSphere Business Monitor Performance teams

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

First Edition (September 2006)

This edition applies to WebSphere Process Server 6.0.1.1, WebSphere Enterprise Service Bus 6.0.1.1, WebSphere Integration Developer 6.0.0, WebSphere Adapters 6.0.0.1, WebSphere Business Monitor 6.0.0.
# Contents

<table>
<thead>
<tr>
<th>Notices</th>
<th>vii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trademarks</td>
<td>viii</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Preface</strong></td>
<td>ix</td>
</tr>
<tr>
<td>The team that wrote this Redpaper</td>
<td>ix</td>
</tr>
<tr>
<td>Become a published author</td>
<td>x</td>
</tr>
<tr>
<td>Comments welcome</td>
<td>xi</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Chapter 1. Performance tuning and configuration</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Tuning checklists</td>
<td>2</td>
</tr>
<tr>
<td>1.2 WebSphere Process Server</td>
<td>3</td>
</tr>
<tr>
<td>1.2.1 Do not run production server in development mode</td>
<td>3</td>
</tr>
<tr>
<td>1.2.2 Disable tracing and monitoring when possible</td>
<td>3</td>
</tr>
<tr>
<td>1.2.3 Move databases off of the default Cloudscape</td>
<td>3</td>
</tr>
<tr>
<td>1.2.4 Configure the statement cache for long-running processes</td>
<td>7</td>
</tr>
<tr>
<td>1.2.5 Configure threads for messaging and in work managers</td>
<td>7</td>
</tr>
<tr>
<td>1.2.6 Use an appropriate Java heap size for production environments</td>
<td>7</td>
</tr>
<tr>
<td>1.2.7 Configure WebSphere Process Server for clustering</td>
<td>8</td>
</tr>
<tr>
<td>1.3 WebSphere ESB</td>
<td>9</td>
</tr>
<tr>
<td>1.3.1 Disable tracing and monitoring when possible</td>
<td>9</td>
</tr>
<tr>
<td>1.3.2 Move databases off the default Cloudscape</td>
<td>9</td>
</tr>
<tr>
<td>1.3.3 Set the WebContainer thread pool settings</td>
<td>10</td>
</tr>
<tr>
<td>1.3.4 Set the Java heap size</td>
<td>10</td>
</tr>
<tr>
<td>1.3.5 Increase the maximum concurrent endpoints in activation specifications</td>
<td>10</td>
</tr>
<tr>
<td>1.3.6 Increase maximum connection pool size in the queue connection factory</td>
<td>10</td>
</tr>
<tr>
<td>1.3.7 Set HTTP connectionKeepAlive</td>
<td>11</td>
</tr>
<tr>
<td>1.3.8 Set DiscardableDataBufferSize and CachedDataBufferSize</td>
<td>11</td>
</tr>
<tr>
<td>1.3.9 Set the DTMManager property</td>
<td>11</td>
</tr>
<tr>
<td>1.3.10 DB2 tuning</td>
<td>11</td>
</tr>
<tr>
<td>1.4 WebSphere Adapters</td>
<td>12</td>
</tr>
<tr>
<td>1.4.1 Disable tracing and monitoring when possible</td>
<td>12</td>
</tr>
<tr>
<td>1.4.2 Move databases off the default Cloudscape</td>
<td>12</td>
</tr>
<tr>
<td>1.4.3 Configure the poll period and quantity</td>
<td>12</td>
</tr>
<tr>
<td>1.5 WebSphere Business Monitor</td>
<td>13</td>
</tr>
<tr>
<td>1.5.1 Configure CEI</td>
<td>14</td>
</tr>
<tr>
<td>1.5.2 Set the event processing batch size</td>
<td>14</td>
</tr>
<tr>
<td>1.5.3 Set connection pool sizes</td>
<td>15</td>
</tr>
<tr>
<td>1.6 Database: general</td>
<td>16</td>
</tr>
<tr>
<td>1.6.1 Use a production-quality database</td>
<td>16</td>
</tr>
<tr>
<td>1.6.2 Place database log files on a fast disk subsystem</td>
<td>16</td>
</tr>
<tr>
<td>1.6.3 Size database cross-referencing tables correctly</td>
<td>16</td>
</tr>
<tr>
<td>1.6.4 Place logs on a separate device from the table spaces</td>
<td>16</td>
</tr>
<tr>
<td>1.7 Database: DB2 specific</td>
<td>16</td>
</tr>
<tr>
<td>1.7.1 Maintain current indexes on tables</td>
<td>17</td>
</tr>
<tr>
<td>1.7.2 Update catalog statistics</td>
<td>17</td>
</tr>
<tr>
<td>1.7.3 Set the buffer pool size correctly</td>
<td>17</td>
</tr>
<tr>
<td>1.8 Java</td>
<td>17</td>
</tr>
<tr>
<td>1.8.1 Set heap and nursery size to handle garbage collection efficiently</td>
<td>18</td>
</tr>
<tr>
<td>1.8.2 Set AIX threading parameters</td>
<td>21</td>
</tr>
</tbody>
</table>
Notices

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

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

AIX®
Cloudscape™
DB2®
IBM®
Redbooks (logo)™
Redbooks™
WebSphere®

The following terms are trademarks of other companies:

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

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

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

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

This IBM® Redpaper was produced by the IBM WebSphere® Process Server, WebSphere Enterprise Service Bus, WebSphere Adapters, and WebSphere Business Monitor performance teams in Austin, Böblingen, and Hursley. It provides performance tuning tips and best practices based on the performance teams experience for the following products:

- WebSphere Process Server 6.0.1.1
- WebSphere Enterprise Service Bus (WebSphere ESB) 6.0.1.1
- WebSphere Integration Developer 6.0.0
- WebSphere Adapters 6.0.0.1
- WebSphere Business Monitor 6.0.0

WebSphere Process Server, WebSphere ESB, and WebSphere Integration Developer represent a new, integrated development and runtime environment, based on a key set of service-oriented architecture (SOA) technologies: Service Component Architecture (SCA), Service Data Object (SDO), and Business Process Execution Language for Web Services (BPEL). These technologies in turn build on the core capabilities of the WebSphere Application Server 6.0, including the Java™ Connector Architecture (JCA) V1.5 on which WebSphere Adapters v6.0.0.1 is based. WebSphere Business Monitor provides the ability to monitor business processes in real-time, providing a visual display of business process status, together with alerts and notifications to key users that enables continuous improvement of business processes.

For those who are either considering or are in the very early stages of implementing a solution incorporating these products, this document should prove a useful reference, both in terms of best practices during application development and deployment, and as a starting point for setup, tuning, and configuration information. It provides a useful introduction to many of the issues affecting these products’ performance, and could act as a guide for making rational first choices in terms of configuration and performance settings.

Finally, because all of these products build upon the capabilities of the WebSphere Application Server infrastructure, solutions based on these products will also benefit from existing tuning, configuration, and best practices information for WebSphere Application Server documented in “Related publications” on page 45. You are encouraged to use the information in this paper in conjunction with these references.

The team that wrote this Redpaper

This document was produced by the following members of the IBM WebSphere Process Server, WebSphere Enterprise Service Bus, WebSphere Adapters, and WebSphere Business Monitor performance teams in Austin, Böblingen, and Hursley.

IBM WebSphere Process Server and WebSphere Integration Developer performance team members:

Rajiv Arora
Michael P Ashley-Rollman
Mike Collins
Gudrun Eckl-Regenhardt
Ivan Eisen
Steve Garde
Jesse Gordon
Jonas Grundler  
Weiming Gu  
Steve Heuer  
Ben Hoflich  
Luc Maquil  
Thomas Muehlfriedel  
Gerhard Pfau  
Sefika Prcic  
Chris Richardson

IBM WebSphere Enterprise Service Bus performance team:  
Paul Harris  
Sam Massey  
Rachel Norris

IBM WebSphere Adapters performance team:  
Jimmy DeWitt  
Riaz Hussain

IBM WebSphere Business Monitor performance team:  
Bill Alexander  
Walter Fraser  
Kean Kuiper

**Become a published author**

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

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at: 
[ibm.com/redbooks/residencies.html](http://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 Redpaper or other IBM Redbooks™ in one of the following ways:

- Use the online Contact us review redbook form found at:
  ibm.com/redbooks

- Send your comments in an e-mail 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
Performance tuning and configuration

To optimize performance, it is usually necessary to configure the system differently than the default settings. This chapter lists several areas to consider during system tuning. This includes tuning the WebSphere Process Server, WebSphere Enterprise Service Bus (WebSphere ESB), WebSphere Integration Developer, WebSphere Adapters and Monitor products, and also other products in the system. As an example, WebSphere Process Server supports different database managers. The documentation for each of these products contains a wealth of information regarding performance, capacity planning, and configuration. This documentation would likely offer the best guidance for performance considerations in a variety of operational environments. Assuming that all these issues have been addressed from the perspective of the product, additional levels of performance implications are introduced at the interface between these products and the products covered in this Redpaper.

A number of configuration parameters are available to the system administrator. While 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 begins with a tuning checklist that enumerates the major components and their associated tuning concepts. The subsections that follow then address each in more detail, first describing performance-related design concepts, then discussing the tuning parameters and their suggested setting (where appropriate), and finally suggesting ways to determine potential settings for a particular configuration. While there is no guarantee that stepping through this checklist will provide acceptable performance immediately, it is likely that degraded performance can be expected if any of these parameters are incorrectly set.

Finally, the last section of this document contains references to related documentation that can prove valuable when tuning a particular configuration.
1.1 Tuning checklists

Table 1-1 contains a checklist of basic tuning actions for each product. We discuss these actions in more detail in later sections.

<table>
<thead>
<tr>
<th>Product</th>
<th>Tuning action</th>
</tr>
</thead>
<tbody>
<tr>
<td>WebSphere Process Server</td>
<td>Do not run production server in development mode</td>
</tr>
<tr>
<td></td>
<td>Disable tracing and monitoring when possible</td>
</tr>
<tr>
<td></td>
<td>Move databases off of the default Cloudscape</td>
</tr>
<tr>
<td></td>
<td>Configure the statement cache for long-running processes</td>
</tr>
<tr>
<td></td>
<td>Configure threads for messaging and in work managers</td>
</tr>
<tr>
<td></td>
<td>Use an appropriate Java heap size for production environments</td>
</tr>
<tr>
<td></td>
<td>Configure WebSphere Process Server for clustering</td>
</tr>
<tr>
<td>WebSphere ESB</td>
<td>Disable tracing and monitoring when possible</td>
</tr>
<tr>
<td></td>
<td>Move databases off the default Cloudscape</td>
</tr>
<tr>
<td></td>
<td>Set the WebContainer thread pool settings</td>
</tr>
<tr>
<td></td>
<td>Set the Java heap size</td>
</tr>
<tr>
<td></td>
<td>Increase the maximum concurrent endpoints in activation specifications</td>
</tr>
<tr>
<td></td>
<td>Increase maximum connection pool size in the queue connection factory</td>
</tr>
<tr>
<td></td>
<td>Set HTTP connectionKeepAlive</td>
</tr>
<tr>
<td></td>
<td>Set DiscardableDataBufferSize and CachedDataBufferSize</td>
</tr>
<tr>
<td></td>
<td>Set the DTMManager property</td>
</tr>
<tr>
<td>WebSphere Adapters</td>
<td>Disable tracing and monitoring when possible</td>
</tr>
<tr>
<td></td>
<td>Move databases off the default Cloudscape</td>
</tr>
<tr>
<td></td>
<td>Configure the poll period and quantity</td>
</tr>
<tr>
<td>WebSphere Business Monitor</td>
<td>Configure CEI</td>
</tr>
<tr>
<td></td>
<td>Set the event processing batch size</td>
</tr>
<tr>
<td></td>
<td>Set connection pool sizes</td>
</tr>
<tr>
<td>Database: general</td>
<td>Use a production-quality database</td>
</tr>
<tr>
<td></td>
<td>Place database log files on a fast disk subsystem</td>
</tr>
<tr>
<td></td>
<td>Size database cross-referencing tables correctly</td>
</tr>
<tr>
<td></td>
<td>Place logs on a separate device from the table spaces</td>
</tr>
<tr>
<td>Database: DB2 specific</td>
<td>Maintain current indexes on tables</td>
</tr>
<tr>
<td></td>
<td>Update catalog statistics</td>
</tr>
<tr>
<td></td>
<td>Set the buffer pool size correctly</td>
</tr>
<tr>
<td>Java</td>
<td>Set heap and nursery size to handle garbage collection efficiently</td>
</tr>
<tr>
<td></td>
<td>Set AIX threading parameters</td>
</tr>
<tr>
<td></td>
<td>Use HotSpot server instead of client</td>
</tr>
<tr>
<td></td>
<td>Adjust heap size if out-of-memory errors occur</td>
</tr>
<tr>
<td></td>
<td>Set the thread stack size if using many threads</td>
</tr>
</tbody>
</table>
1.2 WebSphere Process Server

This section discusses performance tuning tips for WebSphere Process Server. See 3.1, “WebSphere Process Server settings” on page 38 for a list of the values used during the WebSphere Process Server performance tests.

1.2.1 Do not run production server in development mode

WebSphere application servers can be run in development mode. This mode is intended to reduce startup time by using JVM™ settings to disable bytecode verification and reduce JIT compilation costs. This setting should not be used on production servers.

The setting can be found in the administrative console under:

Servers → Application Servers → <server_name> → Configuration

1.2.2 Disable tracing and monitoring when possible

The ability to configure tracing and monitoring at different levels for a variety of system components has proven to be extremely valuable during periods of system analysis or debugging. WebSphere Process Server provides rich monitoring capabilities, both in terms of business monitoring through CEI and audit logging, and system performance monitoring using Performance Monitoring Infrastructure (PMI) and Application Response Measurement (ARM). While these capabilities provide insight into the performance of the running solution, these features can degrade overall system performance and throughput. Therefore, it is recommended that tracing and monitoring be used judiciously and when possible, turned off entirely to ensure optimal performance.

Most tracing and monitoring is controlled through the administrative console. Validate that the appropriate level of tracing/monitoring is set for PMI Monitoring, Logging, and Tracing using the administrative console. Further, use the administrative console to validate that the Audit logging and Common Event Infrastructure logging check boxes are disabled in the Business Process container, unless these capabilities are required for business reasons.

WebSphere Integration Developer 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.

1.2.3 Move databases off of the default Cloudscape

WebSphere Process Server is packaged with the Cloudscape™ database, an open source database designed for ease-of-use and platform neutrality. When performance and reliability are critically important, use an industrial strength database such as IBM DB2® for any performance measurement or production installation. Examples of databases that can be moved to DB2 include the BPE database, relationship databases, and bus messaging engine data stores. The conversion requires the administrator to create new JDBC™ providers in the administrative console under Resources → JDBC Providers. When created, a data source can be added to connect to a database using the new provider.

A key data source property is the Maximum Connections under Connection pools additional properties. This should be set high enough so worker threads do not have to wait for a connection from the pool.
Move the messaging engine data stores to a high performance DBMS

The messaging engine data stores are automatically created during profile creation on Cloudscape without allowing any user choice. The following example demonstrates one method to create alternative data stores for the messaging engines on DB2.

After the Profile Creation Wizard has finished and Business Process Choreographer is configured, the system should contain four buses with one messaging engine each. The names will resemble those shown in Table 1-2.

<table>
<thead>
<tr>
<th>Table 1-2</th>
<th>Buses in WebSphere Process Server installed on machine box01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus</td>
<td>Messaging engine</td>
</tr>
<tr>
<td>SCA.SYSTEM.box01Node01Cell.Bus</td>
<td>box01-server1.SCA.SYSTEM.box01Node01Cell.Bus</td>
</tr>
<tr>
<td>SCA.APPLICATION. box01Node01Cell.Bus</td>
<td>box01-server1.SCA.APPLICATION. box01Node01Cell.Bus</td>
</tr>
<tr>
<td>CommonEventInfrastructure_Bus</td>
<td>box01-server1.CommonEventInfrastructure_Bus</td>
</tr>
<tr>
<td>BPC.box01Node01Cell.Bus</td>
<td>box01-server1.BPC.box01Node01Cell.Bus</td>
</tr>
</tbody>
</table>

Note: the node and cell names are the default

Each of these messaging engines is configured to use a data store in Cloudscape and each data store is located in its own database. When considering DB2, this is not optimal from an administrative point of view. There are already many databases in the system and adding four more databases increases the maintenance and tuning effort substantially. The solution proposed here uses a single DB2 database for all four data stores. The individual data stores (actually a set of tables) are completely separate and each messaging engine acquires an exclusive lock on its set of tables during startup. Each messaging engine uses a unique schema name to identify its set of tables.

Create the DB2 database and load the data store schemas

Instead of having a DB2 database per messaging engine, we put all messaging engines into the same database using several schemas to separate them. Table 1-3 shows the schema names used for the data store for each messaging engine.

<table>
<thead>
<tr>
<th>Table 1-3</th>
<th>New schema names</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td>Messaging Engine</td>
</tr>
<tr>
<td>SCASYS</td>
<td>box01-server1.SCA.SYSTEM.box01Node01Cell.Bus</td>
</tr>
<tr>
<td>SCAAPP</td>
<td>box01-server1.SCA.APPLICATION. box01Node01Cell.Bus</td>
</tr>
<tr>
<td>CEIMSG</td>
<td>box01-server1.CommonEventInfrastructure_Bus</td>
</tr>
<tr>
<td>BPCMSG</td>
<td>box01-server1.BPC.box01Node01Cell.Bus</td>
</tr>
</tbody>
</table>

To create one schema definition for each messaging engine, follow these steps:

1. Create one schema definition for each messaging engine with the following command on Microsoft® Windows®:

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

2. Repeat for each schema.
3. To be able to distribute the database across several disks, edit the created schema definitions and put each table in a tablespace named after the schema used. For example, SCAAPP becomes SCANODE_TS, CEIMSG becomes CEIMSG_TS and so on. The schema definition should look similar to Example 1-1 on page 5 after editing.

**Example 1-1  Commands to create the schema and table**

```sql
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 provide separate table spaces for the various tables here. Optimal distributions depend on application structure and load characteristics. In the example one tablespace per data store was used.

4. After having created 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 as fitting by issuing the following command for a system managed tablespace:

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

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

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

**Create 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.

Table 1-4 shows the default state.

**Table 1-4  Default JDBC provider settings for messaging engine data stores**

<table>
<thead>
<tr>
<th>Messaging engine</th>
<th>JDBC provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>box01-server1.SCA.SYSTEM.box01Node01Cell.Bus</td>
<td>Cloudscape JDBC Provider (XA)</td>
</tr>
<tr>
<td>box01-server1.SCA.APPLICATION.box01Node01Cell.Bus</td>
<td>Cloudscape JDBC Provider</td>
</tr>
<tr>
<td>box01-server1.CommonEventInfrastructure_Bus</td>
<td>Cloudscape JDBC Provider</td>
</tr>
<tr>
<td>box01-server1.BPC.box01Node01Cell.Bus</td>
<td>Cloudscape JDBC Provider</td>
</tr>
</tbody>
</table>
To create the data sources, follow these steps:

1. Create a new JDBC provider using the DB2 Universal JDBC Driver Provider for the non-XA data sources if you do not already have one. 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, the remaining three as single-phase commit (non-XA) data sources.

   Table 1-5 on page 6 provides new names and values for the data sources.

<table>
<thead>
<tr>
<th>Name of data source</th>
<th>JNDI Name</th>
<th>Type of jdbc provider</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEIMSG_sib</td>
<td>jdbc/sib/CEIMSG</td>
<td>DB2 Universal (XA)</td>
</tr>
<tr>
<td>SCAAPP_sib</td>
<td>jdbc/sib/SCAAPPLICATION</td>
<td>DB2 Universal</td>
</tr>
<tr>
<td>SCASYSTEM_sib</td>
<td>jdbc/sib/SCASYSTEM</td>
<td>DB2 Universal</td>
</tr>
<tr>
<td>BPCMSG_sib</td>
<td>jdbc/sib/BPCMSG</td>
<td>DB2 Universal</td>
</tr>
</tbody>
</table>

   When creating each data source:
   - Deselect the check box labeled Use this Data Source in container managed persistence (CMP).
   - Set a component-managed authentication alias.
   - Set the database name to the name used for the database created earlier for messaging (SIB).
   - Select a driver type of 2 or 4; both should work. A driver type of 4 requires a host name and valid port to be set on the database. (While we did not experiment with both types of drivers in our performance tests, we have seen reports of higher performance with the type 4 driver.)

**Change the messaging engine’s data stores**

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

1. For each messaging engine, navigate to Service Integration → Buses → <bus name> → Messaging engines → <ME name> → Data store and change the data store.

2. Put in the new JNDI and schema name for each data store. Deselect the Create Tables check box because the tables have been created already.

   The server immediately restarts the messaging engine. The Systemout.log shows the results of the change and will also indicate if the messaging engine comes up successfully.

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

The last remaining task is to tune the database, using the DB2 Configuration Wizard as a starting point and collecting statistics after the tables are populated. Also check the relevant database parameters, such as the number of connections that are needed to support the databases for the messaging engines.
1.2.4 Configure the statement cache for long-running processes

BPEL macroflows (long-running processes) make extensive use of the BPEDB data source for persisting data relevant to the process. This results in the usage of many different statements, far more than the default capacity of the data source’s cache. This results in an excessive number of cache misses, making the caching ineffective. This problem can be resolved by increasing the size of the cache.

To increase the size of the cache, perform the following steps using the administrative console:

1. Select **Resources → JDBC Providers**.
2. Select the scope that contains your BPEDB data source (server, by default).
3. Click the name of the JDBC provider that supports the BPEDB data source. This is Cloudscape JDBC Provider (XA) by default.
4. Click **Data Sources** under Additional Properties.
5. Click the name of the data source with the `jdbc/BPEDB` JNDI name.
6. Click **WebSphere Application Server data source properties** under Additional Properties.
7. Increase the statement cache size. The recommended cache size is 128.
8. Save your changes and restart your server.

1.2.5 Configure threads for messaging and in work managers

The Platform Messaging Component SPI Resource Adapter is created as part of an application install. It provides an EIS with the ability to communicate with message driven beans configured on the server. Message processing for an application uses properties defined under an activation specification for this Resource Adapter. The `maxConcurrency` custom property under the J2C activation specification for the resource adapter is used to specify the number of threads available to process messages by the application.

If a work manager is used set the maximum number of threads setting to a value high enough to prevent the thread pool from running out of threads. This can be set in the administrative console under **Asynchronous beans → work managers**.

One symptom of insufficient concurrency will be CPU idleness. Vary the concurrency to achieve maximum CPU utilization and throughput.

1.2.6 Use an appropriate Java heap size for production environments

The initial heap size and maximum heap size for an application server JVM can be set by navigating to the following menu options:

**Servers → Application servers → <server_name> → Server Infrastructure → Java and Process Management → Process Definition → Additional Properties → Java Virtual Machine.**

Factors influencing the appropriate settings are discussed in 1.8, “Java” on page 17.
1.2.7 Configure WebSphere Process Server for clustering

When tuning to achieve the best possible horizontal scalability through clustering, additional WebSphere Process Server configuration is required.

Configure activation specification properties

Each SCA module defines a message-driven bean (MDB) and its corresponding activation specification. The default value for maxConcurrency of the SCA module MDB is 10, meaning only up to 10 asynchronous SCA requests in the module can be processed concurrently. If the server CPU is not maxed out, it sometimes is caused by this setting being too low; it needs to be increased.

2. Select the activation specification for the SCA module.
3. Click J2C activation specification custom properties.
4. Change maxConcurrency to a higher value, say 20.

Configure ORB thread pool

This configuration parameter is relevant if the cluster is driven by a driver node through the SCA synchronous binding.

Due to the interaction between synchronous SCA, workload management (WLM), and the ORB, the ORB thread pool size on the cluster nodes needs to be configured to maximize the clustering throughput. The rule of the thumb is to use the same number of ORB threads on all application nodes, and have the total number of ORB threads across all application nodes be the same as the number of driver threads on the driver node. For example, if the driver uses 120 concurrent threads, the ORB thread pool size on each application node on a 6-node cluster should be 20.

The default value of ORB thread is 50. To set this value, navigate to Servers → Application servers → <server_name> → Container Services → ORB Service → Thread Pool. Complete the Maximum Size field.

Configure relationship and BPE data source connection pools

The maximum connections property of the relationship data source should be large enough to allow concurrent access to the database from all threads. For the clustered measurements in the test lab, this property was set to 2 times the largest value of the WebContainer, ORB, or the default thread pool size. The multiplier of 2 comes from the fact that each transaction in the test run could contain two relationship database accesses. If your application contains more or less relationship database access, you need to adjust the multiplier accordingly.

The maximum connections property of the BPE data source should be large enough to allow concurrent access to the database from all threads. It should be larger than the largest value of the WebContainer, ORB, or the default thread pool size.

To set maximum connections for a data source navigate to Resources → JDBC Providers → <JDBC provider name> → Data sources → <Data source name> → Connection pool properties. Complete the Maximum connections field.

To find the Web container thread pool size navigate to Servers → Application servers → <server_name> → Thread Pools → WebContainer. The value is in the Maximum Size field.
The maximum connections for the ORB service can be found by navigating to **Servers → Application servers → `<server_name>` → Container Services → ORB Service → Thread Pool**. The value is in the Maximum Size field.

To find the default thread pool size, navigate to **Servers → Application servers → `<server_name>` → Thread Pools → Default**. The value is in the Maximum Size field.

**Configure IBM HTTP server**

On the driver node, set the `com.ibm.websphere.webservices.http.maxConnection` system property to 200. The default is 50. In the performance tests in a clustered environment, more than 50 driver threads were needed to max out the CPU on six cluster members.

In `httpd.conf` for the IBM HTTP server, set `MaxSpareThreads` to 200 (default is 75). This enables more than 75 driver threads, which was needed for the clustered workload performance tests using Web services to run without errors.

### 1.3 WebSphere ESB

This section discusses configuration options that are relevant to tuning WebSphere ESB. See 3.2, “WebSphere ESB settings” on page 40 for a list of the values used during the WebSphere ESB performance tests.

#### 1.3.1 Disable tracing and monitoring when possible

While the ability to configure tracing and monitoring at different levels for a variety of system components can be useful, they will degrade overall system performance and throughput. Therefore, it is recommended that tracing and monitoring be turned off entirely to ensure optimal performance. Tracing and monitoring is controlled through the administrative console.

To disable tracing, follow these steps:

1. Select to **Troubleshooting → Logs and Trace**.
2. Click the server name.
3. Select **Change Log Detail Levels**.
4. Set both the detail level in both the Configuration and Runtime panels to `*=all=disabled`.

To change the monitoring level, follow these steps:

1. Select **Monitoring and Tuning → Performance Monitoring Infrastructure**.
2. Click the server name.
3. Select **none**.

#### 1.3.2 Move databases off the default Cloudscape

WebSphere ESB by default uses the Cloudscape database, an open source database designed for ease-of-use and platform neutrality. For production deployments use an industrial strength database such as IBM DB2 when performance and reliability are critically important. Performance is further improved if the DB2 database is located on a remote machine.

Changing databases requires the administrator to create new JDBC providers in the administrative console. You can do this by navigating to **Resources → JDBC Providers**. When created, a data source can be added to connect to a database using the new provider.
1.3.3 Set the WebContainer thread pool settings

The WebContainer thread pool sizes and timeouts need to be set to optimum values.

WebContainer thread pool sizes and timeouts can be set by navigating to **Servers → Application servers**. Click the server name and under Additional Properties select **Thread Pools → WebContainer**.

1.3.4 Set the Java heap size

The **Initial Heap Size** and **Maximum Heap Size** settings for an application server JVM can be set under **Servers → Application servers**. Click the server name and select **Server Infrastructure → Java and Process Management → Process Definition**. Under Additional Properties, select **Java Virtual Machine**.

1.3.5 Increase the maximum concurrent endpoints in activation specifications

Increase the maximum concurrent endpoints. This is the maximum number of endpoints to which messages are delivered concurrently. Increasing this number can improve performance, but can increase the number of threads that are in use at any one time. If message ordering must be retained across failed deliveries, set the maximum concurrent endpoints to 1.

The maximum concurrent endpoints setting can be set under **Resources → JMS Providers → Default messaging → JMS activation specification → <activation spec>**.

1.3.6 Increase maximum connection pool size in the queue connection factory

Increase the maximum connection pool size to allow for greater concurrency. It should be noted that if only one deployed application is obtaining connections from a particular connection factory, there would be no benefit in increasing the maximum queue connection pool size beyond the maximum concurrency as configured within the activation specification.

The maximum connection pool size specifies the maximum number of physical connections that you can create in this pool. These are the physical connections to the back-end resource. Once this number is reached, no new physical connections are created and the requester waits until a physical connection that is currently in use returns to the pool, or a ConnectionWaitTimeout exception is issued.

For example, if the maximum connections setting is set to 5, and there are five physical connections in use, the pool manager waits for the amount of time specified in the connection timeout setting for a physical connection to become free.

If the maximum connections setting is set to 0, the connection pool is allowed to grow infinitely. This also has the side effect of causing the connection timeout setting to be ignored.

These settings can be found by navigating to the following location:

**Resources → JMS Providers → Default messaging → JMS queue connection factory → <qcf_name> → Additional properties → Connection pool properties.**
1.3.7 Set HTTP connectionKeepAlive

The connectionKeepAlive property specifies whether the connector should maintain a live or persistent HTTP connection. If the property is set to true, the connector keeps the connection in the connection pool and reuses the connection for subsequent HTTP requests. If the property is set to false, the connection is closed after the HTTP request is sent. If a new request is ready to send and the connection does not exist, the HTTP connector creates one.

The connection Keep Alive property can be created and set by navigating to the following location:


1.3.8 Set DiscardableDataBufferSize and CachedDataBufferSize

The DiscardableDataBufferSize setting is the size in bytes of the data buffer used when processing best effort nonpersistent messages. The purpose of the discardable data buffer is to hold message data in memory because it is never written to the data store for this quality of service. Messages which are too large to fit into this buffer will be discarded.

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

The DiscardableDataBufferSize and CachedDataBufferSize settings can be created by navigating to:

Service Integration → Buses → <bus_name> → Messaging Engines → <messaging engine> → Custom properties.

1.3.9 Set the DTMMManager property

Setting the DTMMManager property in the xalan.properties file prevents repeated searching of libraries to lookup the DTMMManager class. This property is currently searched for after each and every XSLT Transformation primitive, but it is in third party code and there is no programmatic change that can be made to prevent this. It should only be set if it can be verified that no other DTMMManager class is used within applications deployed on the WebSphere ESB server.

To set the property, add the following line to the xalan.properties file in the java\jre\lib directory of your WebSphere ESB installation:

org.apache.xml.dtm.DTMMManager=org.apache.xml.dtm.ref.DTMMManagerDefault

1.3.10 DB2 tuning

If you are using persistent messaging the configuration of your database becomes important. Use a remote DB2 instance with a fast disk array as the database server. You may also find benefit in tuning the connection pooling and statement cache of the data source. See 1.7, “Database: DB2 specific” on page 16 for further information about database tuning.
1.4 WebSphere Adapters

This section discusses performance tuning tips for WebSphere Adapters.

1.4.1 Disable tracing and monitoring when possible

The ability to configure tracing and monitoring at different levels for a variety of system components has proven to be extremely valuable during periods of system analysis or debugging. While these capabilities provide insight into the performance of the running solution, these features can degrade overall system performance and throughput. Therefore, it is recommended that tracing and monitoring be used judiciously and, when possible, turned off entirely to ensure optimal performance.

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

1.4.2 Move databases off the default Cloudscape

WebSphere Adapters by default use the Cloudscape database, an open source database designed for ease-of-use and platform neutrality. When performance and reliability are critically important, use an industrial-strength database such as IBM DB2 for any performance measurement or production installation. First validate that your adapter supports DB2; as of this writing some adapters do not support DB2 (for example, the WebSphere Flat File Adapter).

Examples of databases that can be moved to DB2 include the Event Delivery Table, adapter-specific databases, and messaging engine data stores. The conversion requires the administrator to create new JDBC providers in the administrative console under Resources → JDBC Providers. Once created, a data source can be added to connect to a database using the new provider.

A key data source property is the maximum connections settings under the connection pool additional properties. This should be set high enough so worker threads do not have to wait for a connection from the pool.

1.4.3 Configure the poll period and quantity

Two of the most important configuration parameters for the WebSphere Adapters are PollPeriod and PollQuantity. These parameters can be set on the J2C activation specification.

- PollPeriod specifies the amount of time (in milliseconds) between polling actions.
- PollQuantity specifies the maximum number of events to process during a polling action.

Because these parameters control the rate and amount of work that an adapter will process, the combination of poll period and poll quantity regulate the number of transactions that are processed first by the adapter, and then by the broker (WebSphere Process Server). As such, these parameters influence the performance of the entire solution, not just the adapter. Nonoptimal values for these parameters can result in either low system throughput (if the poll period is too long, poll quantity is too low, or both), or can cause excessive memory usage (and potentially OutOfMemory exceptions) if the parameters are configured to deliver events to the system at rates that are higher than the solution is implemented to handle (if the poll period is too short, poll quantity is too high, or both). Because both of these conditions will
impact overall system performance dramatically, appropriate values for the \texttt{PollPeriod} and \texttt{PollQuantity} settings are critical, and should be configured explicitly to support the level of throughput a solution is designed to handle.

In general, the recommendation is to configure \texttt{PollPeriod} and \texttt{PollQuantity} to enable events to be retrieved and processed at a level that matches the peak throughput of the overall solution.

As an example, if the peak throughput rate of a solution is 20 events per second, and events are continually available to the adapter for processing, set \texttt{PollPeriod} to some small value (perhaps 10 milliseconds) and set \texttt{PollQuantity} to 20. This supports the required peak throughput, while requiring a relatively small number of events to be concurrently held in the adapter process. The poll period enables a minimal delay between poll cycles. Factors that can require an adjustment to these values include:

- The size of the object being processed
  For larger objects, a good rule of thumb is to use a lower poll quantity and longer poll period. This rule does not generally apply for relatively small objects (100 KB or less). However, for larger objects it is important to minimize the number of objects held concurrently in the adapter process to avoid potential OutOfMemory exceptions. To extend this example, if the object size is 1 MB and the throughput rate of the solution is 2 events per second, appropriate settings could be \texttt{PollPeriod} = 100 milliseconds and \texttt{PollQuantity} = 2.

- The Java heap size and physical memory available on the system
  In general, the larger the heap, the higher \texttt{PollQuantity} can be set. However, there are several factors involved in setting the heap size, one very important factor being to ensure that the heap is not set so large that paging results. Paging the Java heap will dramatically reduce system performance. See the 1.8, “Java” on page 17 for a detailed discussion on setting the Java heap sizes appropriately.

- The uniformity of event arrival rates
  The examples above assume that events arrive at a relatively constant rate. This might not be true for many solutions; event arrival is sometimes very uneven. In these cases, care must be taken to balance processing events in a timely manner in order to handle the occasional high arrival rates, while also not holding too many events in memory concurrently and, potentially, encountering OutOfMemory exceptions.

1.5 WebSphere Business Monitor

This section discusses performance tuning tips relevant for WebSphere Business Monitor. Please see 3.4, “WebSphere Business Monitor settings” on page 42 for a list of the values used during the WebSphere Business Monitor performance tests.

Figure 1-1 on page 14 shows an example of a WebSphere Business Monitor installation. Business processes emit BPEL events and place them on a JMS destination queue registered with CEI. Monitor server takes events off this destination queue, processes them, and updates the State database. DB2’s Replication Manager runs periodically and copies newly arrived events into the Runtime database. Replication manager also periodically updates the historical database which maintains DB2 CubeViews to enable data mining. Customers can direct queries to either the runtime or historical databases from a browser using Monitor Dashboards.
1.5.1 Configure CEI

By default, when an event arrives at CEI, it is put into the queue you have specified (in this case into a queue the Monitor Launchpad specified) and into an additional, default queue. Unless the customer needs this extra queue, do not double-store.

You can prevent this extra store by performing the following steps in the administrative console:

1. Set the scope to Cell.
2. Navigate to Resources → Common Event Infrastructure Provider → Event Group Profile List → Event groups list → Event Group Profile.
3. Remove All Events.

Even though the CEI queue is persistent, CEI offers the ability to store events in a database. This is expensive, and unless the customer needs this extra copy, do not save it.

You can disable the CEI data store by following these steps:

1. Set the scope to Cell.
3. Deselect Enable Data Store.

1.5.2 Set the event processing batch size

Taking events off the CEI queue in large batches is much more efficient than one at a time. Up to some limit, the larger the batch size, the higher event processing throughput will be. But there is a trade-off: Taking some events off the CEI queue, processing them, and persisting
them to the STATE database is done as a transaction. So while a larger batch size yields better throughput, it will cost more if you have to roll back.

Monitor Launchpad sets the batch size to 100 events. If you experience frequent rollbacks, consider reducing the batch size.

Reducing the batch size can be done in the administrative console:
1. Set the scope to the server.
3. Set the Event Processing Queue Size: <default 100>.

Monitor processes events most efficiently when the Deserialization WorkManager work request queue is the same size as the event processing queue; if you change one you should change both.

In the administrative console, follow these steps:
1. Set the scope to the server.
3. Set the following values:
   – Work request queue size: <same as batch size>
   – Check that growable is not selected.

### 1.5.3 Set connection pool sizes

The data sources in both Monitor Dashboard and WebSphere Portal Server use pools of connections to communicate with DB2. It is important that these pools be large enough that threads do not have to wait for a connector.

There are three data sources in Dashboard: HistoricalDBDataSource, RepositoryDBDataSource and RuntimeDBDataSource, and two in Portal Server: wmmDS and wpsdbDS. By default WebSphere sets connection pool sizes to 10, but Monitor Launchpad initializes these five pools to 100, which we have found sufficient for the workloads described in this report. However, if, under heavy Dashboard load, there are periods when nothing appears to be happening, try increasing these pool sizes.

For example, to set the maximum connection pool size for HistoricalDBDataSource using the administrative console:
1. Set the scope to Cell.
2. Navigate to Resources → JDBC providers → MonitorDB2XADriver1 → Data sources → HistoricalDBDataSource → Connection pools.

Chapter 1. Performance tuning and configuration
1.6 Database: general

This section discusses performance tuning tips for databases in general. See 3.3, “Database manager common settings” on page 41 for a list of the values used during the performance tests.

1.6.1 Use a production-quality database

WebSphere Process Server is packaged with the Cloudscape database, an open source database designed for ease-of-use and platform neutrality. When performance and reliability are critically important, use an industrial-strength database, such as IBM DB2 for any performance measurement or production installation.

1.6.2 Place database log files on a fast disk subsystem

The database log files should be placed on a fast disk subsystem with write-back cache. This will have a direct bearing on the database performance.

1.6.3 Size database cross-referencing tables correctly

Relationship cross-reference tables for identity relationships and other dynamic relationships grow continuously, and can become quite large in a production environment. Carefully size these tables accordingly. It is recommended that the first extent of the table contains as many rows as possible, even the entire table. This will avoid extent interleaving and enable faster database service times on relationship lookups.

Currently there is no mechanism to specify extent sizes and physical storage attributes for these tables through the administrative console. Therefore, if these tables are expected to be large, the recommended method to achieve this is to export the data in these tables, drop them, recreate them according to the desired storage parameters, and then reload the data.

Determining which tables in the WebSphere Process Server database are the cross reference tables is straightforward. Each relationship participant has a cross reference table dedicated to itself. For example, if there is a relationship definition for a dynamic relationship named Customer with participants SAP, Clarify, and PeopleSoft, there are three distinct tables in the database for each participant.

1.6.4 Place logs on a separate device from the table spaces

A basic strategy for all database storage configurations is to place the database logs on separate devices from the tablespace containers. This prevents I/O to tablespace containers from contending with the I/O to the database logs and improves recoverability.

1.7 Database: DB2 specific

Providing a comprehensive DB2 tuning guide is beyond the scope of this report. On the other hand, there are a few general rules of thumb that can assist in improving the performance of DB2 environments. In the next paragraphs, we discuss these rules, and provide pointers to more detailed information. A quick reference for DB2 performance tuning can be found here:

1.7.1 Maintain current indexes on tables

While the WebSphere Business Integration products create a set of database indexes that are appropriate for many installations, additional indexes may be required in some circumstances. Often, a database environment that requires additional indexes will exhibit performance degradation over time; in some cases the performance degradation can be profound. Environments that need additional indexes often exhibit heavy read I/O on devices holding the tablespace containers. To assist in determining which additional indexes could improve performance, DB2 provides the Design Advisor. The Design Advisor is available from the DB2 Control Center, or can be started from a command line processor. It has the capability to help define and design indexes suitable for a particular workload.

1.7.2 Update catalog statistics

It is important to update the DB2 catalog statistics on a regular basis. These are statistics that are used by the DB2 query optimizer to determine the access plan for evaluating a query. Statistics are maintained on tables and indexes. Examples of statistics include the number of rows in a table, and the number of distinct values in a certain column of a table. These statistics are not maintained by DB2 in real-time; statistics are updated by DB2 commands that are typically run by the DBA. If statistics are not updated on a regular basis, the DB2 query optimizer might create poor-performing access plans for evaluating queries. The following command can be used to update statistics on all tables in the database:

```
db2 -v reorgchk update statistics on table all
```

More information about maintaining catalog statistics can be found in this publication:

- **DB2 Administration Guide: Performance**
  

1.7.3 Set the buffer pool size correctly

While there are many DB2 configuration parameters, one parameter that is critical to performance, and often requires modification in a new installation, is the buffer pool size (BUFFPAGE parameter from the database configuration). In general, the default value of this parameter is too small to run efficiently. Alternatively, the parameter should be set small enough so that the database buffer pool can coexist along with other structures and applications, without exhausting the real memory on the system.

1.8 Java

Because the WebSphere Business Integration product set is written in Java, the performance of the Java Virtual Machine (JVM) has a significant impact on the performance delivered by these products. JVMs externalize multiple tuning parameters that can be used to improve both tooling and runtime performance. The most important of these are related to garbage collection, setting the Java heap size, and configuring threading parameters. This section discusses these topics in detail.

Note that the products covered in this paper use IBM JVMs on most platforms (AIX®, Linux®, Microsoft Windows, and so forth), and the HotSpot JVMs on Solaris™ and HP systems. Vendor-specific JVM implementation details and settings are discussed as appropriate.

While there are more tuning parameters than those discussed in this section, most are for specific situations and are not for general use.
For more information, see the following publications:

- For a useful summary of HotSpot JVM options for Solaris:
  http://java.sun.com/docs/hotspot/VMOptions.html
- For a useful FAQ about the Solaris HotSpot JVM:
  http://java.sun.com/docs/hotspot/PerformanceFAQ.html#20

1.8.1 Set heap and nursery size to handle garbage collection efficiently

Garbage collection (GC) is the process of freeing unused objects so that portions of the JVM heap can be reused. Because the Java language specification does not provide explicit delete() or free() byte codes, it is imperative to occasionally detect and delete objects which no longer have active references and free that space for reuse.

Garbage collection is triggered automatically when there is a request for memory, for example, object creation, and the request cannot be readily satisfied from the free memory available in the heap (allocation failure). Garbage collection can also be activated programmatically using a Java class library System.gc() call. In this case, garbage collection occurs immediately and synchronously.

While the function provided by the HotSpot and IBM garbage collectors is the same, the underlying technology is different. For both JVMs garbage collection takes place in three phases: mark, sweep, and an optional compact phase. The implementation of the garbage collection phases is very different. This is mainly due to the fact that the HotSpot engine is what is known as a generational collector while the IBM JVM (by default) is not.

A detailed discussion of the HotSpot generational GC can be found at the following URL:


In its default configuration, the IBM JVM consumes its entire heap before a garbage collection is triggered. With the HotSpot JVM a garbage collection is triggered when either the nursery or the full heap is consumed. Whether a full heap GC or Nursery GC is being performed the first phase is to mark all referenced objects in the region being collected. This leaves all unreferenced objects unmarked and the space they occupy free to be collected and reused. Following the mark-phase free chunks of memory are added to a free list. This phase is referred to as sweeping.

Occasionally following the sweep phase a compact phase is performed. The compaction moves objects closer together to create larger contiguous free chunks. There are a number of triggers that can cause a compaction. For instance, if after sweep there is still not a large enough contiguous chunk of memory, then compaction executes. Also, for most System.gc() calls a compaction is done. Relative to the other phases involved, compaction can be a time consuming process and should be avoided if possible. The IBM JVM has been optimized to avoid compactions, as this is an expensive process.

Monitoring garbage collection

To set the heap correctly, you must first determine how the heap is being used. This is easily done by collecting a verbosegc trace. A verbosegc trace prints garbage collection actions and statistics to stderr. The verbosegc trace is activated by using the Java run-time option of verbose:gc. Output from verbosegc is different for the HotSpot and IBM JVMs, as shown by the following examples.
Example 1-2 is a sample IBM JVM verbosegc trace output.

**Example 1-2  JVM verbosegc trace output**

```
<AF[8]: Allocation Failure. need 1572744 bytes 5875 ms since last AF>
<AF[8]: managing allocation failure, action=1 (23393256)/131070968)
(2096880/3145728)>
<GC: Tue Dec 18 17:32:26 2001
<GC(12): freed 75350432 bytes in 168 ms, 75% free (100840568)/134216696)>
<GC(12): mark: 129 ms, sweep: 39 ms, compact: 0 ms>
<GC(12): refs: soft 0 (age >= 32), weak 0, final 0, phantom 0>
<AF[8]: completed in 203 ms>
```

This is an example of a 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]
```

Using the IBM JVM output shown in Example 1-2 on page 19, the metric following the word `need` is the size of the failed allocation that caused the garbage collection. On the same line, the amount of time in milliseconds since the last allocation failure is given. The next line with the `<AF[8]>` tag displays the amount of free space in the heap and in an area of the heap referred to as the wilderness. The line reports 23393256 free bytes out of a possibly 131070968 bytes. The (2096880/3145728) refers to wilderness area free, which is usually ignored.

The next set of lines provides information about the garbage collection that was caused to satisfy the allocation failure. The first line is a time stamp. This is followed by a line that includes the time to complete the GC, 168 ms, and the amount of free space after the GC, 75%. Both of these metrics are extremely useful in understanding the efficiency of the garbage collection and the heap usage. Following this line is a line describing the time for the different components of the GC. You should look to make sure that the number following compact is normally 0. That is, a well-tuned heap will avoid compactions. Finally, for the GC, there is a line about soft, weak, and phantom references, as well as a count of finalizers. This is then bracketed by a line with a time for the full allocation failure.

**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 runs concurrently on the same system (for example, if you run both WebSphere Process Server and WebSphere Integration Developer on the same system), then you should also read “Setting the heap size when running multiple JVMs on one system” on page 20.

For many applications, the default heap size setting for the IBM JVM is sufficient in order to get good performance. In general, the HotSpot JVM default heap and nursery size is too small and should be increased. For optimal performance and for applications with unpredictable loads or large live sets (a large number of Java objects that are active concurrently), the heap size should be optimized.

There are several approaches to setting the optimal heap sizes. We describe here the approach that most applications should use when running the IBM JVM on AIX. The essentials can be applied to other systems. First, we provide more background. There is a feature in the IBM JVMs that deals with dynamically growing the heap that is referred to as **rate-trigger heap growth**. This process attempts to set the size of the heap so that pauses...
are not too long and GC does not take too much time. This is done dynamically, and it adjusts with the workload. If too much time is being used in GC, the heap grows. If the heap is mostly free, the heap can shrink.

To use rate-trigger heap growth effectively, set the initial heap size (-ms) option to something reasonable (for example, 256 MB), and the maximum heap size (-mx) option to something reasonable, but large (for example, 1024 MB). Of course, the maximum heap size should never force the heap to page. It is imperative that the heap always stays in physical memory. The JVM will then try to keep the GC time to something reasonable behind the covers by growing and shrinking the heap. The output from verbosegc should then be used to monitor the GC actions.

A similar process can be used to set the size of HotSpot heaps. In addition to setting the minimum and maximum heap size, you should also increase the nursery size to approximately 1/4 of the heap size. Note that you should never increase the nursery to more than 1/2 the full heap. The nursery size is set using the MaxNewSize and NewSize parameters (that is, XX:MaxNewSize=128m, XX:NewSize=128m).

After the heap sizes are set, verbosegc traces should then be used to monitor the GC actions. If the user finds something unpleasant from the verbosegc trace, they can 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, throughput can be improved 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. Note that increasing the maximum size of the Java heap might not always solve this type of problem as it is could be a memory over-usage problem. If the pause time is too long, decrease the heap size. If both problems are observed, an analysis of the application heap usage is required.

Setting the heap size when running multiple JVMs on one system

Each running Java program has a heap associated with it. Therefore, 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. An example of one such configuration is when WebSphere Integration Developer is on the same physical system as WebSphere Process Server. Each of these 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 as well as all other virtual memory allocations) exceeds the size of physical memory, the Java heaps will be subject to paging. As previously noted, this causes total system performance to degrade significantly. To minimize the possibility of this occurring, use the following guidelines:

- 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. Based on this, set the initial heap size to 256 MB (ms256m).
- Based on the verbosegc trace output, set the maximum heap size appropriately. Care must be taken to not set this value too low, or Out Of Memory errors will occur; the maximum heap size must be large enough to allow for peak throughput. Using the above example, a maximum heap size of 768 MB might be appropriate (mx768m).
- Be careful to not set the heap sizes too low, or garbage collections will occur frequently, which might reduce throughput. Again, a verbosegc trace will assist in determining this. A balance must be struck so that the heap sizes are large enough that garbage collections do not occur too often, while still ensuring that the heap sizes are not cumulatively so large as to cause the heap to page. This balancing act will, of course, be configuration-dependent.
Summary of setting heap sizes
The following list summarizes our recommendations about setting heap sizes:

- Make sure that the heap never pages (that is, on a given system, the sum of all the JVM's maximum heap sizes must fit in physical memory).
- Collect and analyze a verbosegc trace in order to optimize memory usage.
- Aim for less than 10% execution time being spent in garbage collection. Analyze the verbosegc trace in order to determine the GC execution time. Object reuse and heap size tuning can help in this area.
- For optimal performance, the heap should be run with less than 60%, possibly even 50%, occupancy. This is readily determined from the verbosegc trace output.
- Avoid finalizers: A developer can never be guaranteed when a finalizer will run, and often they lead to problems. If you do use finalizers, try to avoid allocating objects in the finalizer code. An IBM JVM verbosegc trace shows if finalizers are being called.
- Avoid compaction where possible. A verbosegc trace will show if compaction is occurring.
- Analyze requests for large memory allocations and then devise a method for reusing the object.
- Increase the size of the nursery for the Sun™ HotSpot JVM. A good rule is to set the nursery to one quarter (1/4) the size of the heap.

1.8.2 Set AIX threading parameters
The IBM JVM threading and synchronization components are based upon the AIX POSIX compliant Pthread implementation. The following environments variables have been found to improve Java performance in many situations and have been used for the workloads in this document. The variables control the mapping of Java threads to AIX native threads, turn off mapping information, and allow for spinning on mutex locks.

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

More information about AIX specific Java tuning can be found at the following Web sites:

- Running your Java application on AIX, Part 1: Getting started
- Running your Java application on AIX, Part 2: JVM memory models

1.8.3 Use HotSpot server instead of client
The HotSpot JVM can be configured to run as a server or as a client. When configured as a server, the Just-In-Time Compiler (JIT) uses extra processor cycles and memory to create more highly optimized code. Because the server is a long running process, the extra time and memory spent JITting at initial instantiation is well worth the increased performance during run time.

Therefore, on platforms that ship with the HotSpot JVM, adapters and WebSphere Process Server should always be run in server mode. To do this, the server parameter is added to the Java invocation.
1.8.4 Adjust heap size if out-of-memory errors occur

The java.lang.OutOfMemory exception is used by the JVM in a variety of 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, then this is easily seen in this output. See “Monitoring garbage collection” on page 18 for further information about verbosegc output. Many garbage collections resulting in very little free heap space will occur preceding the exception. If this is the problem, then you should increase the size of the heap.

If, however, there is enough free memory when the java.lang.OutOfMemory exception is thrown, the next item to check is the finalized count from the verbosegc (only the IBM JVM will give this information). If these appear high, then 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.

Note that Out Of Memory errors can also occur for issues unrelated to JVM heap usage, such as running out of certain system resources. Examples of this include insufficient file handles or thread stack sizes that are too small.
Best practices

This chapter provides guidance on how to develop and deploy WebSphere Process Server and WebSphere ESB solutions for high performance. This guidance is based on the current implementation of the WebSphere Process Server and WebSphere ESB runtimes and WebSphere Integration Developer tooling and will very likely evolve as these products evolve and mature. While many of these best practices were developed as a result of the performance testing in the lab, we recognize that they are not always applicable or feasible. You should consider these practices as guidelines when developing your own new solutions.

The purpose of this chapter is to highlight the best practices associated specifically with the new technologies and features delivered in the products covered in this report. However, these products are built on top of existing technologies such as WebSphere Application Server, Service integration technology, and the Business Process Choreographer. Each of these technologies has associated best practices that still apply. It is not our intent to enumerate these here. Instead, refer to “Related publications” on page 45 for a set of references and pointers to this information."
2.1 Tooling adjustments

This section discusses settings that can enhance performance in WebSphere Integration Developer.

2.1.1 Disable auto-publish

The auto-publish feature in WebSphere Integration Developer enables automatic publication of generated workspace artifacts to a running WebSphere Process Server instance. These publication events are triggered on every change to the artifacts. In many cases, this might not be the behavior that the WebSphere Integration Developer developer desires. For instance, it is common in the course of workload development to produce an inconsistent workspace (consider the incremental implementation of several dependent changes). Error detection at build and publish time is usually expensive. Many developers find that they prefer to initiate publish events manually, when the workspace and server are both in a well-understood state.

2.1.2 Consider disabling auto-build

The auto-build feature of WebSphere Integration Developer enables automatic generation of workspace artifacts whenever a change is saved to the source elements upon which they depend. Some developers find that they enjoy this feature and the automatic discovery of workspace inconsistencies and dependency issues that it can help detect. There is a performance cost for this build and error detection capability, which usually presents itself asynchronously to the development work underway (a build cycle consuming several minutes may start as the developer finishes one task and is beginning another, resulting in processor and memory consumption in the background). Many developers prefer to disable the auto-build feature and initiate build events manually.

2.1.3 Monitor memory utilization

Java heap memory utilization within WebSphere Integration Developer depends upon the size and complexity of the workspace under development. As with any Java process it is advisable to monitor heap statistics with the verbosegc command line option. WebSphere Integration Developer users, especially those experiencing memory or response time problems, should consider expanding the amount of memory available when building complex projects using the -Xms and -Xmx command line options. Java command line options can be enabled for WebSphere Integration Developer with the wid.ini file.

2.2 Modeling considerations

This section discusses design issues to consider during application development.

2.2.1 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.
2.2.2 Avoid overuse of WebSphere Process Server components

WebSphere Process Server components are services and as such, are inherently more expensive to locate and invoke than say, method invocations. When developing solutions, aim for the coarsest granularity to minimize the service invocation overhead.

2.2.3 Use the simplest component type for the required function

WebSphere Process Server provides valuable functional building blocks that reduce the complexity of creating business logic. Features like business rules, selectors, and business state machines are examples of this value. However, as in any programming system, value comes at a cost, and the developer should be mindful of the performance tradeoffs. All else being equal, a simple Java condition in a Java component or a Java snippet will be more efficient than invoking a business rule. However, business rules provide a declarative and expandable way to specify business decisions, the benefits of which can override the performance overhead. Such trade-offs are a natural part of any solution development and should be made here as well.

2.2.4 Reduce the number of SCA modules when appropriate

WebSphere Process Server components can be assembled into modules for deployment. When assembling modules we recognize that many factors come into play. While more modules can allow for better distribution across servers and nodes, modularization also has a cost. When components will be placed together in a single server instance, it is best to package them within a single module for best performance.

2.2.5 Prefer synchronous SCA bindings across local modules

For cross-module invocations, where the modules are likely to be deployed locally, such as within the same server JVM, we recommend using the synchronous SCA binding. This binding has been optimized for module locality and will outperform other bindings.

2.2.6 Utilize multi-threaded SCA clients to achieve concurrency

Synchronous components that are invoked locally, such as from a caller in the same server JVM, execute on the context of the caller's thread. Thus concurrency, if you want, must be provided by the caller in the form of multiple threads.

2.2.7 Prefer noninterruptible processes over interruptible processes

If call targets respond quickly, use noninterruptible processes if possible in conjunction with synchronous invocation.

If call targets respond slowly or the process contains human tasks, use interruptible processes. Consider implementing a process hierarchy with noninterruptible processes for heavily used parts of the overall business scenario and interruptible processes for human tasks and longer running asynchronous tasks.

2.2.8 Choose 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 POJOs, EJBs, and servlets for logic without business significance.
2.2.9 Join activities into one transaction if possible

Long-running business processes can run in multiple transactions. The border of those transactions and the number of transactions can be changed with transactional behavior settings on Java Snippet, Staff, and Invoke activities. By default, each activity runs in its own transaction. For many types of activities, this is unnecessary. For example a preceding assign can run safely in the transaction of a succeeding invoke. This is achieved by setting the transactional behavior property of the assign activity to participate. The default setting for this property is commit after, which causes each activity to run in its own transaction.

Adjust the transactional behavior settings such that the resulting execution of the long-running business process performs best. As a rule of thumb, fewer transactions result in better performance.

2.2.10 Minimize variable specifications

Use as few variables as possible and minimize the size of the 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 very costly. Use the GBO/ASBO pattern to reduce the data object size for invocations.

2.2.11 Turn off business relevance flags that are not required

The business relevance flag is turned on by default for BPEL processes, receives, picks, replies, staff, scopes, picks, and invokes. This allows business relevant data to be queried after those activities have been completed. If your scenario does not need this information, turn off the business relevance flag to get better performance for long-running business processes.

2.2.12 Increase the maximum batch size on activation specifications

The maximum batch size can be set in the activation specification. 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 (please note that this does not mean that this is done within a single transaction and therefore does not influence transactional scope). Increasing this value, for example to 8, for activation specifications associated with long-running business processes can improve performance.

2.2.13 Set the preferred interaction style to Sync whenever possible

Many WebSphere Process Server component types, such as interface maps, invoke their target components based on the target's setting of preferred interaction style. Because synchronous cross-component invocations are better performing, it is recommended to set the preferred interaction style to Sync whenever possible. Only in specific cases, for example when invoking a long-running business process, or more generally whenever the target component requires asynchronous invocation, should this be set to Async.

2.2.14 Use asynchrony judiciously

WebSphere Process Server components and modules may be wired to each other either synchronously or asynchronously. The choice of interaction style can have a profound impact on performance and care should be exercised when making this choice.
Avoid unnecessary cross-component asynchronous invocations
It is important to realize that asynchronous invocations in WebSphere Process Server are intended to provide a rich set of qualities of service, including transactions, persistence, and recoverability. As a result, an asynchronous invocation should be thought of as a full messaging hop to its target. When the intended target of the invocation is in the same module, a synchronous invocation will yield much higher performance.

2.2.15 Use object data representation for JMS
When generating JMS bindings for SCA imports and exports, there are two choices for the way that data is represented on the messaging queue: a text representation that uses XML serialization and an object representation that uses Java object serialization. The object representation produces best performance.

2.2.16 Choose efficient meta-data management
This section discusses techniques for using meta-data efficiently.

Follow Java specifications for complex data type names
While WebSphere Process Server allows characters in business object type names that would not be permissible in Java class names (the underscore character ‘_’, for example), the internal data representation of complex data type names does occasionally make use of Java types. As such, performance is better if BO types follow the Java naming standards.

Avoid use of anonymous derived types in XSDs
Some 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, then a new subtype (a derived type) is generated at run time. Performance is generally better if this can be avoided. So, 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 can utilize that type without degraded performance.

Avoid referencing elements from one XSD in another XSD
Assume A.xsd defines an element AEElement as in Example 2-1.

Example 2-1  AEElement definition referenced from another file

```xml
<xs:element name="AEElement">
  <xs:simpleType name="AEElementType">
    <xs:restriction base="xs:string">
      <xs:minLength value="0" />
      <xs:maxLength value="8" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

It can be referenced from another file, B.xsd as:

```xml
<xs:element name ref="AEElement" minOccurs="0" />
```

This reference definition has been shown to perform poorly.
It is much better to define the type concretely, then make any new elements use this type. So, `A.xsd` becomes the definition in Example 2-2.

**Example 2-2  AElement concrete definition**

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

Additionally, `B.xsd` becomes this line:

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

**Reuse data object type metadata where possible**

Within application code, it is common to refer to types, for instance when creating a new business object. It is possible to refer to a business object type by name for instance in the method `DataFactory.create(String uri, String typeName)`. It is also possible to refer to the type by a direct reference as in the method `DataFactory.create(Type type)`. In cases where a `Type` is likely to be used more than once, it is usually faster to retain the `Type`, for instance with `DataObject.getType()`, and reuse that type for the second and future uses.

### 2.2.17 Recommendations modified from previous releases

Beginning with version 6.0.1.1 of WebSphere Process Server, the overhead associated with the JMS and asynchronous SCA Invocations has been greatly reduced. Because of these enhancements, the following recommendations have been revised.

**Cross-module asynchronous invocations**

Prior to WebSphere Process Server version 6.0.1.1, the cost of a cross-module asynchronous invocation was greater than that of an intra-module asynchronous invocation followed by a cross module synchronous invocation. As a result of internal efficiency improvements in WebSphere Process Server 6.0.1.1, the situation is now reversed. Better performance will result from a cross-module asynchronous invocation.

**Preferred interaction style**

A previous recommendation was to set the preferred interaction style (sync or async) of any module called by an interface map component. Beginning with WebSphere Process Server version 6.0.1.1, interface map components no longer reflect the interaction style of the caller when invoking their target. Instead, they inspect the called preferred interaction style of the component.
2.3 Deployment

This section discusses performance tuning tips for deployment.

2.3.1 Place local modules in same server JVM

If you are planning to deploy WebSphere Process Server modules on the same physical server, better performance will be achieved by deploying the modules to the same application server JVM, as this allows the server to exploit this locality.

2.3.2 Configure MDB concurrency for asynchronous components

Asynchronous invocations of an SCA component utilize an MDB to listen for incoming events that arrive in the associated input queue. Threads for these MDBs will come from the default thread pool for an application server.

Concurrency of all MDBs together is determined by the number of threads in the WebSphere Process Server default thread pool. Each application server has a default thread pool setting that can be set with the administrative console under the Thread Pools category. Additionally, WebSphere Process Server can be configured so that MDB threads are taken from an aliased thread pool that can be defined through the WebSphere Process Server administrative console.

The number of concurrent instances of a particular MDB can be configured using the Maximum Concurrent Endpoints parameter. This parameter can also be set using the WebSphere Process Server administrative console in the configuration section for individual MDBs.

One symptom of insufficient concurrency is CPU idleness. Vary the concurrency to achieve maximum CPU utilization and throughput.

2.3.3 Configure the WorkArea service maximum send and receive sizes to no limit

Making synchronous SCA calls across JVMs results in the underlying infrastructure serializing the work area in the request and deserializes the work area from the reply. The default WorkArea service as well as the WorkArea partition service makes use of the user specified size limits as an upper limit for sizes that can be sent or received by a workarea. The possible values are 0 (no limit), -1 (default), or a non-zero positive number. It has been found that setting the sizes to 0 (no limit) is beneficial to performance as it circumvents the costly checking of the outbound or incoming workarea size.

2.3.4 Use events judiciously

The purpose of CEI event emission in WebSphere Process Server is for business activity monitoring. Because WebSphere Process Server CEI emission is a persistent mechanism, it is inherently heavy-weight. One should enable CEI only for events that have a business relevance. What we are seeing in practice appears to be confusion between business and IT monitoring, for which PMI is far more appropriate.

With this in mind, the following concepts generally are true for most customers:

1. Businesses 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 is fairly natural. We want to detect when long-running activities
complete, when human tasks change state, and so forth.

2. For short-running flows that complete within seconds, it is usually sufficient to know that a
flow completed, with perhaps 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, end) are usually sufficient for a microflow. In actuality, one
event should suffice, but two are required currently in our implementation.

2.4 WebSphere Adapters

This section discusses best practices when using WebSphere Adapters.

2.4.1 Monitor the event delivery table (EDT) and JMS queues

Events are initially stored in the EDT, and during adapter processing are also copied to JMS
queues at various points. You should monitor both the EDT and adapter JMS queues
periodically to ensure events are being processed and removed from the queues. Obsolete
events can produce delays in processing, because the adapter might attempt to recover
these events unnecessarily. The EDT table names can be found by using the administrative
console. They are specified in the J2C activation specification of the deployed application.
These tables can then be monitored and cleared using the appropriate database
administration tool (for example, if DB2 is your database use the DB2 Control Center). The
relevant JMS queues can also be found using the administrative console. Again, if using DB2,
they are specified as data sources in the Service Integration and Resources section.

The relevant tables and queues can be monitored while the adapter is running. If you detect
that events need to be removed from the EDT and JMS queues, stop the adapter first. Then,
remove the events, and restart the adapter.

2.4.2 Configuring synchronous event delivery

By default, event delivery is done using asynchronous semantics. Synchronous semantics
are also available for event delivery, and typically provide significantly greater throughput.
Whenever it is reasonable to use either synchronous or asynchronous semantics, it is
recommended that synchronous be used for better performance.

Setting synchronous semantics is done differently, based on the version of WebSphere
Integration Developer used. Please see the sections below for the relevant steps.

Using WebSphere Integration Developer 6.0.1.1

In this version there is an explicit option for configuring an adapter import to use synchronous
semantics. To set event delivery to use synchronous semantics in version 6.0.1.1, perform
the following steps:

1. Open your application in WebSphere Integration Developer. Go to the Business
Integration perspective and open the appropriate module using the Assembly Editor.

2. Right-click the adapter export box and then select Show in Properties. This will display
the Properties tab for the adapter export.

3. Select the Binding tab.
4. Select the **EIS Export Binding** tab.
5. Change the value in the Interaction Style box to **sync**.
6. Save your changes, rebuild your application, export your ear and redeploy as normal.

**Using WebSphere Integration Developer 6.0.1 or earlier**

For these versions of WebSphere Integration Developer, there is no explicit option for configuring an adapter import to use synchronous semantics. Instead, the choice between synchronous and asynchronous is implicit in the type of method imported. Any one-way method uses asynchronous semantics while request-response methods use synchronous. Thus, by changing the methods on the inbound adapter interface from one-way to request-response we can effectively change the invocation from asynchronous to synchronous and realize a significant performance gain. In order to complete the change from one-way to request-response, it is necessary to add an output to each method. This output should be a string and must always be null, otherwise a class cast exception can occur upon invocation of the method.

An example of this follows.

1. Open your application in WebSphere Integration Developer. Go to the Business Integration perspective and open the appropriate module using the Assembly Editor.
2. Right-click the Interface box and then select **Show in Properties**. This will display the Properties tab for the applications interfaces.
3. Select the top interface, go to the Details tab and select **Open** (which might be located to the far right of the details window). This will open a page that lists all of the applications interfaces.
4. Select (left-click) the interface you want to modify, then select the **Request Response** radio button on the Details tab of the Properties view.
5. Right-click the interface that you want to change. Select **Add Output** from the action list. Your interface should now have an Output(s) entry as part of the Interface description.
6. Verify that the type of the output is String (the default).
7. Save your changes by selecting **File → Save**.

You have now completed the synchronous modifications to your interface. Next you will need to change the implementation to match the Interface.

1. Open the Java source file that contains your Interfaces. The default file would be **Component1Impl.java**. Add String to the definition of the interfaces that you changed, for example:

   ```java
   public String emitCreateAfterImageWbiCustomerCi(DataObject emitCreateAfterImageWbiCustomerCiInput)
   ```

2. Add a `return(null);` to all of the exit points in your implementations method.
3. Save your changes, rebuild your application, export your EAR file, and redeploy it as normal.
2.5 Large objects

An issue frequently encountered by field personnel is trying to identify the largest object size both WebSphere Process Server and the corresponding adapters can effectively and efficiently process. Because there is no one simple answer to this question, we present both a discussion of the issues involved as well as some practical guidelines for the current releases of WebSphere Process Server and adapters.

In general, objects 5 MB or larger can be considered large and require special attention. For reasons discussed here, it is recommended that if these large objects must be supported, the solution design should be architected to break the object into multiple objects sized at or below 1 MB, and process them individually. This approach generally produces a more robust and higher-performing solution.

2.5.1 Factors affecting large object size processing

Stated at a high level, the object-size capacity for a given 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 business object that can be successfully processed.

To be able to apply this somewhat general statement, one must first understand that the business object size limit is based on three fundamental implementation facts of Java Virtual Machines:

1. Java heap size limitations
   
   The limit to the size of the Java heap is operating system-dependent. Further details of heap sizes are given later in this section, but it is not unusual to have a heap size limit of around 1.4 GB.

2. Java heap fragmentation
   
   In a production WebSphere Process Server environment, the server and adapters might have been operating for a long period before a large object arrives. All JVMs implement a heap compaction function that will limit, but cannot totally eliminate heap fragmentation. There are some objects in the Java heap which cannot be moved. As a result, fragmentation will always exist in the heap, potentially making the amount of contiguous memory available in a single block much less than the overall total of available memory. Java objects must be allocated in contiguous space. This fragmentation is normally not an issue as most Java object requests are small and easily allocated, even in a fragmented heap.

3. Contiguity of Java objects
   
   If a large Java object is requested and, after compaction and growing the heap to its maximum size, there is not a sufficiently large contiguous chunk of Java Heap available, then Java will throw an out of memory exception. This is a typical scenario encountered with very large business objects, and brings to light two very important facts:
   
   - First, even though there can be plenty of free space available in the heap, an out-of-memory exception can still occur. We have seen instances in the field where half of a 1.6 GB heap was free (800 MB) but a 38 MB allocation caused an out-of-memory exception. As previously described, this is due to the fact that the Java heap becomes fragmented over time.
   
   - Second, there is an issue concerning the relationship between a WebSphere Process Server business object (BO) and the Java objects that represent them. A BO which appears as a 10 MB message on a JMS input queue for an adapter, for example, can
result in the allocation of many larger Java objects as it flows through the WebSphere Process Server and adapters. That is, a BO that consumes 10 MB on the input message queue can result in allocations of 20 to 30 MB on the Java heap, since single-byte binary objects are generally converted to multi-byte character representations (for example, Unicode). The fact that a BO will consume more of the Java heap than the corresponding message might be due to growth in the mapping of objects as well as to the internal implementation details in the WebSphere Process Server and WebSphere Adapters.

2.5.2 Best practices to mitigate large object issues

It has been observed that the use of large objects in an architected solution is frequently associated with batch-oriented business processes, where a large number of smaller transactions are assembled and submitted as a single object for processing. Given that, you must address the three issues outlined in the previous section.

**Heap limitations: increase the Java heap to its maximum**

This is addressed by platform.

**Microsoft Windows**

As mentioned previously, WebSphere Process Server 6.0.1.1 and WebSphere Adapters v6.0.0.1 ship with 1.4.2 JVMs from IBM on Windows platforms. Due to address space limitations in the Windows operating system, the largest heap that can be obtained is around 1.4 GB to 1.6 GB.

**AIX**

The current version of WebSphere Process Server 6.0.1.1 and WebSphere Adapters v6.0.0.1 support the 32-bit version of 1.4.2 JVMs from IBM on AIX. Using the normal Java heap settings, the 1.4.2 JVM supports heaps in excess of 3 GB. Please refer to the document at the following URL for a description of the interaction between the IBM JVM and the AIX virtual memory manager:

http://www.ibm.com/servers/esdd/articles/aix4java/

**Solaris**

On Solaris, WebSphere Process Server 6.0.1.1 and WebSphere Adapters v6.0.0.1 ship with the 32-bit version of the 1.4.2 JVM from Sun Microsystems. Laboratory experiments indicate a maximum of approximately a 3.5 GB heap for WebSphere Process Server 6.0.1.1 workloads.

**Reduce Java heap fragmentation**

Reduce or eliminate other processing within the WebSphere Process Server and Adapters while processing a large object.

One way to significantly reduce the fragmentation in the Java heap and allow for larger object sizes is to limit the concurrent processing activity occurring within the JVM. Obviously, one should not expect to be able to process a steady stream of the largest objects possible concurrently with other WebSphere Process Server and WebSphere Adapters activities. The operational assumption that needs to be made when considering large objects is that not all objects will be large and that large objects will not arrive very often, perhaps once or twice per day. If more than one large object is being processed by WebSphere Process Server concurrently the likelihood of failure increases dramatically.
The size and number of the normally arriving smaller objects will affect the amount of Java heap memory consumption, and possibly fragmentation as well, that exists within the WebSphere Process Server and adapters. Generally speaking, the heavier the load on a system when a large object is being processed, the more likely memory problems will be encountered.

For adapters, the amount of concurrent processing can be influenced by setting the PollPeriod and PollQuantity parameters on the J2C activation spec. To allow for larger object sizes, set a relatively high value for PollPeriod (such as 1 second) and low value for PollQuantity (such as 1) to minimize the amount of concurrent processing that occurs. Note that these settings are not optimal for peak throughput, so if a given adapter instance must support both high throughput for smaller objects interspersed with occasional large objects, then trade-offs must be made. For a detailed discussion on setting these parameters, see 1.4, “WebSphere Adapters” on page 12.

Additionally, for the mySAP.com adapter when using the ALE module for event delivery set the connector-specific NumberOfListeners configuration property to 1. This will limit the number of IDOCs processed concurrently by the adapter.

Contiguity of Java objects: send large objects as multiple small objects
If a large object needs to be processed on current WebSphere Process Server and adapter technologies then the solutions engineer must find a way to limit the large Java objects being allocated. The primary technique for doing so involves decomposing large business objects into smaller objects and submitting them individually.

If the large objects are actually a collection of small objects as assumed above, the solution is to group the smaller objects into conglomerate objects less than 1 MB in size. This has been done at a variety of customer sites for predecessor products. If there are temporal dependencies or an all-or-nothing requirement for the individual objects, then the solution becomes more complex. Implementations at customer sites using predecessor products have shown that dealing with this complexity is worth the effort as demonstrated by both increased performance and stability.

2.6 WebSphere Business Monitor

This section discuss best practices for WebSphere Business Monitor.

2.6.1 Event processing
If the Monitor server is turned off while events are being posted to the JMS queue, the number of events in the queue will grow. The throughput performance of this particular implementation of a JMS queue degrades as the number of events in the queue grows. Therefore, Monitor server should always be running if events are being posted to this queue.

2.6.2 Replication
In steady state, replication can keep up with a high rate of event processing. However, if a large number of unreplicated events is allowed to accumulate before replication is turned on, replication may have a hard time catching up. Thus, it is not a good idea to run event processing for long with replication turned off. Even when replication is on, it only replicates periodically. The replication intervals should be short enough to prevent large numbers of unreplicated events from accumulating.
The DB2 replication manager creates temporary copies and other metadata concerning each event it replicates. Periodically, it cleans up such objects relating to events it has finished replicating in a process called **pruning**. The pruning interval is configurable, and it should be short enough to prevent too many such objects from accumulating. We have found that setting this interval to zero, which means **prune as you go**, yields best performance. This can be done with the following set of DB2 commands in Example 2-3.

**Example 2-3  DB2 commands for pruning intervals**

```sql
  db2 connect to <name of your Runtime DB>
  db2 update WBIRMADM.RMPRUNECTRL set PRUNE_INTERVAL = 0 where TABLE_NAME like 'APP.CCD_%';
  db2 connect reset
  db2 connect to <name of your Historical DB>
  db2 update WBIRMADM.RMPRUNECTRL set PRUNE_INTERVAL = 0 where TABLE_NAME like 'APP.CCD_%';
```

### 2.6.3 CEI's JMS queue

The JMS queue provided by CEI into which processes put BPEL events and out of which Monitor takes events is persistent, and by default the persistence is provided by the default messaging provider, which in turn uses the Cloudscape database management system. We have found that performance can be improved by replacing Cloudscape in this role with DB2.
Measurement configurations used in performance testing

In this chapter we provide suggested initial settings for several relevant parameters. These values will not be optimal in all cases, but we have found these values work well in our internal performance evaluations. They are, at a minimum, useful starting points for many POCs and customer deployments.
3.1 WebSphere Process Server settings

The following settings are used for internal performance evaluations of WebSphere Process Server; consider these settings useful starting points for your usage of this product. Otherwise, you can use the default settings supplied by the product installer as a starting point.

3.1.1 WebSphere Process Server common settings

The following settings were made using the administrative console:

- Server "Run in development mode" is unchecked.
- A cell level DB2 Universal XA JDBC provider and data source were created for use with relationships. The connection pool max connections for the data source was set to 50.
- A server level DB2 Universal XA JDBC provider and data sources were created for the messaging engine data stores. The connection pool max connections for these data sources was set to 20.
- An asynchronous beans work manager is set up for driver threads with the maximum number of threads set to 10 on Windows and 18 for AIX.
- For most workloads, the Java heap was set to a fixed size of 512 MB.
- The statement cache size for the BPE database is set to 300.

3.1.2 WebSphere Process Server settings

For the Business Process Choreography performance evaluations, two DB2 databases are created, largely following the procedures outlined in 1.2.3, "Move databases off of the default Cloudscape" on page 3. One database is the BPC database and the other is the SIB messaging engine database. The SIB database contains four different schemas, one each for the BPC, CEI, SCA System and the SCA Application buses.

Example 3-1 shows the parameters that we used to tune the BPC database.

Example 3-1  BPC database tuning

```
db2 update database configuration using APP_CTL_HEAP_SZ 144;
db2 update database configuration using APPGROUP_MEM_SZ 13001;
db2 update database configuration using CATALOGCACHE_SZ 521;
db2 update database configuration using CHNGPGS_THRESH 55;
db2 update database configuration using DBHEAP 600;
db2 update database configuration using LOCKLIST 500;
db2 update database configuration using LOCKTIMEOUT 30;
db2 update database configuration using LOGBUFSZ 245;
db2 update database configuration using LOGFILSZ 1024;
db2 update database configuration using LOGPRIMARY 11;
db2 update database configuration using LOGSECOND 10;
db2 update database configuration using MAXAPPLS 90;
db2 update database configuration using MAXLOCKS 57;
db2 update database configuration using MINCOMMIT 1;
db2 update database configuration using NUM_IOCLEANERS 6;
db2 update database configuration using NUM_IOSERVERS 10;
db2 update database configuration using PCKCACHESZ 915;
db2 update database configuration using SOFTMAX 440;
db2 update database configuration using SORTHEAP 228;
```
Example 3-2 shows the parameters that were used to tune the SIB database.

Example 3-2  SIB database tuning

```sql
db2 update database configuration using APP_CTL_HEAP_SZ 144;
db2 update database configuration using APPGROUP_MEM_SZ 13001;
db2 update database configuration using CATALOGCACHE_SZ 521;
db2 update database configuration using CHNGPGS_THRESH 55;
db2 update database configuration using DBHEAP 600;
db2 update database configuration using LOCKLIST 500;
db2 update database configuration using LOCKTIMEOUT 30;
db2 update database configuration using LOGBUFSZ 245;
db2 update database configuration using LOGFILSZ 1024;
db2 update database configuration using LOGPRIMARY 11;
db2 update database configuration using LOGSECOND 10;
db2 update database configuration using MAXAPPLS 90;
db2 update database configuration using MAXLOCKS 57;
db2 update database configuration using MINCOMMIT 1;
db2 update database configuration using NUM_IOCLEANERS 6;
db2 update database configuration using NUM_IOSERVERS 10;
db2 update database configuration using PCKCACHESZ 915;
db2 update database configuration using SOFTMAX 440;
db2 update database configuration using SORTHEAP 228;
db2 update database configuration using STMTHSEP 2048;
db2 update database configuration using DFT_DEGREE 1;
db2 update database configuration using DFT_PREFETCH_SZ 32;
db2 update database configuration using UTIL_HEAP_SZ 11663;
db2 update database manager configuration using SHEAPTHRES 27416;
db2 update database manager configuration using INTRA_PARALLEL OFF;
db2 update database manager configuration using MAX_QUERYDEGREE 2;
db2 update database manager configuration using MAXAGENTS 400;
db2 update database manager configuration using NUM_POOLAGENTS 120;
db2 update database manager configuration using NUM_INITAGENTS 0;
db2 update database manager configuration using FCM_NUM_BUFFERS 4096;
db2 update database manager configuration using PRIV_MEM_THRESH 32767;
db2 alter bufferpool IBMDEFAULTBP SIZE 214990;
db2 set current QUERY OPTIMIZATION = 5;
```
In addition to these database level parameter settings, several other parameters were also modified using the administrative console, mostly those affecting concurrency, such as thread settings.

1. The first change was to increase the total number of threads available to use by the Default thread pool:

   Navigate to Servers → server1 → Thread Pools → Default.

   Increase the Maximum Size from 20 to 30.

2. The next tuning actions impact bus concurrency, increasing the maxConcurrency parameter for the activation specifications used in testing the throughput performance of a typical long running business process (macroflow).

   These settings can be found in the following locations:


      Set maximumConcurrency for each application-specific activation spec to 30.


      • Set maximumConcurrency for each application specific activation spec to 25.
      • Set maximumConcurrency for BPEApiActivationSpec to 15.
      • Set maximumConcurrency for BPEInternalActivationSpec to 15.

3. Set Database Connection concurrency parameters as follows:

   a. Navigate to Resources → JDBC Providers → DB2 Universal JDBC Provider (XA) → Data Sources → <BpeDataSourceDb2 >.

      • Under Connection Pool Properties, set Maximum Connections to 60.
      • Under WebSphere Application Server data source properties, set Statement Cache Size to 300.

   b. Navigate to Resources → JDBC Providers → DB2 Universal JDBC Provider (non-XA) → Data Sources.

      • Select the data source for the Process Choreographer database. Under Connection Pool Properties, set Maximum Connections to 60.
      • Select the data source for SCA application service integration bus. Under Connection Pool Properties, set Maximum Connections to 40.
      • Select the data source for SCA system service integration bus. Under Connection Pool Properties, set Maximum Connections to 40.

4. The final set of tuning changes impact JMS Connection Pool, as follows:

   a. Navigate to Resources → JMS Providers → Default Messaging (server scope) → JMS Connection Factories.

      For each connection factory (BPECFC, BPECF, and any used by the application), select Connection Pool Properties and set Maximum Connections to 40.

3.2 WebSphere ESB settings

   This section discusses settings used for internal performance evaluations of WebSphere ESB; consider these settings useful starting points for your use of this product. Otherwise, the default settings supplied by the product installer can be used as a starting point.
3.2.1 WebSphere ESB common settings

These settings are used for all the WebSphere ESB performance evaluations, for both Web services and JMS.

- Tracing is disabled.
- Java Heap size is set to 1280 MB max and min.
- Set DTMManager property in the xalan.properties file.

3.2.2 WebSphere ESB settings for Web services tests

The following settings were used for the Web services performance evaluations:

- PMI monitoring is disabled.
- WebContainer thread pool sizes set to max 15 and min 10.
- WebContainer thread pool inactivity timeouts for thread pools set to 3500.
- HTTP settings keepAliveEnabled set to true and maxPersistentRequests to -1.
- EJB™ Cache properties set ejbCacheSize to 2053.
- Set Java2 security to false.

3.2.3 WebSphere ESB settings for JMS tests

These settings were used for the JMS performance evaluations:

- Activation specification, set maximum concurrent endpoints to 20.
- Queue Connection factory, set the maximum connection pool size to 20.
- DiscardableDataBufferSize set to 10 MB and CachedDataBufferSize set to 40 MB.

3.2.4 DB2 Settings for WebSphere ESB JMS persistent tests

These settings are only relevant to the JMS persistent performance evaluations because those performance evaluations use the database to persist messages.

- Place database table spaces 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.
- Raw partition for DB2 logs.

3.3 Database manager common settings

The settings in this section are used for internal performance evaluations using DB2 8.1 FP7; consider these settings useful starting points for your use of this product. Otherwise, the default settings supplied by the product installer can be used as a starting point.

- db2set DB2_RR_TO_RS=1
- db2set DB2COMM=tcpip
- db2set DB2AUTOSTART=TRUE
No other tuning was done for the database except for the stress 10-hour runs. For high-volume, long-running processing, use the following settings for the relationship DB:

- `db2 update db cfg for db2cw using dbheap 4800`
- `db2 update db cfg for db2cw using logbufsz 512`
- `db2 update db cfg for db2cw using logfilsiz 4000`
- `db2 update db cfg for db2cw using logprimary 6`
- `db2 update db cfg for db2cw using logsecond 10`

### 3.4 WebSphere Business Monitor settings

The settings in Example 3-3 are DB2 8.2 settings used for internal performance evaluations of WebSphere Business Monitor; consider these settings useful starting points for your usage of this product. Otherwise, the default settings supplied by the product installer can be used as a starting point.

**Example 3-3   DB2 8.2 settings**

```
db2set DB2_EXTENDED_OPTIMIZATION=ON
db2set DB2_HASH_JOIN=y
db2set DB2_RR_TQ_RS=YES
db2set DB2COMM=tcpip
db2set DB2AUTOSTART=YES

db2 update db cfg for stdm7 using locklist 2048
db2 update db cfg for rtmd7 using locklist 2048
db2 update db cfg for histmd7 using locklist 2048
db2 update db cfg for stdm7 using maxlocks 25
db2 update db cfg for rtmd7 using maxlocks 25
db2 update db cfg for histmd7 using maxlocks 25

db2 update db cfg for stdm7 using sortheap 4096
db2 update db cfg for rtmd7 using sortheap 4096
db2 update db cfg for histmd7 using sortheap 4096

update db cfg for histmd7 using logprimary 10 logsecond 80 logfilsiz 5000

```

```
3.5 Java settings

This section lists Java settings used for internal performance evaluations; consider these settings useful starting points for your usage of this product. Otherwise, the default settings supplied by the product installer can be used as a starting point.

- General
  
  `-verbosegc (for gathering verbose garbage collection statistics)

- WebSphere Process Server
  
  `-Xms512M `-Xmx512M

- WebSphere Business Monitor
  
  `-Xms1000M `-Xmx1000M
Related publications

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

Online resources

These Web sites are also relevant as further information sources:

- **WebSphere Process Integration Version 6.0 information center**
  

- **Messaging best practices**
  

- **DB2 best practices**
  

- **DB2 tuning**
  

- **DB2 Administration Guide: Performance**
  

- **WebSphere Application Server Performance URL**
  
  [http://www.ibm.com/software/webservers/appserv/was/performance.html](http://www.ibm.com/software/webservers/appserv/was/performance.html)

- **Tuning Performance section of the WebSphere Application Server v6 Information Center (Includes tuning for messaging)**
  

- **Modeling Efficient BPEL Processes (Document)**
  

- **WebSphere ESB 6011 Fixpack**
  

- **Using J2CA Adapters with WebSphere ESB**
  

- **Guided Tour of WebSphere Integration Developer  Part 1**
  

- **WebSphere ESB InfoCentre**
  
How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads
ibm.com/support

IBM Global Services
ibm.com/services
## Index

### A
- activation specification 7–8, 10, 26
- AIX native threads 21
- anonymous derived type 27
- asynchronous beans 7
- asynchronous component 29
- asynchronous invocation 27–29
- audit logging 3
- auto-build 24
- auto-publish 24

### B
- BPC 6
- BPE data source 8
- BPE data source connection pool 8
- BPE database 3
- BPEDB 7
- BPEL 13, 26
- BPEL macroflow 7
- buffer pool size 17
- BUFFPAGE 17
- Business Process Choreographer 4
- business relevance flag 26
- bytecode verification 3

### C
- cache 7
- cache miss 7
- cache size 7
- cached data buffer 11
- CachedDataBufferSize 11
- catalog statistics 17
- CEI 3, 6, 13–14, 35
- CEI event emission 29
- CEI queue 14
- class cast exception 31
- Cloudscape 3–4, 9, 12, 16, 35
- Cloudscape JDBC Provider (XA) 7
- cluster 8–9
- com.ibm.websphere.webservices.http.maxConnection 9
- commit after 26
- compactions 19
- complex data type 27
- concurrency 10, 25, 29
- connection pool size 15
- connection pooling 11
- connectionKeepAlive 11
- ConnectionWaitTimeout 10
- container managed persistence (CMP) 6
- CPU idle 7, 29
- CPU utilization 7
- cross-module asynchronous invocation 28
- cross-referencing table 16

### D
- data mining 13
- data source 5–6
- database log 5
- database log file 16
- DB2 3–4, 9, 11, 16
- DB2 Configuration Wizard 6
- DB2 Control Center 17
- DB2 CubeViews 13
- DB2 Replication Manager 13
- DB2 replication manager 35
- DB2 Universal JDBC Driver Provider 6
- default heap size 19
- default thread pool 29
- default thread pool size 8
- Deserialization WorkManager work request queue 15
- Design Advisor 17
- development mode 3
- DiscardableDataBufferSize 11
- driver type 6
- DTMManager 11

### E
- event arrival rate 13
- Event Delivery Table 12
- event delivery table 30
- event processing 34
- event processing batch size 14
- Event Processing Queue Size 15
- excessive memory usage 12
- export 27

### F
- full heap GC 18

### G
- garbage collection 18, 20
- garbage collection (GC) 18
- GBO/ASBO pattern 26
- growable 15

### H
- heap compaction 32
- heap fragmentation 32
- heap limitations 33
- heap size 7, 13, 19–21
- heap usage 19
- historical database 13
- HistoricalDBDataSource 15
- horizontal scalability 8
- HotSpot 20–21
- HotSpot JVM 17
HTTP connection
KeepAlive 11
tcpdump 9

human task activities 29

I
IBM HTTP server 9
IDOC 34
import 27
Initial Heap Size 10
initial heap size 7, 20
insufficient concurrency 7
Interaction Style 31
interruptible process 25
Invoke 26

J
Java component 24–25
Java heap 32
Java heap fragmentation 32–33
Java heap memory utilization 24
Java heap size 7, 13, 17, 19, 32
Java object 32
Java object contiguity 34
Java object serialization 27
Java Snippet 26
Java snippet 24–25
Java Virtual Machine (JVM) 17
java.lang.OutOfMemory exception 22
JDBC provider 6
jdbc/BPEDB 7
JIT 21
JIT compilation 3
JMS binding 27
JMS queue 30, 34
JNDI 6–7
JVM 7, 10, 32–33

L
large object 33–34
large objects 32
long-running activities 30
long-running process 7, 26
low system throughput 12

M
maxConcurrency 7–8
maximum batch size 26
Maximum Concurrent Endpoints 29
maximum concurrent endpoints 10
maximum connection pool size 10
Maximum Connections 3
maximum connections 8, 10, 12
maximum CPU utilization 29
Maximum Heap Size 10
maximum heap size 7, 20–21
maximum number of threads 7
MaxNewSize 20
MaxSpareThreads 9
MDB 8, 29
messaging engine 4–6
messaging engine data store 3–4, 12
messaging engines data store 6
metadata 28
microflow 30
Monitor Dashboard 15
Monitor Dashboards 13
Monitor Launchpad 14–15
monitoring 3, 9
MS queue 35
multi-threaded SCA clients 25
mySAP.com adapter 34

N
NewSize 20
non-interruptible process 25
NumberOfListeners 34
Nursery GC 18
nursery size 20

O
object representation 27
one-way method 31
ORB 8
ORB thread 8
ORB thread pool 8
ORB thread pool size 8
Out Of Memory error 20, 22
out of memory exception 32
OutOfMemory exception 12–13

P
paging 13
participate 26
Platform Messaging Component SPI Resource Adapter 7
PMI 3, 12, 29
POJO 25
PollPeriod 12–13, 34
PollQuantity 12–13, 34
pool manager 10
preferred interaction style 26, 28
Profile Creation Wizard 4
prune as you go 35
pruning interval 35
Pthread 21

Q
queue connection factory 10

R
rate-trigger heap growth 19
Redbooks Web site 46
Contact us xi
relationship 16
relationship data source 8
relationship database  3
replication  34
RepositoryDBDataSource  15
RuntimeDBDataSource  15

S
SCA   8
SCA module   8, 25
SCA service   24
SCA synchronous binding   8
schema   4–6
short running flows   30
single-byte binary objects   33
single-phase commit (non-XA)   6
Staff   26
statement cache   7, 11
synchronous event delivery   30
synchronous SCA binding   25
synchronous SCA call   29
System.gc()   18
Systemout.log   6

T
table space   5, 17
text representation   27
thread   7–8, 12, 17, 21, 29
thread stack size   22
tracing   3, 9
transactional scope   26

V
verbosegc   18, 20–21

W
Web container   8
Web container thread pool   10
Web service   9
WebContainer   8
wid.ini   24
WLM   8
wmmds   15
work manager   7
Work request queue size   15
WorkArea partition service   29
WorkArea service   29
wpsdbDS   15

X
xalan.properties   11
XSD   27
XSLT Transformation primitive   11
This IBM Redpaper provides performance tuning tips and best practices based on the performance teams experience for WebSphere Process Server 6.0.1.1, WebSphere Enterprise Service Bus (WebSphere ESB) 6.0.1.1, WebSphere Integration Developer 6.0.0, WebSphere Adapters 6.0.0.1, and WebSphere Business Monitor 6.0.0.

For those who are either considering or are in the very early stages of implementing a solution incorporating these products, this document should prove a useful reference, both in terms of best practices during application development and deployment, and as a starting point for setup, tuning and configuration information. It provides a useful introduction to many of the issues affecting these products' performance, and could act as a guide for making rational first choices in terms of configuration and performance settings.

Finally, because all of these products build upon the capabilities of the WebSphere Application Server infrastructure, solutions based on these products will also benefit from existing tuning, configuration, and best practices information for WebSphere Application Server documented in a list of related publications for this paper. You are encouraged to use the information in this paper in conjunction with these references.