WebSphere® Application Server for z/OS, Version 6.1

IBM

# Tuning guide

SA22-7963-05

**Note**

Before using this information, be sure to read the general information under “Notices” on page 165.

**Compilation date: May 5, 2006**

# Contents

## How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information.

- To send comments on articles in the WebSphere Application Server Information Center
  1. Display the article in your Web browser and scroll to the end of the article.
  2. Click on the **Feedback** link at the bottom of the article, and a separate window containing an e-mail form appears.
  3. Fill out the e-mail form as instructed, and click on **Submit feedback** .
- To send comments on PDF books, you can e-mail your comments to: **wasdoc@us.ibm.com** or fax them to 919-254-0206.

  Be sure to include the document name and number, the WebSphere Application Server version you are using, and, if applicable, the specific page, table, or figure number on which you are commenting.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

# Chapter 1. Overview and new features for tuning performance

Use the links provided in this topic to learn about tuning applications and their environment.

**New for administrators: Improved monitoring and performance tuning**
A section of this topic describes what is new in the area of performance tuning.

## Tuning parameter hot list

The following hot list contains recommendations that have improved performance or scalability, or both, for many applications.

WebSphere Application Server provides several tunable parameters and options to match the application server environment to the requirements of your application.

- **Review the hardware and software requirements**

  Review the hardware and software requirements on the IBM WebSphere Application Server supported hardware, software, and APIs Web site to get started.
- **Check hardware configuration and settings**

  Check network connections to make sure that they are running at their highest speed. Sometimes transient errors can cause Ethernet adapters to shift down to a lower speed. Verify that the system has adequate memory and that the number and position of memory dual inline memory module (DIMM) are optimal. With some systems, there are some memory DIMM configurations that permit higher performance than other DIMM configurations. Verify that the hardware that is used is the hardware that is supposed to be used.
- **Review your application design**

  You can track many performance problems back to the application design. Review the design to determine if it causes performance problems.
- **Tune the operating systems**

  Operating system configuration plays a key role in performance. In many cases, adjustments to some TCP/IP parameters might be necessary for your application.
- **Set the minimum and maximum Java virtual machine (JVM) heap sizes**

  Many applications need a larger heap size for best performance.
- **Use a type 4 (or pure Java) JDBC driver**

  In general, the type 2 JDBC driver is recommended. However, the type 4 JDBC drivers perform faster in the area of multi-row fetch. Use the link above to view a list of database vendor-specific requirements, which can tell you if a type 4 JDBC driver is supported for your database.

  See the *Administering applications and their environment* PDF for more information.
- **Tune WebSphere Application Server JDBC data sources and associated connection pools**

  The JDBC data source configuration might have a significant performance impact. For example, the connection pool size and prepared statement cache need to be sized based on the number of concurrent requests being processed and the design of the application.

  See the *Administering applications and their environment* PDF for more information.
- **Ensure that the transaction log is assigned to a fast disk**

  Some applications generate a high rate of writes to the WebSphere Application Server transaction log. Locating the transaction log on a fast disk or disk array can improve response time

  See the *Administering applications and their environment* PDF for more information.
- **Tune related components, for example, database**

  In many cases, some other component, for example, a database, needs adjustments to achieve higher throughput for your entire configuration.

  For more information, see the *Administering applications and their environment* PDF for more information.
- **Tuning index for WebSphere Application Server for z/OS**

One of the goals of the WebSphere Application Server for z/OS programming model and runtime is to significantly simplify the work required for application developers to write and deploy applications. Sometimes we say that WebSphere Application Server for z/OS relieves the application programmer of many of the plumbing tasks involved in developing applications. For example, application code in WebSphere Application Server for z/OS does not concern itself directly with remote communication--it locates objects which may be local or remote and drives methods. Therefore, you won't see any direct use of socket calls or TCP/IP programming in a WebSphere Application Server for z/OS application.

This separation of what you want to do from where you do it is one aspect of removing the application programmers from plumbing tasks. Other considerations are not having to deal with data calls for some types of beans, potentially user authentication, and threading. There are generally no calls from the application code to touch sockets, RACF calls, or management of threading. Removing this from the application programmer doesn't mean this work won't get done. Rather, it means that there may be more work for the DBA, the network administrator, the security administrator, and the performance analyst.

There are four layers of tuning that need to be addressed:
- "Tuning the z/OS operating system" on page 66
- Tuning for subsystems
- Tuning the WebSphere Application Server for z/OS runtime
- Tuning for J2EE applications

We deal with the first three in separate sections under this article and briefly touch on the fourth. For more information on tuning applications, refer to Using application clients.

- **Tuning for subsystems**

  Steps involved in tuning the z/OS subsystems to optimize WebSphere performance include:
  - "DB2 tuning tips for z/OS" on page 3
  - "Security tuning tips" on page 126
  - "Tuning TCP/IP buffer sizes" on page 56
  - "GRS tuning tips for z/OS" on page 141
  - "Tuning Java virtual machines" on page 57
  - "CICS tuning tips for z/OS" on page 141
- **Tuning the WebSphere Application Server for z/OS runtime**

  Steps involved in tuning the WebSphere Application Server for z/OS runtime to optimize performance include reviewing the:
  - Review the WebSphere for z/OS configuration
  - Internal tracing tips for WebSphere Application Server for z/OS
  - Location of executable programs tips for z/OS
  - "Security tuning tips" on page 126
- **Review the WebSphere for z/OS configuration**

  The first thing to do is review the WebSphere for z/OS configuration. One simple way to do this is to look in your application control and server regions in SDSF. When each server starts, the runtime prints out the current configuration data in the joblog.
- **Internal tracing tips for WebSphere for z/OS**

  WebSphere traces can be extremely helpful in detecting and diagnosing problems. By properly setting trace options, you can capture the information needed to detect problems without significant performance overhead.
  - Ensure that you are not collecting more diagnostic data than you need.

    You should check your WebSphere for z/OS tracing options to ensure that ras_trace_defaultTracingLevel=0 or 1, and that ras_trace_basic and ras_trace_detail are not set.

    **How to view or set:** Use the WebSphere administrative console:
    1. Click **Environment > Manage WebSphere Variables**.
    2. On the Configuration Tab check for any of these variables in the name field and observe the variable setting in the value field.
    3. To change or set a variable, specify the variable in the name field and specify the setting in the value field. You can also describe the setting in the description field on this tab.

- If you use any level of tracing, including ras_trace_defaultTracingLevel=1, ensure that you set ras_trace_outputLocation to BUFFER.

  ras_trace_defaultTracingLevel=1 will write exceptions to the trace log as well as to the ERROR log.
  - It is best to trace to CTRACE.

    If you are tracing to sysprint with ras_trace_defaultTracingLevel=3, you may experience an almost 100% throughput degradation. If you are tracing to CTRACE, however, you may only experience a 15% degradation in throughput.
- Set the ras_trace_BufferCount=4 and ras_trace_BufferSize=128.

  This will get 512KB of storage for the trace buffers (the minimum allowed) and reduce memory requirements.
- Make sure you disable JRAS tracing.

  To do this, look for the following lines in the trace.dat file pointed to by the JVM properties file:

  ```
  com.ibm.ejs.*=all=disable

  com.ibm.ws390.orb=all=disable
  ```

  Ensure that both lines are set to **=disable** or delete the two lines altogether.

  **Note:** If ras_trace_outputLocation is set, you may be tracing and not know it.

• **Location of executable programs tips for z/OS**

  The next thing to review in the configuration is where your program code is located. IBM recommends that you install as much of the WebSphere for z/OS code in LPA as is reasonable, and the remainder in the linklist. This ensures that you have eliminated any unnecessary steplibs which can adversely affect performance. If you must use STEPLIBs, verify that any `STEPLIB DD`s in the controller and servant procs do not point to any unnecessary libraries. Refer to “UNIX System Services (USS) tuning tips for z/OS” on page 72 for USS shared file system tuning considerations.

  If you choose to not put most of the runtime in LPA, you may find that your processor storage gets a bigger workout as the load increases. At a minimum, WebSphere for z/OS will start three address spaces, so that any code that is not shared will load three copies rather than one. As the load increases, many more servants may start and will contribute additional load on processor storage.

  Review the `PATH` statement to ensure that only required programs are in the PATH and that the order of the PATH places frequently-referenced programs in the front.

• **Tuning for J2EE applications**

  Steps involved in tuning the J2EE applications performance include:
  - “Tuning WebSphere applications” on page 78
  - “Tuning TCP/IP buffer sizes” on page 56
  - Tuning for SOAP
  - “Tuning MDB processing on z/OS” on page 6

## DB2 tuning tips for z/OS

Performance tuning for DB2 is usually critical to the overall performance of a WebSphere Application Server application. DB2 is often the preferred datastore for Enterprise JavaBeans (EJBs). Listed here are some basic guidelines for DB2 tuning as well as some guidelines for tuning DB2 for WebSphere Application Server. For more complete information on DB2 tuning, refer to the DB2 Universal Database for OS/390 and z/OS Administration Guide Document Number SC26-9931-03. The DB2 books can be accessed at the following Internet location:http://www.ibm.com/servers/eserver/zseries/zos/.

### Benefits of Structured Query Language in Java (SQLJ)

If you use the DB2 Universal JDBC driver provider, you can implement SQLJ as the query language for both BMP and CMP beans. SQLJ incurs less transaction overhead than the default query language for JDBC transactions, which is dynamic SQL. SQLJ is static and uses pre-prepared plans. Thus SQLJ generally improves application performance. For DB2 for z/OS database administrators, SQLJ is often easy to adopt because the security model and the statement repeatability features are similar to those of static SQL. SQLJ does require extra steps that are features of newer versions of WebSphere Studio Application Developer and Rational Application Developer..

See the *Administering applications and their environment* PDF for more information.

**General DB2 tuning tips:**

This discussion relates only to DB2 for z/OS JDBC Driver which is referred to as the DB2 for z/OS Legacy JDBC Driver.

- First, ensure that your DB2 logs are large enough, are allocated on the fastest volumes you have, and make sure they have optimal CI sizes.
- Next, ensure that you have tuned your bufferpools so that the most often-read data is in memory as much as possible. Use ESTOR and hyperpools.
- You many want to consider pre-formatting tables that are going to be heavily used. This avoids formatting at runtime.

**DB2 for WebSphere tuning tips:**

- Ensuring DB2 Tracing Under the DB2 for z/OS Universal Driver is Turned Off.
  - If the db2.jcc.propertiesFile jvm property has been defined to specify a DB2 jcc properties file to the WebSphere Application Server for z/OS, ensure that the following trace statements in the file are commented out if they are specified:

    ```
    # jcc.override.traceFile=<file name>
    # jcc.override.traceFile=<file name>
    ```
  - If any of the DB2 Universal JDBC Driver datasources your applications are using are defined with a nonzero traceLevel custom property, use the WebSphere Application Server for z/OS Administrative console to set the traceLevel to zero.
- We recommend that you ensure indexes are defined on all your object primary keys. Failure to do so will result in costly tablespace scans.
- Ensure that, once your tables are sufficiently populated, you do a re-org to compact the tables. Running RUNSTATS will ensure that the DB2 catalog statistics about table and column sizes and accesses are most current so that the best access patterns are chosen by the optimizer.
- You will have to define more connections called threads in DB2. WebSphere Application Server uses a lot of threads. Sometimes this is the source of throughput bottlenecks since the server will wait at the create thread until one is available.
- Make sure you are current with JDBC maintenance. Many performance improvements have been made to JDBC. To determine the JDBC maintenance level, enter the following from the shell:

  ```
  java  com.ibm.db2.jcc.DB2Jcc  -version
  ```

  If this returns a class not found, either you are at a level of the driver that is older and doesn't support this command or you have not issued the command properly.
- We recommend that you enable dynamic statement caching in DB2. To do this, modify your ZPARMS to say `CACHEDYN(YES) MAXKEEPD(16K)`. Depending on the application, this can make a very significant improvement in DB2 performance. Specifically, it can help JDBC and LDAP query.
- Increase DB2 checkpoint interval settings to a large value. To do this, modify your ZPARMS to include `CHKFREQ=xxxxx`, where xxxxx is set at a high value when doing benchmarks. On production systems there are other valid reasons to keep checkpoint frequencies lower, however.

**Example:** This example identifies zparm values discussed in this article.

```
//DB2INSTE   JOB MSGCLASS=H,CLASS=A,NOTIFY=IBMUSER
/*JOBPARM SYSAFF=*
//*****************************************************************
```

```
//* JOB NAME = DSNTIJUZ
//*
//* DESCRIPTIVE NAME = INSTALLATION JOB STREAM
//*
//*    LICENSED MATERIALS - PROPERTY OF IBM
//*    5675-DB2
//*    (C) COPYRIGHT 1982, 2000 IBM CORP.  ALL RIGHTS RESERVED.
//*
//*    STATUS = VERSION 7
//*
//* FUNCTION = DSNZPARM AND DSNHDECP UPDATES
//*
//* PSEUDOCODE =
//*   DSNTIZA  STEP  ASSEMBLE DSN6.... MACROS, CREATE DSNZPARM
//*   DSNTIZL  STEP  LINK EDIT DSNZPARM
//*   DSNTLOG  STEP  UPDATE PASSWORDS
//*   DSNTIZP  STEP  ASSEMBLE DSNHDECP DATA-ONLY LOAD MODULE
//*   DSNTIZQ  STEP  LINK EDIT DSNHDECP LOAD MODULE
//*   DSNTIMQ  STEP  SMP/E PROCESSING FOR DSNHDECP
//*
//* NOTES = STEP DSNTIMQ MUST BE CUSTOMIZED FOR SMP.  SEE THE NOTES
//*         NOTES PRECEDING STEP DSNTIMQ BEFORE RUNNING THIS JOB.
//*
//*  LOGLOAD=16000000,
//*********************************************************************/
//*
//DSNTIZA EXEC PGM=ASMA90,PARM='OBJECT,NODECK'
//STEPLIB DD DSN=ASM.SASMMOD1,DISP=SHR
//SYSLIB   DD  DISP=SHR,
//         DSN=DB2710.SDSNMACS
//         DD  DISP=SHR,
//         DSN=SYS1.MACLIB
//SYSLIN   DD  DSN=&LOADSET(DSNTILMP),DISP=(NEW,PASS),
//             UNIT=SYSALLDA,
//             SPACE=(800,(50,50,2)),DCB=(BLKSIZE=800)
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSUT1   DD  UNIT=SYSALLDA,SPACE=(800,(50,50),,,ROUND)
//SYSUT2   DD  UNIT=SYSALLDA,SPACE=(800,(50,50),,,ROUND)
//SYSUT3   DD  UNIT=SYSALLDA,SPACE=(800,(50,50),,,ROUND)
//SYSIN    DD  *
    DSN6ENV    MVS=XA
    DSN6SPRM   RESTART,                 X
               .
               .
               .
               AUTH=YES,
                                               X
               AUTHCACH=1024,                                       X
               BINDNV=BINDADD,                                      X
               BMPTOUT=4,                                           X
               CACHEDYN=YES,
                                         X
               .
               .
               .
               MAXKEEPD=16000,
                                       X
               .
               .
               .
   DSN6ARVP    ALCUNIT=CYL,                                         X
               .
               .
               .
   DSN6LOGP    DEALLCT=(0),                                         X
               .
```

```
             .
             .
   DSN6SYSP    AUDITST=NO,                                     X
               BACKODUR=5,                                     X
              CHKFREQ=16000000,
                                    X
               CONDBAT=400,                                    X
               CTHREAD=1200,                                   X
               DBPROTCL=PRIVATE,                               X
               DLDFREQ=5,                                      X
               DSSTIME=5,                                      X
               EXTRAREQ=100,                                   X
               EXTRASRV=100,                                   X
               EXTSEC=NO,                                      X
               IDBACK=1800,
                                         X
               .
               .
               .
             //*
```

## WebSphere Application Server tuning tips for use with DB2

WebSphere Application Server uses JDBC prepared statement caching as a performance enhancing feature. If you are using this feature together with DB2 for z/OS, be aware of the potential impact on the number of DB2 JDBC cursor objects available.

### Prepared statement caching effects on DB2 for OS/390 JDBC cursor objects

When you obtain a *ResultSet* object by running a *PreparedStatement* object, a DB2 JDBC cursor object is bound to it until the corresponding DB2 prepared statement is closed. This happens when the DB2 *Connection* object is released from the WebSphere Application Server connection pool. From an application perspective, the result set, prepared statement, and connection are each closed in turn. However, the underlying DB2 *Connection* is pooled by the WebSphere Application Server, the underlying DB2 *PreparedStatement* is cached by the application server, and each underlying DB2 JDBC cursor object associated with each *ResultSet* created on this *PreparedStatement* object is not yet freed.

Each *PreparedStatement* object in the cache can have one or more result sets associated with it. If a result set is opened and not closed, even though you close the connection, that result set is still associated with the prepared statement in the cache. Each of the result sets has a unique JDBC cursor attached to it. This cursor is kept by the statement and is not released until the prepared statement is cleared from the WebSphere Application Server cache.

If there are more of the cached statements than there are cursors, eventually the execution of a *PreparedStatement* object results in the following exception:

```
java.sql.SQLException: DB2SQLJJDBCProfile Error: No more JDBC Cursors without hold
```

Some WebSphere Application Server tuning suggestions to help avoid this problem are:

1. Decrease the *statement cache size* setting on the DB2 for OS/390 data source definition. Setting this value to zero (0) eliminates statement caching, but causes a noticeable performance impact.
2. Decrease the *minConnections* connection pool setting on the DB2 for OS/390 data source definition.
3. Decrease the *Aged Timeout* connection pool setting on the DB2 for OS/390 data source definition. However, it is **NOT** recommended that you set this to zero (0), as this disables the Aged Timeout function.

## Tuning MDB processing on z/OS

To tune MDB processing, you consider and act on a variety of settings together. There is a wide range of values that you can select, and possibilities to consider, because of the variety of workloads possible to run in any given server.

When a message-driven bean is mapped (that is, listening to) a queue, or to a topic through a durable subscription, a JMS message first enters into the WebSphere server in the controller, so we say the server is “listening in the controller” for these messages. The “listening in the controller” term is used throughout this description of tuning MDB processing.

If you want to optimize the processing of messages by message-driven beans, use this task to consider and act on the associated settings.

Tuning MDB processing in the server is a part of the greater task of tuning the server’s entire workload. This needs consideration of a variety of settings and the interactions between those settings.

To tune MDB processing, the following settings must be considered together: WLM service class definitions, WebSphere workload profile selection, Message Listener Service Listener Port settings, JMS Connection Factory pooling settings, and WebSphere MQ Queue Manager settings.

It is difficult to give one recommendation about what values to select for each of these settings, given the variety of workloads possible to run in any given server. There are a wide variety of possibilities to take into consideration, including the following considerations:
- The number of message-driven beans.
- The administrative configuration choices, such as whether to map two message-driven beans to the same or different listener ports.
- Different goals regarding the importance of work for message-driven beans relative to other (HTTP, IIOP) types of work running in the server.

The suggested settings below provide a starting point, under the assumption that the server is configured with only a single application consisting of a single message-driven bean installed and running on this server.

The starting point settings are followed by more detailed discussions that explain the rationale behind the suggestions, and describe the listener port function in more detail in the “listening in the controller” case on z/OS. Together they can help you to understand the function and the settings enough to make your own setting selections for your own systems and servers.

1. 1. Set the Listener Port maximum sessions property to at least twice the maximum number of servant worker threads available to the entire server. The value of this property determines the high threshold value (high threshold = maximum sessions) and is used by the throttle to decide when to block or allow requests.
   a. To view this administrative console page, click **Servers-> Application Servers->** ***application_server*****-> Message Listener Service-> Listener Ports->** ***listener_port***
   b. Set the maximum sessions property to the value you want the MDB throttle to use as its high threshold value.

      The suggested minimum value is computed by the formula:

      ```
      2 * (maximum number of servants) * (number of worker threads in one servant)
      ```

      Here “servants” means the same as “server instances” as the setting is described on the administrative console. To calculate the number of worker threads in a single servant, see the description of “Workload profile”. See the *Developing and deploying applications* PDF for more information.

   For further considerations setting the Listener Port maximum sessions property, see “Concepts and considerations for MDB settings on z/OS” on page 8.
2. Set the WebSphere MQ Queue Connection Factory properties.
   a. To view this administrative console page, click **Resources-> WebSphere MQ JMS Provider-> WebSphere MQ Queue Connection Factories (under additional properties)**
   b. Select the Connection Factory specified for the Listener Port.
   c. Under the Additional Properties, activate the Connection Pool panel.

d. Set the Max Connections property for the Connection Pool. Allow one connection for each message-driven bean. This property value could include message-driven beans mapped onto different listener ports, if those listener ports were each, in turn, mapped onto the same connection factory. For more detailed discussions about considerations that affect this setting, see "Concepts and considerations for MDB settings on z/OS."

e. Under Additional Properties of the Connection Factory, activate the Session Pool panel.

f. Set the Max Connections property for the Session Pool. Allow one session for each worker thread in a single servant. This property should be set to at least the number of worker threads available to a single servant. For more detailed discussions about considerations that affect this setting, see "Concepts and considerations for MDB settings on z/OS."

3. Set the WebSphere MQ-related properties. Make sure that the backing WebSphere MQ queue manager has been configured with enough resources to support the intended JMS workload coming from WebSphere Application Server (and other clients). In particular, you may want to consider your queue manager's CTHREAD, IDBACK, and IDFORE parameter settings. For more information on these WebSphere MQ settings, see the WebSphere MQ books:

   WebSphere MQ System Setup Guide (SC34-6052-02).

   WebSphere MQ Script Command (MQSC) Reference Book (SC34-6055-03)

   WebSphere MQ Problem Determination Guide (GC34-6054-01).

1. Suppose your server is configured with the maximum server instances value set to 3, (whatever the minimum number may be). Also suppose that your workload profile is LONGWAIT, which means each servant contains 40 worker threads.

   In this case, set your listener port maximum sessions value to at least 240 = 2 * 3 * 40.

2. Suppose that your application contains two individual message-driven beans, each of which has an onMessage() implementation that forwards the message to another JMS destination. Therefore, each message-driven bean needs its own JMS connection factory to perform this task. Suppose the Administrator has mapped each MDB's JMS connection factory resource reference onto the same administratively-defined connection factory used by the listener port that each of these message-driven beans is mapped onto.

   In this case, you need to set the connection factory's Connection Pool Max Connections value to 42. One connection for each of the two message-driven beans to be used by the listener port, and one connection potentially for each of the 40 onMessage() dispatches that night be running concurrently. (Remember that the connection pool is a per-servant pool).

3. Set the connection factory's Session Pool Max Connections to 40, the number of worker threads in a single servant, regardless of the number of servants.

For debugging tips, refer to "Best practices for debugging MDB throttle support" on page 16.

## Concepts and considerations for MDB settings on z/OS

To be able to tune the processing of endpoint message-driven beans (MDBs), you need to understand the concepts and considerations for the MDB settings that you configure.

When a message-driven bean is mapped (that is, listening to) a queue, or to a topic through a durable subscription, a JMS message first enters into the WebSphere server in the controller, so we say the server is "listening in the controller" for these messages. The "listening in the controller" term is used throughout this description of tuning MDB processing.

The following sub-topics provide a set of information that describe the concepts and considerations that you should be aware of to be able to configure MDB settings on z/OS:

- "Basic "listening in the controller" messaging flow" on page 9

  When a message-driven bean is mapped (that is, listening to) a queue, or to a topic through a durable subscription, a JMS message first enters into the WebSphere server in the controller, so we say the server is "listening in the controller" for these messages. When a message arrives, it flows through a sequence of events.

- "MDB throttle"

  On z/OS, the "MDB throttle" is used to control the amount of work that the server processes at a given time for a message-driven bean. The MDB throttle limits how far the message listener port will "read ahead" to try to ensure that the work request queue does not have a backlog of messages to be processed.
- "MDB throttle settings for message-driven beans on z/OS" on page 10

  You can tune a variety of settings for the "MDB throttle", to control the amount of MDB work that the server processes at a given time.
- "Connection factory settings for message-driven beans on z/OS" on page 12

  You can tune a variety of connection factory settings to control the creation of connections and sessions for MDB work.
- "Message-driven beans, heterogeneous workloads, and workload management on z/OS" on page 14

  A message-driven bean can run on an application server that hosts a heterogeneous workload including other message-driven beans and non-MDB work items. To manage a heterogeneous workload on z/OS, you should use WLM classification and define unique service classes for different-priority work running in the same server.

***Basic "listening in the controller" messaging flow:***

When a message-driven bean is mapped (that is, listening to) a queue, or to a topic through a durable subscription, a JMS message first enters into the WebSphere server in the controller, so we say the server is "listening in the controller" for these messages. When a message arrives, it flows through a sequence of events.

When a message has arrived on a JMS destination (queue or topic) on which a message-driven bean running is listening, the following sequence of events takes place:
1. The WebSphere MQ JMS queue agent thread (running in the controller) browses the JMS destination. The queue agent identifies the application server on which a message-driven bean is running for that destination. The queue agent calls a Message Reference Handler (MRH) registered by the application server, to tell the server that a message has arrived on a destination on which one of its message-driven beans is listening. A message reference corresponding to this message is sent along with the call to the MRH.
2. If the throttle high threshold value (see below) has not been exceeded, a work request corresponding to the just-browsed message is queued onto the WLM queue in the controller.
3. The work request is dispatched to an individual servant. A ServerSession is obtained from the appropriate ServerSessionPool in that servant and, using the message reference as index, the message is destructively consumed from the JMS destination.
4. The MDB application onMessage(Message) method is dispatched with the consumed message.
5. When the onMessage(Message) method finishes, the servant notifies the controller that the work record has completed.

***MDB throttle:***

On z/OS, the "MDB throttle" is used to control the amount of work that the server processes at a given time for a message-driven bean. The MDB throttle limits how far the message listener port will "read ahead" to try to ensure that the work request queue does not have a backlog of messages to be processed.

The preprocessing, classification, building of and queuing of a work record to prepare for dispatching a specific message-driven bean is a relatively basic operation, especially when compared to the actual business logic of the message-driven bean and the container infrastructure on the application dispatch path. When the rate of messages arriving reaches occasional high volumes, or peaks, the controller can preprocess many more messages than the back-end servants can process by running the associated MDB application. The result of these peaks (in asynchronous work) is that the WLM work request queue becomes backed up waiting for worker threads (in the servants) that have a backlog of messages to be processed.

A backlog of messages to be processed can also occur as a result of situations where the scalable server is taken out of service for a period of time. This causes messages to build up on the JMS destination waiting for the server to restart. When the server does restart, there is a flood of new work introduced into the server.

The MDB throttle limits how far the message listener port will “read ahead” down the JMS queue (or topic), to try to ensure that the work request queue does not have a backlog of messages to be processed.

***MDB throttle settings for message-driven beans on z/OS:***

You can tune a variety of settings for the “MDB throttle”, to control the amount of MDB work that the server processes at a given time.

This topic describes the following concepts and considerations that affect your choice of MDB throttle settings:
- “MDB throttle – high and low thresholds”
- “MDB throttle – tuning considerations” on page 11
- “MDB throttle – alternative tuning considerations” on page 11
- “MDB throttle example” on page 11

**MDB throttle – high and low thresholds**

The MDB throttle support maintains a count of the current number of in-flight messages for each listener port.

When a message reference is sent to the MRH, as described in “Basic “listening in the controller” messaging flow” on page 9, the in-flight count is incremented by 1. Next, the in-flight count is compared against the high threshold value for this listener port.
- If the in-flight count is less than or equal to the high threshold value, then a work record is queued up onto the WLM queue.
- If the in-flight count exceeds the high threshold value, the queue agent thread that is running the MRH becomes blocked, entering a wait state, and we say that the throttle is “blocking”.

The in-flight count is decremented by 1 whenever the controller is notified that a work record for this Listener Port has completed (whether or not that the application’s transaction was committed). After being decremented, the in-flight count is checked against the low threshold value for this Listener Port. If the in-flight count drops down to the low threshold value, the previously-blocked queue agent thread is awoken (notified). At that point new work records can be queued onto the WLM queue and we say the throttle has been “released”.

The low threshold and high threshold values are set externally by one setting, the listener port’s Maximum sessions parameter. The high threshold value is set internally equal to the Maximum sessions value defined externally. The low threshold value is computed and set internally by the formula: `low threshold = (high threshold / 2)`, with the value rounding down to the nearest integer.

However, if the Message Listener service has been configured with the Custom Property of MDB.THROTTLE.THRESHOLD.LOW.EQUALS.HIGH defined and set to a value of “true”, then the low threshold value is set internally to the high threshold value (which is the externally-set Maximum sessions property of the listener port).

**Note:** Please consider the following information, although most users are not be affected by what is described. One queue agent thread is established for each destination, rather than for each listener port. Therefore, it two listener ports are mapped onto the same queue, a throttle-blocking condition on one listener port also results in a blocking of the second listener port's queueing of work records. The second listener port is blocked even if it has not reached its high threshold value. For simplicity it is recommended that you do not share a destination across more than one listener port.

### MDB throttle – tuning considerations

Consider the reason for recommending that you set the listener port "Maximum sessions" value to twice the number of worker threads available in all servants in the scalable server (2*WT). The reason is based on the goal that if you have an available worker thread in some servant, then you do not want to leave it idle because of a blocked MDB throttle. That is, you do not want to have an empty WLM queue, an available servant worker thread, and a blocked throttle.

With the basic recommendation of using the 2*WT value, then, a blocked throttle is released at the moment when the following conditions are true:

- There is one free servant worker thread
- There is nothing on the WLM queue
- There is one message reference browsed but for which a work request was not added to the WLM queue (the throttle blocked instead)

Furthermore, by setting the high threshold to 2*(WT+N) you can ensure that, at the moment a servant worker thread frees up and releases the throttle, there is a backlog of N messages pre-processed and sitting on the WLM queue ready for dispatch. A very high value would introduce the WLM queue overload problem which the throttle was introduced to avoid, but we do not have a specific upper limit to recommend here. (These tuning considerations assume the queue (or topic) is fully-loaded with messages to be processed.)

So, raising the high threshold value allows the server to create a small backlog of preprocessed messages sitting on the WLM queue if a workload spike occurs. The downside or negative of raising the high threshold value is that it increases the chance that a work record for a given message might time out before the application can be dispatched with the given message. (That is, the server might reach the MDB Timeout limit.) The given message is eventually re-delivered to the server, but only later and the processing done up until that time would be wasted. Also, a very large high threshold value would effectively bypass the MDB throttle function, in which case the WLM queue could be overloaded; this would cause the server to fail.

### MDB throttle – alternative tuning considerations

Although the scalable server was designed with the goal of maximizing throughput, it is possible to use the listener port settings to achieve other workflow management goals.

For example, a high threshold setting of '1' guarantees that messages are processed in the order that they are received onto the destination.

There might also be other business reasons, based on capacity or other factors, to restrict a particular listener port to much less concurrency than the server would otherwise support. While this is certainly a supported configuration, it might cause the throttle to block when there are idle worker threads available.

### MDB throttle example

Suppose your server is configured with the maximum server instances value set to 3, with workload profile of IOBOUND. You have two CPUs, therefore WebSphere Application server will create six worker threads in each servant. Your application (a single MDB mapped to a queue) handles each message relatively quickly (so there is less risk of timeout) and you want the total time from arrival of a given message on your MDB queue until the end of MDB dispatch for this message to be as small as possible.

To provide a quick response time in the case of a surge in work, you opt for a bigger backlog. You set your Listener Port maximum sessions value to `100 =  2 * (3 * 6 + 32)`.

**Note:** Any value greater than or equal to `36 = 2 * 3 * 6` would keep all available servant worker threads busy. In practice it would probably not be worth the effort to pick the best possible "backlog factor". Therefore, we make a mental estimate after giving the situation some thought, and pick a value so that we end up with the round number, 100.

***Connection factory settings for message-driven beans on z/OS:***

You can tune a variety of connection factory settings to control the creation of connections and sessions for MDB work.

This topic describes the following concepts and considerations that affect your choice of connection factory settings:
- "Connection factory settings"
- "Connection pool maximum connections settings"
- "Session pool maximum connections settings" on page 13
- "Should I use a few or many connection factories?" on page 13
- "Connection factories – examples" on page 14

**Connection factory settings**

To attach an application and a server to a particular queue manager with authentication parameters, applications and message listener ports are both bound to connection factories. The server uses the same administrative model to listen for messages arriving for delivery to message-driven beans as the application uses to exploit JMS, in that message listener ports are bound to a queue connection factory or topic connection factory and a corresponding queue or topic.

In addition to specifying the messaging provider settings (for example, queue manager settings) on connection factories, you also specify connection and session pool properties. In particular, for a connection factory, the values of the connection pool property Maximum connections and session pool property Maximum connections must be chosen with different considerations in mind, depending on whether you are using the connection factory for a listener port, for an application, or for both.

**Connection pool maximum connections settings**

First, in order to avoid waiting for a JMS connection, because of reaching a WebSphere Application Server-defined limit, the connection pool Maximum connections should be set to at least the sum of the following values:
1. One for each message-driven bean that is mapped to any listener port mapped onto this connection factory.
2. The maximum number of connections for one dispatch multiplied by the number of work threads in a servant.

   To determine this maximum connection count, first find the maximum number of connections obtained in a single application dispatch (typically '1'), checking all applications that use this connection factory. Multiply this maximum number of connections by the number of worker threads in a servant. (See the following note for more information about this value.)

Note that only a single servant's worker threads are counted here, because each servant gets its own connection factory with connection and session pools, although the Administrator defines only a single set of property values.

**Clarification about part 2 of the preceding Maximum connection count**

**Note:**

> A message-driven bean might or might not get a JMS connection in performing its application logic in onMessage(). For example, it might get a connection to forward the message to another destination or send a reply, or it might simply update some log and perform no JMS-related calls of its own.
>
> In either case, we need to count 'one' for this message-driven bean in part 1 of the preceding Maximum connection count, as this is the connection used by the listener port. If the message-driven bean onMessage() logic gets one JMS connection, then we then add the number of servant worker threads to the Maximum connection count. If not, then we need not add to the Maximum connection count on behalf of this (MDB) application component.
>
> Of course, other non-MDB application components which use this connection factory to perform JMS calls might cause the administrator to need to include part 2 of the overall Maximum connections count. But, regardless of how many MDB or non-MDB application components use this connection factory, if they each only use one JMS connection per dispatch, then the count in part 2 is only equal to the number of servant worker threads (not the number of applications multiplied by the number of servant worker threads).

**Session pool maximum connections settings**

This calculation is an easier one to make than it might first appear. For MDB listener port processing in all typical cases, the session pool Maximum connections property should be set to the number of worker threads in a single servant.

The reasons are the fact that JMS sessions are not shared across application dispatches or by the listener port infrastructure, even for clients of the same connection factory, along with the fact that there is a unique session pool for each JMS connection obtained from the connection factory (although the pool property settings are specified only once, at the connection factory level).

It is possible to imagine a case for which the same connection factory is used both by a listener port and an application, with the application having a higher Maximum connections requirement on the session pool setting, but this does not merit further discussion here.

**Important:** : It is possible to restrict the concurrent MDB work in a server by setting this session pool Maximum connections to less than the number of servant worker threads. This is not recommended. In such a case, an MDB work request could be dispatched over to a servant, without a session available to process the message. The worker thread at that point would then wait for a session to become available, tying up the valuable worker thread resource.

**Should I use a few or many connection factories?**

Some users may prefer to keep these calculations simple; for example, by creating a separate connection factory for each message-driven bean (in which case the connection pool Maximum connections property value could simply be set to 1). Others may prefer to manage fewer connection factory administrative objects.

The fact that the connections and sessions used for MDB processing cannot be shared (that is, cannot be used by more than one flow at a time) means that the desire to take advantage of pooling should not be considered a reason for using fewer connection factories. In other words, adding another connection factory does not prevent connection pooling that could otherwise be exploited by MDB listener port processing.

**Connection factories – examples**

**Example 1**

**Note:**

The scenario:

- Each servant has 12 worker threads. (The number of servants in the server is not important since each servant gets its own pools).
- Listener port LP1 is mapped to connection factory CF1. Message-driven bean MDB1 is mapped to LP1. The onMessage() application code of MDB1 puts a new message onto a forwarding queue, and so MDB1 has a resource reference which is also resolved to CF1.
- Also, in the same server, listener port LP2 is defined and mapped to connection factory CF2. Message-driven beans MDB2A and MDB2B are defined in the same ejb-jar file and are both mapped to LP2 with complementary JMS selectors. The onMessage() application code of MDB2A and MDB2B each does some logging, but neither message-driven bean makes any JMS API calls of its own.

The solution:

- For connection factory CF1, we count one for MDB1. The MDB1 application (which also uses connection factory CF1 to send its forwarding message) uses one JMS connection, for which we count the number of worker threads (12), multiplied by one. Our total connection pool Maximum connections for connection factory CF1, then, is `13 = 12 + 1`.
- For connection factory CF2, we count one for each of MDB2A and MDB2B. There are no applications using CF2, (only the listener port infrastructure), so we set the connection pool Maximum connections for connection factory CF2 equal to 2.
- For each of the two connection factories, the session pool Maximum connections value is set to 12.

**Example 2**

**Note:**

The scenario:

- Again, each servant has 12 worker threads. In this example we only want to use a single connection factory, CF1.
- Each of two listener ports LP1 and LP2 is mapped to connection factory CF1. The message-driven beans MDB1, MDB2, and MDB3 are part of three unique application EAR files. MDB1 is mapped to LP1, but MDB2 and MDB3 are each mapped to LP2.

The solution:

- Up to this point we count that we need three connections for connection factory CF1. However, there is also a servlet component which puts messages on a queue and it uses the same connection factory CF1. So for connection factory CF1, the connection pool Maximum connections setting is `15 = 3 + 12`.

***Message-driven beans, heterogeneous workloads, and workload management on z/OS:***

A message-driven bean can run on an application server that hosts a heterogeneous workload including other message-driven beans and non-MDB work items. To manage a heterogeneous workload on z/OS, you should use WLM classification and define unique service classes for different-priority work running in the same server.

In the set of topics under "Concepts and considerations for MDB settings on z/OS" on page 8, the explanations assume, for simplicity, that your server hosts a single message-driven bean, and that several MDB instances could be simultaneously running on all servant worker threads. Of course, in reality it is likely that any message-driven bean runs on an application server that hosts a heterogeneous workload including other message-driven beans, enterprise beans accessed through IIOP, work such as servlets and JSPs accessed through HTTP, and more work items.

Although the various MDB-related tuning controls provide a way to exert a fine-grained control over the amount of MDB work performed for a given message-driven bean or set of message-driven beans in a given server, we do not recommend using these settings to prioritize certain MDB work above or below other work in the server. Instead, we recommend using WLM classification and defining unique service classes for different-priority work running in the same server. Be sure to allow for at least one servant per unique service class.

This introduces complications in the case that work moving through a single listener port is configured with different selectors mapping onto different service classes. For more information about workload classification, see Classifying z/OS workload.

There is only one throttle for the listener port. This throttle controls the rate at which differently-prioritized messages get queued up as work records on to the WLM queue. The prioritization only occurs with the help of workload management, when the work records are queued up. It does not apply to the phase in which the messages are browsed by the queue agent thread and work records are queued up.

One strategy you can take in such a case is to raise the high threshold value (the value of the Maximum sessions property of the listener port) above the baseline recommendation of "twice the combined number of worker threads in all the server's servants". The rationale would be to make sure the WLM queue is loaded enough to allow workload management to decide which work to process next, rather than cutting off the queuing up of work requests. (Cutting off the queuing up of work requests can leave an overabundance of lower-priority work records on the WLM queue when higher priority messages are waiting to be browsed.)

You can develop other strategies beyond what can be discussed here.

Finally, it is also possible that both the following values can be known to some degree of certainty:

- The average number of servant worker threads processing a given message-driven bean
- The average number of available servants (some number between the minimum and maximum is started at any given time)

This could be computed perhaps using PMI, other monitoring tools, or perhaps even by a high-level understanding of how the message-driven bean fits into a greater application flow within a given server.

Should you adjust the baseline formula setting listener port maximum sessions to become twice the number of worker threads available for the *maximum number* of servants in the scalable server instead of twice the actual number of worker threads available in all servants? Although it is hard to give general recommendations, it may not be worth changing the formula. Lowering the setting introduces the possibility of idle worker threads, and a setting higher than strictly necessary only results in an extra buildup of messages on the WLM queue. The number of extra messages on the queue should still be a small enough number to prevent the serious problem of an overloaded WLM queue causing the server to fail.

## Best practices for debugging MDB throttle support

In order to have minimal impact on performance while still collecting needed debugging information for the MDB throttle support, use the following options:

- Trace option: com.ibm.ejs.jms.listener.MessageReferenceListenerPort=all=enabled

  The above trace option does not need to be specified if the MDB trace option is already set.
- System property: com.ibm.mdb.throttle.trace.enabled

  Disabled, if the property is not defined or set to 0.

  Enabled, if the property is set to 1.
- Dynamic trace support for statistics gathering and presentation.

  To enable, disable, or reset messaging statistics, use the following modify command:

  ```
  f <server>,mdbstats,[enable | disable | reset]
  ```

  The response from the console should be:

  ```
  BBOO0211I MODIFY COMMAND MDBSTATS, [ENABLE | DISABLE | RESET] COMPLETED SUCCESSFULLY
  ```

  The following message appears if statistics gathering is not enabled:

  ```
  BBOO0284I  STATISTICS GATHERING NOT ENABLED FOR <string>
  ```

  To display statistics, use the following modify display command:

  ```
  f <server>,display,work,mdb,stats
  ```

  The displayed information per listener port includes:

  **NAME** The name of the Listener Port for which the statistics are being displayed.

  **TIME** The amount of time, in seconds, since the stats were enabled or reset.

  **TOTAL**
  Total number of message references browsed since the stats were enabled or reset.

  **IN-FLIGHT**
  The current number of in-flight Work Requests.

  **EXCS** The number of exceptions while queuing the requests.

  **BLOCKS**
  The total number of instances the Throttle restricts queuing of work requests.

# Chapter 2. How do I tune performance?

Hold your cursor over the task icon () to see a description of the task. The task preview feature is unavailable for Mozilla Web browsers.

| | |
|---|---|
| **Tune the application serving environment** | Documentation |
| **Obtain tuning advice: Performance advisor** | Documentation |
| **Obtain tuning advice: TPV advisor** | Documentation |
| **Tune WebSphere applications** | Documentation |
| **Pool database connections** | Documentation |
| **Configure caching** | Documentation |

## Legend for ″How do I?...″ links

| Detailed steps | Show me | Tell me | Guide me | Teach me |
|---|---|---|---|---|
| Refer to the detailed steps and reference | Watch a brief multimedia demonstration | View the presentation for an overview | Be led through the console pages | Perform the tutorial with sample code |
| **Approximate time:** Varies | **Approximate time:** 3 to 5 minutes | **Approximate time:** 10 minutes+ | **Approximate time:** 1/2 hour+ | **Approximate time:** 1 hour+ |

# Chapter 3. Planning for performance

How well a Web site performs while receiving heavy user traffic is an essential factor in the overall success of an organization. This section provides online resources that you can consult to ensure that your site performs well under pressure.

- Consult the following Web resources for learning.

  **IBM Patterns for e-Business**

  IBM Patterns for e-business is a group of reusable assets that can help speed the process of developing Web-based applications. The patterns leverage the experience of IBM architects to create solutions quickly, whether for a small local business or a large multinational enterprise.

  **Planning for availability in the enterprise**

  Availability is an achievable service-level characteristic that every enterprise struggles with. The worst case scenario is realized when load is underestimated or bandwidth is overloaded because availability planning was not carefully conducted. Applying the information in this article and the accompanying spreadsheet to your planning exercises can help you avoid such a scenario.

  **Hardware configurations for WebSphere Application Server production environments**

  This article describes the most common production hardware configurations, and provides the reasons for choosing each one. It begins with a single machine configuration, and then proceeds with additional configurations that have higher fault tolerance, horizontal scaling, and a separation of Web and enterprise bean servers.

- See the documentation for the product functionality to improve performance .

## Application design consideration

This topic describes the architectural suggestions in design and how to tune applications.

Consult the Designing applications topic in the *Developing and deploying applications* PDF, which highlights Web sites and other ideas for finding best practices for designing WebSphere applications, particularly in the realm of WebSphere extensions to the Java 2 Platform, Enterprise Edition (J2EE) specification.

The Designing applications topic in the *Developing and deploying applications* PDF contains the architectural suggestions in design and the implementation of applications. For existing applications, the suggestions might require changing the existing implementations. Tuning the application server and resource parameters can have the greatest effect on performance of the applications that are well designed.

**best-practices:** Use the following information as an architectural guide when implementing applications:

- Persistence
- Model-view-controller pattern
- Statelessness
- Caching
- Asynchronous considerations
- Third-party libraries

**Persistence**

Java 2 Platform, Enterprise Edition (J2EE) applications load, store, create, and remove data from relational databases, a process commonly referred to as *persistence*. Most enterprise applications have significant database access. The architecture and performance of the persistence layer is critical to the performance of an application. Therefore, persistence is a very important area to consider when making architectural choices that require trade-offs related to performance. This guide recommends first focusing on a solution that has clean architecture. The clean architecture considers data consistency, security, maintenance, portability, and the performance of that solution. Although this approach might not yield the absolute peak performance obtainable from manual coding a solution that ignores the mentioned qualities of service, this approach can achieve the appropriate balance of data consistency, maintainability, portability, security, and performance.

Multiple options are available in J2EE for persistence: Session beans using entity beans including container-managed persistence (CMP) or bean-managed persistence (BMP), session beans using Java Database Connectivity (JDBC), and Java beans using JDBC. For the reasons previously mentioned, consider CMP entity persistence because it provides maximum security, maintenance, and portability. CMP is also recommended for good performance. Refer to the Tune the EJB container section of the Tuning application servers topic on tuning enterprise beans and more specifically, CMP.

If an application requires using enterprise beans not using EJB entities, the persistence mechanism usually involves the JDBC API. Because JDBC requires manual coding, the Structured Query Language (SQL) that runs against a database instance, it is critical to optimize the SQL statements that are used within the application. Also, configure the database server to support the optimal performance of these SQL statements. Finally, usage of specific JDBC APIs must be considered including prepared statements and batching.

Regardless of which persistence mechanism is considered, use container-managed transactions where the bean delegates management of transactions to the container. For applications that use JDBC, this is easily achieved by using the session façade pattern, which wraps all JDBC functions with a stateless session bean.

Finally, information about tuning the connection over which the EJB entity beans or JDBC communicates can be found in the Tune the data sources section of the Tuning application servers topic.

**Model-view-controller pattern**

One of the standard J2EE programming architectures is the model-view-controller (MVC) architecture, where a call to a controller servlet might include one or more child JavaServer Pages (JSP) files to construct the view. The MVC pattern is a recommended pattern for application architecture. This pattern requires distinct separation of the view (JSP files or presentation logic), the controller (servlets), and the model (business logic). Using the MVC pattern enables optimization of the performance and scalability of each layer separately.

**Statelessness**

Implementations that avoid storing the client user state scale and perform the best. Design implementations to avoid storing state. If state storage is needed, ensure that the size of the state data and the time that the state is stored are kept to the smallest possible values. Also, if state storage is needed, consider the possibility of reconstructing the state if a failure occurs, instead of guaranteeing state failover through replication.

Specific tuning of state affects HTTP session state, dynamic caching, and enterprise beans. Refer to the follow tuning guides for tuning the size, replication, and timing of the state storage:
- “Session management tuning” on page 79
- “EJB Container tuning” on page 85
- “Tuning dynamic cache with the cache monitor” on page 147

**Caching**

Most J2EE application workloads have more read operations than write operations. Read operations require passing a request through several topology levels that consist of a front-end Web server, the Web container of an application server, the EJB container of an application server, and a database. WebSphere Application Server provides the ability to cache results at all levels of the network topology and J2EE programming model that include Web services.

Application designers must consider caching when the application architecture is designed because caching integrates at most levels of the programming model. Caching is another reason to enforce the MVC pattern in applications. Combining caching and MVC can provide caching independent of the presentation technology and in cases where there is no presentation to the clients of the application.

Network designers must consider caching when network planning is performed because caching also integrates at most levels of the network topology. For applications that are available on the public Internet, network designers might want to consider Edge Side Include (ESI) caching when WebSphere Application Server caching extends into the public Internet. Network caching services are available in the proxy server for WebSphere Application Server, WebSphere Edge Component Caching Proxy, and the WebSphere plug-in.

**Asynchronous considerations**

J2EE workloads typically consist of two types of operations. You must perform the first type of operation to respond to a system request. You can perform the second type of operation asynchronously after the user request that initiated the operation is fulfilled.

An example of this difference is an application that enables you to submit a purchase order, enables you to continue while the system validates the order, queries remote systems, and in the future informs you of the purchase order status. This example can be implemented synchronously with the client waiting for the response. The synchronous implementation requires application server resources and you wait until the entire operations complete. If the process enables you to continue, while the result is computed asynchronously, the application server can schedule the processing to occur when it is optimal in relation to other requests. The notification to you can be triggered through e-mail or some other interface within the application.

Because the asynchronous approach supports optimal scheduling of workloads and minimal server resource, consider asynchronous architectures. WebSphere Application Server supports asynchronous programming through J2EE Java Message Service (JMS) and message-driven beans (MDB) as well as asynchronous beans that are explained in the Tuning Java Message Service and Tuning MDB topics.

**Third-party libraries**

Verify that all the libraries that applications use are also designed for server-side performance. Some libraries are designed to work well within a client application and fail to consider server-side performance concerns, for example, memory utilization, synchronization, and pooling. It is suggested that all libraries that are not developed as part of an application undergo performance testing using the same test methodologies as used for the application.

Additional reference:

IBM WebSphere Developer Technical Journal: The top 10 (more or less) J2EE best practices

Improve performance in your XML applications, Part 2

# Chapter 4. Taking advantage of performance functions

This topic highlights a few main ways you can improve performance through a combination of product features and application development considerations.

- Use this product functionality to improve performance.

  **Balancing workloads with clusters**

  Clusters are sets of servers that are managed together and participate in workload management. The servers that are members of a cluster can be on different host machines, as opposed to the servers that are part of the same node and must be located on the same host machine. A cell can have no clusters, one cluster, or multiple clusters.

  **Using the dynamic cache service to improve performance**

  The dynamic cache service improves performance by caching the output of servlets, commands, and JavaServer Pages (JSP) files. Dynamic caching features include cache replication among clusters, cache disk offload, Edge-side include caching, and external caching, which is the ability to control caches outside of the application server, such as that of your Web server.

- Ensure your applications perform well.

  Details are available in the following topics:

  – “Application design consideration” on page 19 (architectural suggestions)

  – Designing applications.

    See the *Developing and deploying applications* PDF for more information.(coding best practices)

# Chapter 5. Obtaining advice from the advisors

Advisors provide a variety of recommendations that help improve the performance of your application server.

The advisors provide helpful performance as well as diagnostic advice about the state of the application server.

Tuning WebSphere Application Server is a critical part of getting the best performance from your Web site. However, tuning WebSphere Application Server involves analyzing performance data and determining the optimal server configuration. This determination requires considerable knowledge about the various components in the application server and their performance characteristics. The performance advisors encapsulate this knowledge, analyze the performance data, and provide configuration recommendations to improve the application server performance. Therefore, the performance advisors provide a starting point to the application server tuning process and help you without requiring that you become an expert.

The Runtime Performance Advisor is extended to also provide diagnostic advice and is now called the Performance and Diagnostic Advisor. Diagnostic advice provides useful information regarding the state of the application server. Diagnostic advice is especially useful when an application is not functioning as expected, or simply as a means of monitoring the health of application server.

- Decide which performance advisor is right for the purpose, Performance and Diagnostic Advisor or Tivoli Performance Viewer advisor.
- Use the chosen advisor to periodically check for inefficient settings, and to view recommendations.
- Analyze Performance Monitoring Infrastructure data with performance advisors.

## Why you want to use the performance advisors

The advisors analyze the Performance Monitoring Infrastructure (PMI) data of WebSphere Application Server using general performance principles, best practices, and WebSphere Application Server-specific rules for tuning. The advisors that are based on this information provide advice on how to set some of your configuration parameters to better tune WebSphere Application Server.

The advisors provide a variety of advice on the following application server resources:

- Object Request Broker service thread pools
- Web container thread pools
- Connection pool size
- Persisted session size and time
- Data source statement cache size
- Session cache size
- Dynamic cache size
- Java virtual machine heap size
- DB2 Performance Configuration wizard

For example, consider the data source statement cache. It optimizes the processing of *prepared statements* and *callable statements* by caching those statements that are not used in an active connection. (Both statements are SQL statements that essentially run repeatable tasks without the costs of repeated compilation.) If the cache is full, an old entry in the cache is discarded to make room for the new one. The best performance is generally obtained when the cache is large enough to hold all of the statements that are used in the application. The PMI counter, `prepared statement cache discards`, indicates the number of statements that are discarded from the cache. The performance advisors check this counter and provide recommendations to minimize the cache discards.

Using another example with pools in the application server, the idea behind pooling is to use an existing thread or connection from the pool instead of creating a new instance for each request. Because each thread or connection in the pool consumes memory and increases the context-switching cost, the pool size is an important configuration parameter. A pool that is too large can hurt performance as much as a pool that is too small. The performance advisors use PMI information about current pool usage, minimum or maximum pool size, and the application server CPU utilization to recommend efficient values for the pool sizes.

The advisors can also issue diagnostic advice to help in problem determination and health monitoring. For example, if your application requires more memory than is available, the diagnostic adviser tells you to increase the size or the heap for application server.

## Performance advisor types and purposes

Two performance advisors are available: the Performance and Diagnostic Advisor and the performance advisor in Tivoli Performance Viewer. The Performance and Diagnostic Advisor runs in the Java virtual machine (JVM) process of application server; therefore, it does not provide expensive advice. In a stand-alone application server environment, the performance advisor in Tivoli Performance Viewer runs within the application server JVM. In a Network Deployment environment, the performance advisor in Tivoli Performance Viewer runs within the JVM of the node agent and can provide advice on resources that are more expensive to monitor and analyze. The Tivoli Performance Viewer advisor requires that you enable performance modules, counters, or both.

The following chart shows the differences between the Performance and Diagnostic Advisor and the Tivoli Performance Viewer advisor:

| | **Performance and Diagnostic Advisor** | **Tivoli Performance Viewer advisor** |
|---|---|---|
| Start location | Application server | Tivoli Performance Viewer client |
| Invocation of tool | Administrative console | Tivoli Performance Viewer |
| Output | • The SystemOut.log file<br>• The administrative console<br>• JMX notifications | Tivoli Performance Viewer in the administrative console |
| Frequency of operation | Configurable | When you select refresh in the Tivoli Performance Viewer administrative console |
| Types of advice | Performance advice:<br>• Object Request Broker (ORB) service thread pools<br>• Web container thread pools<br>• Connection pool size<br>• Persisted session size and time<br>• Prepared statement cache size<br>• Session cache size<br>• Memory leak detection<br>Diagnostic advice:<br>• Connection factory diagnostics<br>• Data source diagnostic | Performance advice:<br>• ORB service thread pools<br>• Web container thread pools<br>• Connection pool size<br>• Persisted session size and time<br>• Prepared statement cache size<br>• Session cache size<br>• Dynamic cache size<br>• Java virtual machine (JVM) heap size<br>• DB2 Performance Configuration wizard |

## Performance and Diagnostic Advisor

Use this topic to understand the functions of the Performance and Diagnostic Advisor.

The Performance and Diagnostic Advisor provides advice to help tune systems for optimal performance and is configured using the WebSphere Application Server administrative console or the wsadmin tool. Running in the Java virtual machine (JVM) of the application server, the Performance and Diagnostic Advisor periodically checks for inefficient settings and issues recommendations as standard product warning messages. These recommendations are displayed both as warnings in the administrative console under Runtime Messages in the WebSphere Application Server Status panel and as text in the application server `SystemOut.log` file. Enabling the Performance and Diagnostic Advisor has minimal system performance impact.

The Performance and Diagnostic Advisor provides performance advice and diagnostic advice to help tune systems for optimal performance, and also to help understand the health of the system. It is configured using the WebSphere Application Server administrative console or the wsadmin tool. Running in the Java virtual machine (JVM) of the application server, the Performance and Diagnostic Advisor periodically checks for inefficient settings and issues recommendations as standard product warning messages. These recommendations are displayed as warnings in the administrative console under Runtime Messages in the WebSphere Application Server Status panel, as text in the application server `SystemOut.log` file, and as Java Management Extensions (JMX) notifications. Enabling the Performance and Diagnostic Advisor has minimal system performance impact.

From WebSphere Application Server, Version 6.0.2, you can use the Performance and Diagnostic Advisor to enable the lightweight memory leak detection, which is designed to provide early detection of memory problems in test and production environments.

The advice that the Performance and Diagnostic Advisor gives is all on the server level. The only difference when running in a Network Deployment environment is that you might receive contradictory advice on resources that are declared at the node or cell level and used at the server level.

For example, two sets of advice are given if a data source is declared at the node level to have a connection pool size of {10,50} and is used by two servers (server1 and server2). If server1 uses only two connections and server2 uses all fifty connections during peak load, the optimal connection pool size is different for the two servers. Therefore, the Performance and Diagnostic Advisor gives two sets of advice (one for server1 and another for server2). The data source is declared at the node level and you must make your decisions appropriately by setting one size that works for both, or by declaring two different data sources for each server with the appropriate level.

Read "Using the Performance and Diagnostic Advisor" on page 28 for startup and configuration steps.

## Diagnostic alerts

In WebSphere Application Server Version 6.1 the Performance and Diagnostic Advisors are extended to provide more diagnostic alerts to help common troubleshoot problems.

Several alerts are made available to monitor connection factory and data sources behavior. See the *Administering applications and their environment* PDF for more information. Some of these alerts are straightforward and easy to comprehend. Others are much more involved and are intended for use by IBM support only.

### ConnectionErrorOccured diagnostic alert

When a resource adapter or data source encounters a problem with connections such that the connection might no longer be usable, it informs the connection manager that a connection error occurred. This causes the destruction of the individual connection or a pool purge, which is the destruction of all connections in the pool, depending on the pool purge policy configuration setting. An alert is sent, indicating a potential problem with the back-end if an abnormally high number of unusable connections are detected.

### Connection low-percent efficiency diagnostic alert

If the percentage of time that a connection is used versus held for any individual connections drops below a threshold, an alert is sent with a call stack.

**Pool low-percent efficiency diagnostic alert**

If the average time that a connection is held versus used for the all connections in the pool drops below a threshold, an alert is sent.

**Surge mode entered or exited diagnostic alert**

When surge mode is configured, an alert is sent whenever surge mode engages or disengages. See the surge mode documentation in the *Administering applications and their environment* PDF for more information.

**Stuck connection block mode entered or exited diagnostic alert**

When stuck connection detection is configured, an alert is sent whenever stuck connection blocking starts or stops. See the stuck connection documentation in the *Administering applications and their environment* PDF .

**Local transaction containment (LTC) nesting threshold exceeded diagnostic alert**

For LTC definition, see the Local transaction containment (LTC) and Transaction type and connection behavior topics in the *Administering applications and their environment* PDF, and Default behavior of managed connections in WebSphere Application Server topic.

If a high number of LTCs are started on a thread before completing, an alert is raised. This alert is useful in debugging some situations where the connection pool is unexpectedly running out of connections due to multiple nested LTCs holding onto multiple shareable connections.

**Thread maximum connections exceeded diagnostic alert**

When one or more LTCs on a thread ties too many managed connections, or poolable connections for data sources an alert is issued.

**Serial reuse violation diagnostic alert**

For information on what serial reuse is, see the Transaction type and connection behavior topic in the *Administering applications and their environment* PDF. Some legitimate scenarios exist, where a serial reuse violation is appropriate, but in most cases this violation is not intended and might lead to data integrity problems.

If this alert is enabled, any time a serial reuse violation occurs within an LTC, an alert is sent.

### Tivoli Performance Viewer advisor

The performance advisor in Tivoli Performance Viewer (TPV) provides advice to help tune systems for optimal performance and provide recommendations on inefficient settings by using collected Performance Monitoring Infrastructure (PMI) data. Obtain the advice by selecting the performance advisor in TPV.

## Using the Performance and Diagnostic Advisor

The advisors analyze the Performance Monitoring Infrastructure (PMI) data of WebSphere Application Server using general performance principles, best practices, and WebSphere Application Server-specific rules for tuning.

1. Ensure that PMI is enabled, which is default. If PMI is disabled, consult theEnabling PMI using the administrative console topic. To obtain advice, you must first enable PMI through the administrative console and restart the server. The Performance and Diagnostic Advisor enables the appropriate monitoring counter levels for all enabled advice when PMI is enabled. If specific counters exist that are not wanted, or when disabling the Performance and Diagnostic Advisor, you might want to disable PMI or the counters that the Performance and Diagnostic Advisor enabled.
2. Click **Servers** > **Application servers** in the administrative console navigation tree.
3. Click *server_name* > **Performance and Diagnostic Advisor Configuration**.
4. Under the **Configuration** tab, specify the number of processors on the server. This setting is critical to ensure accurate advice for the specific configuration of the system.
5. Select the **Calculation Interval**. PMI data is taken over time and averaged to provide advice. The calculation interval specifies the length of time over which data is taken for this advice. Therefore, details within the advice messages display as averages over this interval.
6. Select the **Maximum Warning Sequence**. The maximum warning sequence refers to the number of consecutive warnings that are issued before the threshold is updated. For example, if the maximum warning sequence is set to `3`, then the advisor sends only three warnings, to indicate that the prepared statement cache is overflowing. After three warnings, a new alert is issued only if the rate of discards exceeds the new threshold setting.
7. Specify **Minimum CPU for Working System**. The minimum central processing unit (CPU) for a working system refers to the CPU level that indicates a application server is under production load. Or, if you want to tune your application server for peak production loads that range from 50-90% CPU utilization, set this value to `50`. If the CPU is below this value, some diagnostic and performance advice are still issued. For example, regardless of the CPU level if you are discarding prepared statements at a high rate, you are notified.
8. Specify **CPU Saturated**. The CPU saturated level indicates at what level the CPU is considered fully utilized. The level determines when concurrency rules no longer increase thread pools or other resources, even if they are fully utilized.
9. Click **Apply**.
10. Click **Save**.
11. Click the **Runtime** tab.
12. Click **Restart**. Select **Restart** on the Runtime tab to reinitialize the Performance and Diagnostic Advisor using the last configuration information that is saved to disk.

    This action also resets the state of the Performance and Diagnostic Advisor. For example, the current warning count is reset to zero (0) for each message.
13. Simulate a production level load. If you use the Performance and Diagnostic Advisor in a test environment, do any other tuning for performance, or simulate a realistic production load for your application. The application must run this load without errors. This simulation includes numbers of concurrent users typical of peak periods, and drives system resources, for example, CPU and memory, to the levels that are expected in production. The Performance and Diagnostic Advisor provides advice when CPU utilization exceeds a sufficiently high level only. For a list of IBM business partners that provide tools to drive this type of load, see the topic, Performance: Resources for learning in the subsection of Monitoring performance with third-party tools.
14. Select the check box to enable the Performance and Diagnostic Advisor.

    **Tip:** To achieve the best results for performance tuning, enable the Performance and Diagnostic Advisor when a stable production-level load is applied.
15. Click **OK**.
16. Select **Runtime Warnings** in the administrative console under the Runtime Messages in the Status panel or look in the SystemOut.log file, which is located in the following directory:

    ```
    profile_root/logs/server_name
    ```

    Some messages are not issued immediately.

17. Update the product configuration for improved performance, based on advice. Although the performance advisors attempt to distinguish between loaded and idle conditions, misleading advice might be issued if the advisor is enabled while the system is ramping up or down. This result is especially likely when running short tests. Although the advice helps in most configurations, there might be situations where the advice hinders performance. Because of these conditions, advice is not guaranteed. Therefore, test the environment with the updated configuration to ensure that it functions and performs better than the previous configuration.

    Over time, the advisor might issue differing advice. The differing advice is due to load fluctuations and the runtime state. When differing advice is received, you need to look at all advice and the time period over which it is issued. Advice is taken during the time that most closely represents the peak production load.

    Performance tuning is an iterative process. After applying advice, simulate a production load, update the configuration that is based on the advice, and retest for improved performance. This procedure is continued until optimal performance is achieved.

You can enable and disable advice in the Advice Configuration panel. Some advice applies only to certain configurations, and can be enabled only for those configurations. For example, unbounded Object Request Broker (ORB) service thread pool advice is only relevant when the ORB service thread pool is unbounded, and can only be enabled when the ORB thread pool is unbounded. For more information on Advice configuration, see the topic, “Advice configuration settings” on page 31.

## Performance and Diagnostic Advisor configuration settings

Use this page to specify settings for the Performance and Diagnostic Advisor.

To view this administrative page, click **Servers** > **Application Servers** > *server_name* > **Performance and Diagnostic Advisor Configuration** under the Performance section.

### Enable Performance and Diagnostic Advisor Framework

Specifies whether the Performance and Diagnostic Advisor runs on the server startup.

The Performance and Diagnostic Advisor requires that the Performance Monitoring Infrastructure (PMI) be enabled. It does not require that individual counters be enabled. When a counter that is needed by the Performance and Diagnostic Advisor or is not enabled, the Performance and Diagnostic Advisor enables it automatically. When disabling the Performance and Diagnostic Advisor, you might want to disable Performance Monitoring Infrastructure (PMI) or the counters that Performance and Diagnostic Advisor enabled. The following counters might be enabled by the Performance and Diagnostic Advisor:

- ThreadPools (module)
  - Web Container (module)
    - Pool Size
    - Active Threads
  - Object Request Broker (module)
    - Pool Size
    - Active Threads
- JDBC Connection Pools (module)
  - Pool Size
  - Percent used
  - Prepared Statement Discards
- Servlet Session Manager (module)
  - External Read Size
  - External Write Size
  - External Read Time
  - External Write Time
  - No Room For New Session
- System Data (module)
  - CPU Utilization

– Free Memory

### Enable automatic heap dump collection

Specifies whether the Performance and Diagnostic Advisor automatically generates heap dumps for post analysis when suspicious memory activity is detected.

### Calculation Interval

Specifies the length of time over which data is taken for this advice.

PMI data is taken over an interval of time and averaged to provide advice. The calculation interval specifies the length of time over which data is taken for this advice. Details within the advice messages display as averages over this interval. The default value is automatically set to four minutes.

### Maximum warning sequence

The maximum warning sequence refers to the number of consecutive warnings that are issued before the threshold is relaxed.

For example, if the maximum warning sequence is set to 3, the advisor only sends three warnings to indicate that the prepared statement cache is overflowing. After three warnings, a new alert is only issued if the rate of discards exceeds the new threshold setting. The default value is automatically set to one.

### Number of processors

Specifies the number of processors on the server.

This setting is helpful to ensure accurate advice for the specific configuration of the system. Depending your configuration and system, there may be only one processor utilized. The default value is automatically set to two.

### Minimum CPU For Working System

The minimum CPU for working system refers to the point at which concurrency rules do not attempt to free resources in thread pools.

There is a set of concurrency alerts to warn you if all threads in a pool are busy. This can affect performance, and it may be necessary for you to increase them. The CPU bounds are a mechanism to help determine when an application server is active and tunable.

The Minimum CPU for working system sets a lower limit as to when you should consider adjusting thread pools. For example, say you set this value to 50%. If the CPU is less than 50%, concurrency rules *do not* try to free up resources by decreasing pools to get rid of unused threads. That is, if the pool size is 50-100 and only 20 threads are consistently used then concurrency rules would like to decrease the minimum pool size to 20.

### CPU Saturated

The CPU Saturated setting determines when the CPU is deemed to be saturated.

There is a set of concurrency alerts to warn you if all threads in a pool are busy. This can affect performance, and it may be necessary for you to increase them. The CPU bounds are a mechanism to help determine when an application server is active and tunable.

The CPU saturated setting determines when the CPU has reached its saturation point. For example, if this is set to 95%, when the CPU is greater than 95% the concurrency rules *do not* try to improve things, that is, increase the size of a thread pool.

## Advice configuration settings

Use this page to select the advice you wish to enable or disable.

To view this administrative page, click **Servers** > **Application Servers** > *server_name* . Under the Performance section, click **Performance and Diagnostic Advisor Configuration** > **Performance and Diagnostic Advice Configuration**.

### Advice name

Specifies the advice that you can enable or disable.

### Advice applied to component

Specifies the WebSphere Application Server component to which the advice applies.

### Advice type

Categorizes the primary indent of a piece of Advice.

Use Advice type for grouping, and then enabling or disabling sets of advice that is based upon your purpose. Advice has the following types:

- Performance: Performance advice provides tuning recommendations, or identifies problems with your configuration from a performance perspective.
- Diagnostic: Diagnostic advice provide automated logic and analysis relating to problem identification and analysis. These types advice are usually issued when unexpected circumstances are encountered by the application server.

### Performance impact

Generalizes the performance overhead that an alert might incur.

The performance impact of a particular piece of advice is highly dependant upon the scenario being run and upon the conditions meet. The performance categorization of alerts is based upon worst case scenario measurements. The performance categorizations are:

- **Low:** Advice has minimal performance overhead. Advice might be run in test and production environments. Cumulative performance overhead is within run to run variance when all advice of this type is enabled.
- **Medium:** Advice has measurable but low performance overhead. Advice might be run within test environments, and might be run within production environments if deemed necessary. Cumulative performance overhead is less than 4% when all advice of this type is enabled.
- **High:** Advice impact is high or unknown. Advice might be run during problem determination tests and functional tests. It is not run in production simulation or production environments unless deemed necessary. Cumulative performance overhead might be significant when all advice of this type is enabled.

### Advice status

Specifies whether the advice is stopped, started, or unavailable.

The advice status has one of three values: **Started**, **Stopped** or **Unavailable**.

- **Started**: The advice is enabled.
- **Stopped**: The advice is not enabled.
- **Unavailable**: The advice does not apply to the current configuration, for example, persisted session size advice in a configuration without persistent sessions.

## Viewing the Performance and Diagnostic Advisor recommendations

Runtime Performance Advisor uses Performance Monitoring Infrastructure (PMI) data to provide recommendations for performance tuning.

The Performance and Diagnostic Advisor uses Performance Monitoring Infrastructure (PMI) data to provide recommendations for performance tuning. Running in the Java virtual machine (JVM) of the application server, this advisor periodically checks for inefficient settings, and issues recommendations as standard product warning messages.

The Performance and Diagnostic Advisor recommendations are displayed in two locations:
1. The WebSphere Application Server `SystemOut.log` log file.
2. The Runtime Messages panel in the administrative console. To view this administrative page, click **Troubleshooting** > **Runtime Messages** > **Runtime Warning**.

The following log file is a sample output of advice on the `SystemOut.log` file:

```
[4/2/04 15:50:26:406 EST] 6a83e321 TraceResponse W CWTUN0202W:
Increasing the Web Container thread pool Maximum Size to 48
might improve performance.

Additional explanatory data follows.

Average number of threads: 48.

Configured maximum pool size: 2.

This alert has been issued 1 time(s) in a row.
The threshold will be updated to reduce the
overhead of the analysis.
```

## Starting the lightweight memory leak detection

Use this task to start the lightweight memory leak detection using the Performance and Diagnostic Advisor.

If you have a memory leak and want to confirm the leak, or you want to automatically generate heap dumps on Java virtual machines (JVM) in WebSphere Application Server, consider changing your minimum and maximum heap sizes to be equal. This change provides the memory leak detection more time for reliable diagnosis.

To start the lightweight memory leak detection using the Performance and Diagnostic Advisor, perform the following steps in the administrative console:
1. Click **Servers > Application servers** in the administrative console navigation tree.
2. Click *server_name* **> Performance and Diagnostic Advisor Configuration**.
3. Click the **Runtime** tab.
4. Enable the Performance and Diagnostic Advisor Framework.
5. Click **OK**.
6. From the Runtime or Configuration tab of Performance and Diagnostic Advisor Framework, click **Performance and Diagnostic Advice configuration**.
7. Start the memory leak detection advice and stop any other unwanted advice.

The memory leak detection advice is started.

**Important:** To achieve the best results for performance tuning, start the Performance and Diagnostic Advisor when a stable production level load is running.

You can monitor any notifications of memory leaks by checking the `SystemOut.log` file or Runtime Messages. For more information, see the "Viewing the Performance and Diagnostic Advisor recommendations" on page 32 topic.

## Lightweight memory leak detection

This topic describes memory leaks in Java applications and introduces lightweight memory leak detection, a new function available in WebSphere Application Server Version 6.0.2 and above.

### Memory leaks in Java applications

Although a Java application has a built-in garbage collection mechanism, which frees the programmer from any explicit object deallocation responsibilities, memory leaks are still common in Java applications. Memory leaks occur in Java applications when unintentional references are made to unused objects. This occurrence prevents Java garbage collection from freeing memory.

The term *memory leak* is overused; a memory leak refers to a memory misuse or mismanagement. Old unused data structures might have outstanding references but are never garbage collected. A data structure might have unbounded growth or there might not be enough memory that is allocated to efficiently run a set of applications.

### Lightweight memory leak detection in WebSphere Application Server

Most existing memory leak technologies are based upon the idea that you know that you have a memory leak and want to find it. Because of these analysis requirements, these technologies have significant performance burdens and are not designed for use as a detection mechanism in production. This limitation means that memory leaks are generally not detected until the problem is critical; the application passes all system tests and is put in production, but it crashes and nobody knows why.

WebSphere Application Server has implemented a lightweight memory leak detection mechanism that runs within the WebSphere Performance and Diagnostic Advisor framework. This mechanism is designed to provide early detection of memory problems in test and production environments. This framework is not designed to provide analysis of the source of the problem, but rather to provide notification and help generating the information that is required to use analysis tools. The mechanism only detects memory leaks in the Java heap and does not detect native leaks.

The lightweight memory leak detection in WebSphere Application Server does not require any additional agents. The detection relies on algorithms that are based on information that is available from the Performance Monitoring Infrastructure service and has minimal performance overhead.

## Enabling automated heap dump generation

Use this task to enable automated heap dump generation on AIX, Linux, and Windows operating systems.

Although heap dumps are only generated in response to a detected memory leak, you must understand that generating heap dumps can have a severe performance impact on WebSphere Application Server for several minutes.

To help you analyze memory leak problems when memory leak detection occurs, some automated heap dump generation support is available.

To enable automated heap dump generation support, perform the following steps in the administrative console:

1. Click **Servers > Application servers** in the administrative console navigation tree.
2. Click *server_name* >**Performance and Diagnostic Advisor Configuration**.
3. Click the **Runtime** tab.
4. Select the **Enable automatic heap dump collection** check box.
5. Click **OK**.

The automated heap dump generation support is enabled.

**Important:** To preserve disk space, the Performance and Diagnostic Advisor does not take heap dumps if more than 10 heap dumps already exist in the WebSphere Application Server home directory. Depending on the size of the heap and the workload on the application server, taking a heap dump might be quite expensive and might temporarily affect system performance.

You can monitor any notifications of memory leaks by checking the `SystemOut.log` file or Runtime Messages. For more information, see the "Viewing the Performance and Diagnostic Advisor recommendations" on page 32 topic. If a memory leak is detected and you want to find the heap dump, refer to the Locating and analyzing heap dumps topic.

## Automated heap dump generation support

This topic explains the automated heap dump generation support, which is available only on IBM Software Development Kit and analyzes memory leak problems on AIX, Linux, and Windows operating systems.

Most memory leak analysis tools perform some forms of difference evaluation on two heap dumps. Upon detection of a suspicious memory situation, two heap dumps are automatically generated at appropriate times. The general idea is to take an initial heap dump as soon as problem detection occurs. Monitor the memory usage and take another heap dump when you determine that enough memory is leaked, so that you can compare the heap dumps to find the source of the leak.

## Generating heap dumps manually

Use this task to manually generate heap dumps on AIX, Linux, and Windows operating systems.

Although heap dumps are generated only in response to a detected memory leak, you must understand that generating heap dumps can have a severe performance impact on WebSphere Application Server for several minutes. When generating multiple heap dumps manually for memory leak analysis, make sure that significant objects are leaked in between the two heap dumps. This approach enables problem determination tools to identify the source of the memory leak.

You might want to manually generate heap dumps for the analysis of memory leaks. On a Java virtual machines (JVM) in WebSphere Application Server, you cannot enable automated heap dump generation. You might want to designate certain times to take heap dumps because of the overhead involved. On JVM in WebSphere Application Server, you can manually produce heap dumps by using the generateHeapDump operation on WebSphere Application Server managed beans (MBeans) that are special Java beans.

The WebSphere Application Server wsadmin tool provides the ability to run scripts. You can use the wsadmin tool to manage a WebSphere Application Server installation, as well as configuration, application deployment, and server runtime operations. WebSphere Application Server supports the Jacl and Jython scripting languages only. To learn more about the wsadmin tool, see the *Administering applications and their environment* PDF for more information.

1. Start the wsadmin scripting client. You have several options to run scripting commands, ranging from running them interactively to running them in a profile.
2. Invoke the generateHeapDump operation on a JVM MBean, for example,
   - Finding JVM objectName:
     ```
     <wsadmin> set objectName [$AdminControl queryNames
     WebSphere:type=JVM,process=<servername>,node=<nodename>,*]
     ```
   - Invoking the generateHeapDump operation on JVM MBean:
     ```
     <wsadmin> $AdminControl invoke $objectName generateHeapDump
     ```
     where,

| | |
|---|---|
| `$` | is a Jacl operator for substituting a variable name with its value |
| `invoke` | is the command |

| | |
|---|---|
| `generateHeapDump` | is the operation you are invoking |
| *<servername>* | is the name of the server on which you want to generate a heap dump |
| *<nodename>* | is the node to which *<servername>* belongs |

After running the **wsadmin** command, the file name of the heap dump is returned. For more information on finding heap dumps, refer to the Locating and analyzing heap dumps topic. When you have a couple of heap dumps, use a number of memory leak problem determination tools to analyze your problem. Memory Dump Diagnostic for Java™ is an offline tool for diagnosing root causes behind memory leaks in the Java heap. See Diagnosing out-of-memory errors and Java heap memory leaks for more information.

### Locating and analyzing heap dumps

Use this task to locate and analyze heap dumps.

Do not analyze heap dumps on the WebSphere Application Server machine because analysis is very expensive. For analysis, transfer heap dumps to a dedicated problem determination machine.

When a memory leak is detected and heap dumps are generated, you must analyze heap dumps on a problem determination machine and not on the application server because the analysis is very central processing unit (CPU) and disk I/O intensive.

Perform the following procedure to locate the heap dump files.

1. On the physical application server where a memory leak is detected, go to the WebSphere Application Server home directory. For example, on the Windows operating system, the directory is:

   `profile_root\myProfile`
2. IBM heap dump files are usually named in the following way:

   `heapdump.<date>..<timestamp><pid>.phd`
3. Gather all the .phd files and transfer them to your problem determination machine for analysis.
4. Many tools are available to analyze heap dumps that include Rational Application Developer 6.0. WebSphere Application Server serviceability released a technology preview called Memory Dump Diagnostic For Java. You can download this preview from the product download Web site.

When you have a couple of heap dumps, use a number of memory leak problem determination tools to analyze your problem.

## Using the performance advisor in Tivoli Performance Viewer

The performance advisor in Tivoli Performance Viewer (TPV) provides advice to help tune systems for optimal performance and provides recommendations on inefficient settings by using the collected Performance Monitoring Infrastructure (PMI) data.

Obtain advice by clicking **Performance advisor** in TPV. The performance advisor in TPV provides more extensive advice than the "Performance and Diagnostic Advisor" on page 26. For example, TPV provides advice on setting the dynamic cache size, setting the Java virtual machine (JVM) heap size and using the DB2 Performance Configuration wizard.

1. Enable PMI in the application server as described in the Enabling PMI using the administrative console article. To monitor performance data through the PMI interfaces, you must first enable PMI through the administrative console before restarting the server. If running in a Network Deployment environment, you must enable PMI on both the server and on the node agent before restarting the server and the node agent.
2. Enable data collection and set the PMI monitoring level to Extended. The monitoring levels that determine which data counters are enabled can be set dynamically, without restarting the server.

These monitoring levels and the data selected determine the type of advice you obtain. The performance advisor in TPV uses the extended monitoring level; however, the performance advisor in TPV can use a few of the more expensive counters (to provide additional advice) and provide advice on which counters can be enabled.

For example, the advice pertaining to session size needs the PMI statistic set to All. Or, you can use the PMI Custom Monitoring Level to enable the Servlet Session Manager SessionObjectSize counter. The monitoring of the SessionSize PMI counter is expensive, and is not in the Extended PMI statistic set. Complete this action in one of the following ways:

a. Performance Monitoring Infrastructure settings.

b. Enabling Performance Monitoring Infrastructure using the wsadmin tool.

3. In the administrative console, click **Monitoring and Tuning** > **Performance Viewer** > **Current activity**.
4. Simulate a production level load. Simulate a realistic production load for your application, if you use the performance advisor in a test environment, or do any other performance tuning. The application must run this load without errors. This simulation includes numbers of concurrent users typical of peak periods, and drives system resources, for example, CPU and memory to the levels that are expected in production. The performance advisor only provides advice when CPU utilization exceeds a sufficiently high level. For a list of IBM business partners providing tools to drive this type of load, see the article, Performance: Resources for learning in the subsection of Monitoring performance with third party tools.
5. Log performance data with TPV.
6. Clicking **Refresh** on top of the table of advice causes the advisor to recalculate the advice based on the current data in the buffer.
7. Tuning advice displays when the Advisor icon is chosen in the TPV Performance Advisor. Double-click an individual message for details. Because PMI data is taken over an interval of time and averaged to provide advice, details within the advice message display as averages.

   **Note:** If the Refresh Rate is adjusted, the Buffer Size must also be adjusted to enable sufficient data to be collected for performing average calculations. Currently 5 minutes of data is required. Hence, the following guidelines intend to help you use the Tivoli Performance Advisor:

   a. You cannot have a Refresh Rate of more than 300 seconds.

   b. RefreshRate * BufferSize > 300 seconds. Buffer Size * Refresh Rate is the amount of PMI data available in memory and it must be greater than 300 seconds.

   c. For the Tivoli Performance Advisor to work properly with TPV logs, the logs must be at least 300 seconds of duration.

   For more information about configuring user and logging settings of TPV, refer to the Configuring TPV settings article.
8. Update the product configuration for improved performance, based on advice. Because Tivoli Performance Viewer refreshes advice at a single instant in time, take the advice from the peak load time. Although the performance advisors attempt to distinguish between loaded and idle conditions, misleading advice might be issued if the advisor is enabled while the system is ramping up or down. This result is especially likely when running short tests. Although the advice helps in most configurations, there might be situations where the advice hinders performance. Because of these conditions, advice is not guaranteed. Therefore, test the environment with the updated configuration to ensure it functions and performs well.

   Over a period of time the advisor might issue differing advice. The differing advice is due to load fluctuations and run-time state. When differing advice is received, you need to look at all advice and the time period over which it was issued. You must take advice during the time that most closely represents the peak production load.

Performance tuning is an iterative process. After applying advice, simulate a production load, update the configuration that is based on the advice, and retest for improved performance. This procedure is continued until optimal performance is achieved.

# Chapter 6. Tuning the application serving environment

This topic describes the benefits of tuning for optimal performance, highlights the tunable parameters of the major WebSphere Application Server components, and provides insight about how these parameters affect performance.

WebSphere Application Server provides tunable settings for its major components to enable you to make adjustments to better match the runtime environment to the characteristics of your application. Many applications can run successfully without any changes to the default values for these tuning parameters. Other applications might need changes, for example, a larger heap size, to achieve optimal performance.

Performance tuning can yield significant gains in performance even if an application is not optimized for performance. However, correcting shortcomings of an application typically results in larger performance gains than are possible with just altering tuning parameters. Many factors contribute to a high performing application.

**Tuning parameter index for z/OS** Performance tuning for WebSphere Application Server for z/OS becomes a complex exercise because the nature of the runtime involves many different components of the operating system and middleware. Use the “Tuning parameter hot list” on page 1 to find information and parameters for tuning the z/OS operating system, subsystems, the WebSphere Application Server for z/OS run-time environment, and some Java 2 Platform, Enterprise Edition (J2EE) application tuning tips.

**Recommendation:** Before you read a description of WebSphere Application Server for z/OS tuning guidelines, it is important to note that, no matter how well the middleware is tuned, it cannot make up for poorly designed and coded applications. Focusing on the application code can help improve performance. Often, poorly written or designed application code changes will make the most dramatic improvements to overall performance.

The tuning guide focuses on server tuning. If you want to tune your applications, see the Performance: Resources for learning article for more information about application tuning.

For your convenience, procedures for tuning parameters in other products, such as DB2, Web servers and operating systems are included. Because these products might change, consider these descriptions as suggestions.

Each WebSphere Application Server process has several parameters that influence application performance. You can use the WebSphere Application Server administrative console to configure and tune applications, Web containers, Enterprise JavaBeans (EJB) containers, application servers and nodes in the administrative domain.

If you are a WebSphere Application Server administrator or Systems programmer on WebSphere Application Server for z/OS, refer to “Tuning index for WebSphere Application Server for z/OS” on page 40 for z/OS specific tuning tips.

Each parameter description: explains the parameter; provides reasons to adjust the parameter; discusses how to view or set the parameter; as well as indicates default and recommended values.

**Additional references:**

- WebSphere Application Server - Performance Web site

## Tuning index for WebSphere Application Server for z/OS

One of the goals of the WebSphere Application Server for z/OS programming model and runtime is to significantly simplify the work required for application developers to write and deploy applications. Sometimes we say that WebSphere Application Server for z/OS relieves the application programmer of many of the plumbing tasks involved in developing applications. For example, application code in WebSphere Application Server for z/OS does not concern itself directly with remote communication--it locates objects which may be local or remote and drives methods. Therefore, you won't see any direct use of socket calls or TCP/IP programming in a WebSphere Application Server for z/OS application.

This separation of what you want to do from where you do it is one aspect of removing the application programmers from plumbing tasks. Other considerations are not having to deal with data calls for some types of beans, potentially user authentication, and threading. There are generally no calls from the application code to touch sockets, RACF calls, or management of threading. Removing this from the application programmer doesn't mean this work won't get done. Rather, it means that there may be more work for the DBA, the network administrator, the security administrator, and the performance analyst.

There are three layers of tuning that need to be addressed:
- "Tuning the z/OS operating system" on page 66
- "Tuning the WebSphere Application Server for z/OS runtime" on page 73

We deal with these in separate sections under this article. For more information on tuning applications, refer to Using application clients.

## Tuning parameter hot list

The following hot list contains recommendations that have improved performance or scalability, or both, for many applications.

WebSphere Application Server provides several tunable parameters and options to match the application server environment to the requirements of your application.
- **Review the hardware and software requirements**

  Review the hardware and software requirements on the IBM WebSphere Application Server supported hardware, software, and APIs Web site to get started.
- **Check hardware configuration and settings**

  Check network connections to make sure that they are running at their highest speed. Sometimes transient errors can cause Ethernet adapters to shift down to a lower speed. Verify that the system has adequate memory and that the number and position of memory dual inline memory module (DIMM) are optimal. With some systems, there are some memory DIMM configurations that permit higher performance than other DIMM configurations. Verify that the hardware that is used is the hardware that is supposed to be used.
- **Review your application design**

  You can track many performance problems back to the application design. Review the design to determine if it causes performance problems.
- **Tune the operating systems**

  Operating system configuration plays a key role in performance. In many cases, adjustments to some TCP/IP parameters might be necessary for your application.
- **Set the minimum and maximum Java virtual machine (JVM) heap sizes**

  Many applications need a larger heap size for best performance.
- **Use a type 4 (or pure Java) JDBC driver**

  In general, the type 2 JDBC driver is recommended. However, the type 4 JDBC drivers perform faster in the area of multi-row fetch. Use the link above to view a list of database vendor-specific requirements, which can tell you if a type 4 JDBC driver is supported for your database.

  See the *Administering applications and their environment* PDF for more information.
- **Tune WebSphere Application Server JDBC data sources and associated connection pools**

The JDBC data source configuration might have a significant performance impact. For example, the connection pool size and prepared statement cache need to be sized based on the number of concurrent requests being processed and the design of the application.

See the *Administering applications and their environment* PDF for more information.

- **Ensure that the transaction log is assigned to a fast disk**

  Some applications generate a high rate of writes to the WebSphere Application Server transaction log. Locating the transaction log on a fast disk or disk array can improve response time

  See the *Administering applications and their environment* PDF for more information.
- **Tune related components, for example, database**

  In many cases, some other component, for example, a database, needs adjustments to achieve higher throughput for your entire configuration.

  For more information, see the *Administering applications and their environment* PDF for more information.
- **Tuning index for WebSphere Application Server for z/OS**

  One of the goals of the WebSphere Application Server for z/OS programming model and runtime is to significantly simplify the work required for application developers to write and deploy applications. Sometimes we say that WebSphere Application Server for z/OS relieves the application programmer of many of the plumbing tasks involved in developing applications. For example, application code in WebSphere Application Server for z/OS does not concern itself directly with remote communication--it locates objects which may be local or remote and drives methods. Therefore, you won't see any direct use of socket calls or TCP/IP programming in a WebSphere Application Server for z/OS application.

  This separation of what you want to do from where you do it is one aspect of removing the application programmers from plumbing tasks. Other considerations are not having to deal with data calls for some types of beans, potentially user authentication, and threading. There are generally no calls from the application code to touch sockets, RACF calls, or management of threading. Removing this from the application programmer doesn't mean this work won't get done. Rather, it means that there may be more work for the DBA, the network administrator, the security administrator, and the performance analyst.

  There are four layers of tuning that need to be addressed:
  - "Tuning the z/OS operating system" on page 66
  - Tuning for subsystems
  - Tuning the WebSphere Application Server for z/OS runtime
  - Tuning for J2EE applications

  We deal with the first three in separate sections under this article and briefly touch on the fourth. For more information on tuning applications, refer to Using application clients.
- **Tuning for subsystems**

  Steps involved in tuning the z/OS subsystems to optimize WebSphere performance include:
  - "DB2 tuning tips for z/OS" on page 3
  - "Security tuning tips" on page 126
  - "Tuning TCP/IP buffer sizes" on page 56
  - "GRS tuning tips for z/OS" on page 141
  - "Tuning Java virtual machines" on page 57
  - "CICS tuning tips for z/OS" on page 141
- **Tuning the WebSphere Application Server for z/OS runtime**

  Steps involved in tuning the WebSphere Application Server for z/OS runtime to optimize performance include reviewing the:
  - Review the WebSphere for z/OS configuration
  - Internal tracing tips for WebSphere Application Server for z/OS
  - Location of executable programs tips for z/OS
  - "Security tuning tips" on page 126
- **Review the WebSphere for z/OS configuration**

The first thing to do is review the WebSphere for z/OS configuration. One simple way to do this is to look in your application control and server regions in SDSF. When each server starts, the runtime prints out the current configuration data in the joblog.

- **Internal tracing tips for WebSphere for z/OS**

  WebSphere traces can be extremely helpful in detecting and diagnosing problems. By properly setting trace options, you can capture the information needed to detect problems without significant performance overhead.

  – Ensure that you are not collecting more diagnostic data than you need.

    You should check your WebSphere for z/OS tracing options to ensure that ras_trace_defaultTracingLevel=0 or 1, and that ras_trace_basic and ras_trace_detail are not set.

    **How to view or set:** Use the WebSphere administrative console:
    1. Click **Environment > Manage WebSphere Variables**.
    2. On the Configuration Tab check for any of these variables in the name field and observe the variable setting in the value field.
    3. To change or set a variable, specify the variable in the name field and specify the setting in the value field. You can also describe the setting in the description field on this tab.

  – If you use any level of tracing, including ras_trace_defaultTracingLevel=1, ensure that you set ras_trace_outputLocation to BUFFER.

    ras_trace_defaultTracingLevel=1 will write exceptions to the trace log as well as to the ERROR log.

    - It is best to trace to CTRACE.

      If you are tracing to sysprint with ras_trace_defaultTracingLevel=3, you may experience an almost 100% throughput degradation. If you are tracing to CTRACE, however, you may only experience a 15% degradation in throughput.

  – Set the ras_trace_BufferCount=4 and ras_trace_BufferSize=128.

    This will get 512KB of storage for the trace buffers (the minimum allowed) and reduce memory requirements.

  – Make sure you disable JRAS tracing.

    To do this, look for the following lines in the trace.dat file pointed to by the JVM properties file:

    ```
    com.ibm.ejs.*=all=disable

    com.ibm.ws390.orb=all=disable
    ```

    Ensure that both lines are set to **=disable** or delete the two lines altogether.

    **Note:** If ras_trace_outputLocation is set, you may be tracing and not know it.

- **Location of executable programs tips for z/OS**

  The next thing to review in the configuration is where your program code is located. IBM recommends that you install as much of the WebSphere for z/OS code in LPA as is reasonable, and the remainder in the linklist. This ensures that you have eliminated any unnecessary steplibs which can adversely affect performance. If you must use STEPLIBs, verify that any `STEPLIB DD`s in the controller and servant procs do not point to any unnecessary libraries. Refer to “UNIX System Services (USS) tuning tips for z/OS” on page 72 for USS shared file system tuning considerations.

  If you choose to not put most of the runtime in LPA, you may find that your processor storage gets a bigger workout as the load increases. At a minimum, WebSphere for z/OS will start three address spaces, so that any code that is not shared will load three copies rather than one. As the load increases, many more servants may start and will contribute additional load on processor storage.

  Review the `PATH` statement to ensure that only required programs are in the PATH and that the order of the PATH places frequently-referenced programs in the front.

- **Tuning for J2EE applications**

  Steps involved in tuning the J2EE applications performance include:

  – “Tuning WebSphere applications” on page 78

  – “Tuning TCP/IP buffer sizes” on page 56

  – Tuning for SOAP

– “Tuning MDB processing on z/OS” on page 6

## DB2 tuning tips for z/OS

Performance tuning for DB2 is usually critical to the overall performance of a WebSphere Application Server application. DB2 is often the preferred datastore for Enterprise JavaBeans (EJBs). Listed here are some basic guidelines for DB2 tuning as well as some guidelines for tuning DB2 for WebSphere Application Server. For more complete information on DB2 tuning, refer to the DB2 Universal Database for OS/390 and z/OS Administration Guide Document Number SC26-9931-03. The DB2 books can be accessed at the following Internet location:http://www.ibm.com/servers/eserver/zseries/zos/.

**Benefits of Structured Query Language in Java (SQLJ)**

If you use the DB2 Universal JDBC driver provider, you can implement SQLJ as the query language for both BMP and CMP beans. SQLJ incurs less transaction overhead than the default query language for JDBC transactions, which is dynamic SQL. SQLJ is static and uses pre-prepared plans. Thus SQLJ generally improves application performance. For DB2 for z/OS database administrators, SQLJ is often easy to adopt because the security model and the statement repeatability features are similar to those of static SQL. SQLJ does require extra steps that are features of newer versions of WebSphere Studio Application Developer and Rational Application Developer..

See the *Administering applications and their environment* PDF for more information.

**General DB2 tuning tips:**

This discussion relates only to DB2 for z/OS JDBC Driver which is referred to as the DB2 for z/OS Legacy JDBC Driver.

- First, ensure that your DB2 logs are large enough, are allocated on the fastest volumes you have, and make sure they have optimal CI sizes.
- Next, ensure that you have tuned your bufferpools so that the most often-read data is in memory as much as possible. Use ESTOR and hyperpools.
- You many want to consider pre-formatting tables that are going to be heavily used. This avoids formatting at runtime.

**DB2 for WebSphere tuning tips:**

- Ensuring DB2 Tracing Under the DB2 for z/OS Universal Driver is Turned Off.
  - If the db2.jcc.propertiesFile jvm property has been defined to specify a DB2 jcc properties file to the WebSphere Application Server for z/OS, ensure that the following trace statements in the file are commented out if they are specified:

    ```
    # jcc.override.traceFile=<file name>
    # jcc.override.traceFile=<file name>
    ```
  - If any of the DB2 Universal JDBC Driver datasources your applications are using are defined with a nonzero traceLevel custom property, use the WebSphere Application Server for z/OS Administrative console to set the traceLevel to zero.
- We recommend that you ensure indexes are defined on all your object primary keys. Failure to do so will result in costly tablespace scans.
- Ensure that, once your tables are sufficiently populated, you do a re-org to compact the tables. Running RUNSTATS will ensure that the DB2 catalog statistics about table and column sizes and accesses are most current so that the best access patterns are chosen by the optimizer.
- You will have to define more connections called threads in DB2. WebSphere Application Server uses a lot of threads. Sometimes this is the source of throughput bottlenecks since the server will wait at the create thread until one is available.
- Make sure you are current with JDBC maintenance. Many performance improvements have been made to JDBC. To determine the JDBC maintenance level, enter the following from the shell:

```
java  com.ibm.db2.jcc.DB2Jcc  -version
```

If this returns a class not found, either you are at a level of the driver that is older and doesn't support this command or you have not issued the command properly.

- We recommend that you enable dynamic statement caching in DB2. To do this, modify your ZPARMS to say `CACHEDYN(YES) MAXKEEPD(16K)`. Depending on the application, this can make a very significant improvement in DB2 performance. Specifically, it can help JDBC and LDAP query.
- Increase DB2 checkpoint interval settings to a large value. To do this, modify your ZPARMS to include `CHKFREQ=xxxxx`, where xxxxx is set at a high value when doing benchmarks. On production systems there are other valid reasons to keep checkpoint frequencies lower, however.

**Example:** This example identifies zparm values discussed in this article.

```
//DB2INSTE   JOB MSGCLASS=H,CLASS=A,NOTIFY=IBMUSER
/*JOBPARM SYSAFF=*
//*******************************************************************
//* JOB NAME = DSNTIJUZ
//*
//* DESCRIPTIVE NAME = INSTALLATION JOB STREAM
//*
//*    LICENSED MATERIALS - PROPERTY OF IBM
//*    5675-DB2
//*    (C) COPYRIGHT 1982, 2000 IBM CORP.  ALL RIGHTS RESERVED.
//*
//*    STATUS = VERSION 7
//*
//* FUNCTION = DSNZPARM AND DSNHDECP UPDATES
//*
//* PSEUDOCODE =
//*   DSNTIZA  STEP  ASSEMBLE DSN6.... MACROS, CREATE DSNZPARM
//*   DSNTIZL  STEP  LINK EDIT DSNZPARM
//*   DSNTLOG  STEP  UPDATE PASSWORDS
//*   DSNTIZP  STEP  ASSEMBLE DSNHDECP DATA-ONLY LOAD MODULE
//*   DSNTIZQ  STEP  LINK EDIT DSNHDECP LOAD MODULE
//*   DSNTIMQ  STEP  SMP/E PROCESSING FOR DSNHDECP
//*
//* NOTES = STEP DSNTIMQ MUST BE CUSTOMIZED FOR SMP.  SEE THE NOTES
//*         NOTES PRECEDING STEP DSNTIMQ BEFORE RUNNING THIS JOB.
//*
//*  LOGLOAD=16000000,
//*******************************************************************/
//*
//DSNTIZA EXEC PGM=ASMA90,PARM='OBJECT,NODECK'
//STEPLIB DD DSN=ASM.SASMMOD1,DISP=SHR
//SYSLIB   DD  DISP=SHR,
//         DSN=DB2710.SDSNMACS
//         DD  DISP=SHR,
//         DSN=SYS1.MACLIB
//SYSLIN   DD  DSN=&LOADSET(DSNTILMP),DISP=(NEW,PASS),
//             UNIT=SYSALLDA,
//             SPACE=(800,(50,50,2)),DCB=(BLKSIZE=800)
//SYSPRINT DD  SYSOUT=*
//SYSUDUMP DD  SYSOUT=*
//SYSUT1   DD  UNIT=SYSALLDA,SPACE=(800,(50,50),,,ROUND)
//SYSUT2   DD  UNIT=SYSALLDA,SPACE=(800,(50,50),,,ROUND)
//SYSUT3   DD  UNIT=SYSALLDA,SPACE=(800,(50,50),,,ROUND)
//SYSIN    DD  *
    DSN6ENV    MVS=XA
    DSN6SPRM   RESTART,                 X
               .
               .
               .
               AUTH=YES,
                                    X
               AUTHCACH=1024,                                       X
               BINDNV=BINDADD,                                      X
               BMPTOUT=4,                                           X
               CACHEDYN=YES,
                                   X
               .
               .
               .
               MAXKEEPD=16000,
                                 X
               .
               .
               .
      DSN6ARVP ALCUNIT=CYL,                                         X
               .
               .
               .
      DSN6LOGP DEALLCT=(0),                                         X
               .
               .
               .
      DSN6SYSP AUDITST=NO,                                          X
               BACKODUR=5,                                          X
              CHKFREQ=16000000,
                               X
               CONDBAT=400,                                         X
               CTHREAD=1200,                                        X
               DBPROTCL=PRIVATE,                                    X
               DLDFREQ=5,                                           X
               DSSTIME=5,                                           X
               EXTRAREQ=100,                                        X
               EXTRASRV=100,                                        X
               EXTSEC=NO,                                           X
               IDBACK=1800,
                                     X
               .
               .
               .
             //*
```

## WebSphere Application Server tuning tips for use with DB2

WebSphere Application Server uses JDBC prepared statement caching as a performance enhancing feature. If you are using this feature together with DB2 for z/OS, be aware of the potential impact on the number of DB2 JDBC cursor objects available.

### Prepared statement caching effects on DB2 for OS/390 JDBC cursor objects

When you obtain a *ResultSet* object by running a *PreparedStatement* object, a DB2 JDBC cursor object is bound to it until the corresponding DB2 prepared statement is closed. This happens when the DB2 *Connection* object is released from the WebSphere Application Server connection pool. From an application perspective, the result set, prepared statement, and connection are each closed in turn. However, the underlying DB2 *Connection* is pooled by the WebSphere Application Server, the underlying DB2 *PreparedStatement* is cached by the application server, and each underlying DB2 JDBC cursor object associated with each *ResultSet* created on this *PreparedStatement* object is not yet freed.

Each *PreparedStatement* object in the cache can have one or more result sets associated with it. If a result set is opened and not closed, even though you close the connection, that result set is still associated with the prepared statement in the cache. Each of the result sets has a unique JDBC cursor attached to it. This cursor is kept by the statement and is not released until the prepared statement is cleared from the WebSphere Application Server cache.

If there are more of the cached statements than there are cursors, eventually the execution of a *PreparedStatement* object results in the following exception:

```
java.sql.SQLException: DB2SQLJJDBCProfile Error: No more JDBC Cursors without hold
```

Some WebSphere Application Server tuning suggestions to help avoid this problem are:

1. Decrease the *statement cache size* setting on the DB2 for OS/390 data source definition. Setting this value to zero (0) eliminates statement caching, but causes a noticeable performance impact.
2. Decrease the *minConnections* connection pool setting on the DB2 for OS/390 data source definition.
3. Decrease the *Aged Timeout* connection pool setting on the DB2 for OS/390 data source definition. However, it is **NOT** recommended that you set this to zero (0), as this disables the Aged Timeout function.

## Tuning MDB processing on z/OS

To tune MDB processing, you consider and act on a variety of settings together. There is a wide range of values that you can select, and possibilities to consider, because of the variety of workloads possible to run in any given server.

When a message-driven bean is mapped (that is, listening to) a queue, or to a topic through a durable subscription, a JMS message first enters into the WebSphere server in the controller, so we say the server is “listening in the controller” for these messages. The “listening in the controller” term is used throughout this description of tuning MDB processing.

If you want to optimize the processing of messages by message-driven beans, use this task to consider and act on the associated settings.

Tuning MDB processing in the server is a part of the greater task of tuning the server’s entire workload. This needs consideration of a variety of settings and the interactions between those settings.

To tune MDB processing, the following settings must be considered together: WLM service class definitions, WebSphere workload profile selection, Message Listener Service Listener Port settings, JMS Connection Factory pooling settings, and WebSphere MQ Queue Manager settings.

It is difficult to give one recommendation about what values to select for each of these settings, given the variety of workloads possible to run in any given server. There are a wide variety of possibilities to take into consideration, including the following considerations:

- The number of message-driven beans.
- The administrative configuration choices, such as whether to map two message-driven beans to the same or different listener ports.
- Different goals regarding the importance of work for message-driven beans relative to other (HTTP, IIOP) types of work running in the server.

The suggested settings below provide a starting point, under the assumption that the server is configured with only a single application consisting of a single message-driven bean installed and running on this server.

The starting point settings are followed by more detailed discussions that explain the rationale behind the suggestions, and describe the listener port function in more detail in the “listening in the controller” case on z/OS. Together they can help you to understand the function and the settings enough to make your own setting selections for your own systems and servers.

1. 1. Set the Listener Port maximum sessions property to at least twice the maximum number of servant worker threads available to the entire server. The value of this property determines the high threshold value (high threshold = maximum sessions) and is used by the throttle to decide when to block or allow requests.

   a. To view this administrative console page, click **Servers-> Application Servers->** ***application_server*-> Message Listener Service-> Listener Ports->** ***listener_port***
   b. Set the maximum sessions property to the value you want the MDB throttle to use as its high threshold value.

      The suggested minimum value is computed by the formula:

      ```
      2 * (maximum number of servants) * (number of worker threads in one servant)
      ```

      Here "servants" means the same as "server instances" as the setting is described on the administrative console. To calculate the number of worker threads in a single servant, see the description of "Workload profile". See the *Developing and deploying applications* PDF for more information.

   For further considerations setting the Listener Port maximum sessions property, see "Concepts and considerations for MDB settings on z/OS" on page 8.
2. Set the WebSphere MQ Queue Connection Factory properties.
   a. To view this administrative console page, click **Resources-> WebSphere MQ JMS Provider-> WebSphere MQ Queue Connection Factories (under additional properties)**
   b. Select the Connection Factory specified for the Listener Port.
   c. Under the Additional Properties, activate the Connection Pool panel.
   d. Set the Max Connections property for the Connection Pool. Allow one connection for each message-driven bean. This property value could include message-driven beans mapped onto different listener ports, if those listener ports were each, in turn, mapped onto the same connection factory. For more detailed discussions about considerations that affect this setting, see "Concepts and considerations for MDB settings on z/OS" on page 8.
   e. Under Additional Properties of the Connection Factory, activate the Session Pool panel.
   f. Set the Max Connections property for the Session Pool. Allow one session for each worker thread in a single servant. This property should be set to at least the number of worker threads available to a single servant. For more detailed discussions about considerations that affect this setting, see "Concepts and considerations for MDB settings on z/OS" on page 8.
3. Set the WebSphere MQ-related properties. Make sure that the backing WebSphere MQ queue manager has been configured with enough resources to support the intended JMS workload coming from WebSphere Application Server (and other clients). In particular, you may want to consider your queue manager's CTHREAD, IDBACK, and IDFORE parameter settings. For more information on these WebSphere MQ settings, see the WebSphere MQ books:

   WebSphere MQ System Setup Guide (SC34-6052-02).

   WebSphere MQ Script Command (MQSC) Reference Book (SC34-6055-03)

   WebSphere MQ Problem Determination Guide (GC34-6054-01).

1. Suppose your server is configured with the maximum server instances value set to 3, (whatever the minimum number may be). Also suppose that your workload profile is LONGWAIT, which means each servant contains 40 worker threads.

   In this case, set your listener port maximum sessions value to at least 240 = 2 * 3 * 40.
2. Suppose that your application contains two individual message-driven beans, each of which has an onMessage() implementation that forwards the message to another JMS destination. Therefore, each message-driven bean needs its own JMS connection factory to perform this task. Suppose the Administrator has mapped each MDB's JMS connection factory resource reference onto the same administratively-defined connection factory used by the listener port that each of these message-driven beans is mapped onto.

   In this case, you need to set the connection factory's Connection Pool Max Connections value to 42. One connection for each of the two message-driven beans to be used by the listener port, and one connection potentially for each of the 40 onMessage() dispatches that night be running concurrently. (Remember that the connection pool is a per-servant pool).
3. Set the connection factory's Session Pool Max Connections to 40, the number of worker threads in a single servant, regardless of the number of servants.

For debugging tips, refer to “Best practices for debugging MDB throttle support” on page 16.

## Concepts and considerations for MDB settings on z/OS

To be able to tune the processing of endpoint message-driven beans (MDBs), you need to understand the concepts and considerations for the MDB settings that you configure.

When a message-driven bean is mapped (that is, listening to) a queue, or to a topic through a durable subscription, a JMS message first enters into the WebSphere server in the controller, so we say the server is “listening in the controller” for these messages. The “listening in the controller” term is used throughout this description of tuning MDB processing.

The following sub-topics provide a set of information that describe the concepts and considerations that you should be aware of to be able to configure MDB settings on z/OS:

- “Basic “listening in the controller” messaging flow” on page 9

  When a message-driven bean is mapped (that is, listening to) a queue, or to a topic through a durable subscription, a JMS message first enters into the WebSphere server in the controller, so we say the server is “listening in the controller” for these messages. When a message arrives, it flows through a sequence of events.
- “MDB throttle” on page 9

  On z/OS, the “MDB throttle” is used to control the amount of work that the server processes at a given time for a message-driven bean. The MDB throttle limits how far the message listener port will “read ahead” to try to ensure that the work request queue does not have a backlog of messages to be processed.
- “MDB throttle settings for message-driven beans on z/OS” on page 10

  You can tune a variety of settings for the “MDB throttle”, to control the amount of MDB work that the server processes at a given time.
- “Connection factory settings for message-driven beans on z/OS” on page 12

  You can tune a variety of connection factory settings to control the creation of connections and sessions for MDB work.
- “Message-driven beans, heterogeneous workloads, and workload management on z/OS” on page 14

  A message-driven bean can run on an application server that hosts a heterogeneous workload including other message-driven beans and non-MDB work items. To manage a heterogeneous workload on z/OS, you should use WLM classification and define unique service classes for different-priority work running in the same server.

***Basic “listening in the controller” messaging flow:***

When a message-driven bean is mapped (that is, listening to) a queue, or to a topic through a durable subscription, a JMS message first enters into the WebSphere server in the controller, so we say the server is “listening in the controller” for these messages. When a message arrives, it flows through a sequence of events.

When a message has arrived on a JMS destination (queue or topic) on which a message-driven bean running is listening, the following sequence of events takes place:

1. The WebSphere MQ JMS queue agent thread (running in the controller) browses the JMS destination. The queue agent identifies the application server on which a message-driven bean is running for that destination. The queue agent calls a Message Reference Handler (MRH) registered by the application server, to tell the server that a message has arrived on a destination on which one of its message-driven beans is listening. A message reference corresponding to this message is sent along with the call to the MRH.
2. If the throttle high threshold value (see below) has not been exceeded, a work request corresponding to the just-browsed message is queued onto the WLM queue in the controller.

3. The work request is dispatched to an individual servant. A ServerSession is obtained from the appropriate ServerSessionPool in that servant and, using the message reference as index, the message is destructively consumed from the JMS destination.
4. The MDB application onMessage(Message) method is dispatched with the consumed message.
5. When the onMessage(Message) method finishes, the servant notifies the controller that the work record has completed.

***MDB throttle:***

On z/OS, the “MDB throttle” is used to control the amount of work that the server processes at a given time for a message-driven bean. The MDB throttle limits how far the message listener port will “read ahead” to try to ensure that the work request queue does not have a backlog of messages to be processed.

The preprocessing, classification, building of and queuing of a work record to prepare for dispatching a specific message-driven bean is a relatively basic operation, especially when compared to the actual business logic of the message-driven bean and the container infrastructure on the application dispatch path. When the rate of messages arriving reaches occasional high volumes, or peaks, the controller can preprocess many more messages than the back-end servants can process by running the associated MDB application. The result of these peaks (in asynchronous work) is that the WLM work request queue becomes backed up waiting for worker threads (in the servants) that have a backlog of messages to be processed.

A backlog of messages to be processed can also occur as a result of situations where the scalable server is taken out of service for a period of time. This causes messages to build up on the JMS destination waiting for the server to restart. When the server does restart, there is a flood of new work introduced into the server.

The MDB throttle limits how far the message listener port will “read ahead” down the JMS queue (or topic), to try to ensure that the work request queue does not have a backlog of messages to be processed.

***MDB throttle settings for message-driven beans on z/OS:***

You can tune a variety of settings for the “MDB throttle”, to control the amount of MDB work that the server processes at a given time.

This topic describes the following concepts and considerations that affect your choice of MDB throttle settings:
- “MDB throttle – high and low thresholds” on page 10
- “MDB throttle – tuning considerations” on page 11
- “MDB throttle – alternative tuning considerations” on page 11
- “MDB throttle example” on page 11

**MDB throttle – high and low thresholds**

The MDB throttle support maintains a count of the current number of in-flight messages for each listener port.

When a message reference is sent to the MRH, as described in “Basic “listening in the controller” messaging flow” on page 9, the in-flight count is incremented by 1. Next, the in-flight count is compared against the high threshold value for this listener port.
- If the in-flight count is less than or equal to the high threshold value, then a work record is queued up onto the WLM queue.

- If the in-flight count exceeds the high threshold value, the queue agent thread that is running the MRH becomes blocked, entering a wait state, and we say that the throttle is “blocking”.

The in-flight count is decremented by 1 whenever the controller is notified that a work record for this Listener Port has completed (whether or not that the application’s transaction was committed). After being decremented, the in-flight count is checked against the low threshold value for this Listener Port. If the in-flight count drops down to the low threshold value, the previously-blocked queue agent thread is awoken (notified). At that point new work records can be queued onto the WLM queue and we say the throttle has been “released”.

The low threshold and high threshold values are set externally by one setting, the listener port’s Maximum sessions parameter. The high threshold value is set internally equal to the Maximum sessions value defined externally. The low threshold value is computed and set internally by the formula: `low threshold = (high threshold / 2)`, with the value rounding down to the nearest integer.

However, if the Message Listener service has been configured with the Custom Property of MDB.THROTTLE.THRESHOLD.LOW.EQUALS.HIGH defined and set to a value of “true”, then the low threshold value is set internally to the high threshold value (which is the externally-set Maximum sessions property of the listener port).

**Note:** Please consider the following information, although most users are not be affected by what is described. One queue agent thread is established for each destination, rather than for each listener port. Therefore, it two listener ports are mapped onto the same queue, a throttle-blocking condition on one listener port also results in a blocking of the second listener port’s queueing of work records. The second listener port is blocked even if it has not reached its high threshold value. For simplicity it is recommended that you do not share a destination across more than one listener port.

**MDB throttle – tuning considerations**

Consider the reason for recommending that you set the listener port “Maximum sessions” value to twice the number of worker threads available in all servants in the scalable server (2*WT). The reason is based on the goal that if you have an available worker thread in some servant, then you do not want to leave it idle because of a blocked MDB throttle. That is, you do not want to have an empty WLM queue, an available servant worker thread, and a blocked throttle.

With the basic recommendation of using the 2*WT value, then, a blocked throttle is released at the moment when the following conditions are true:
- There is one free servant worker thread
- There is nothing on the WLM queue
- There is one message reference browsed but for which a work request was not added to the WLM queue (the throttle blocked instead)

Furthermore, by setting the high threshold to 2*(WT+N) you can ensure that, at the moment a servant worker thread frees up and releases the throttle, there is a backlog of N messages pre-processed and sitting on the WLM queue ready for dispatch. A very high value would introduce the WLM queue overload problem which the throttle was introduced to avoid, but we do not have a specific upper limit to recommend here. (These tuning considerations assume the queue (or topic) is fully-loaded with messages to be processed.)

So, raising the high threshold value allows the server to create a small backlog of preprocessed messages sitting on the WLM queue if a workload spike occurs. The downside or negative of raising the high threshold value is that it increases the chance that a work record for a given message might time out before the application can be dispatched with the given message. (That is, the server might reach the MDB Timeout limit.) The given message is eventually re-delivered to the server, but only later and the processing done up until that time would be wasted. Also, a very large high threshold value would effectively bypass the MDB throttle function, in which case the WLM queue could be overloaded; this would cause the server to fail.

**MDB throttle – alternative tuning considerations**

Although the scalable server was designed with the goal of maximizing throughput, it is possible to use the listener port settings to achieve other workflow management goals.

For example, a high threshold setting of ‘1’ guarantees that messages are processed in the order that they are received onto the destination.

There might also be other business reasons, based on capacity or other factors, to restrict a particular listener port to much less concurrency than the server would otherwise support. While this is certainly a supported configuration, it might cause the throttle to block when there are idle worker threads available.

**MDB throttle example**

Suppose your server is configured with the maximum server instances value set to 3, with workload profile of IOBOUND. You have two CPUs, therefore WebSphere Application server will create six worker threads in each servant. Your application (a single MDB mapped to a queue) handles each message relatively quickly (so there is less risk of timeout) and you want the total time from arrival of a given message on your MDB queue until the end of MDB dispatch for this message to be as small as possible.

To provide a quick response time in the case of a surge in work, you opt for a bigger backlog. You set your Listener Port maximum sessions value to `100 =  2 * (3 * 6 + 32)`.

**Note:** Any value greater than or equal to `36 = 2 * 3 * 6` would keep all available servant worker threads busy. In practice it would probably not be worth the effort to pick the best possible “backlog factor”. Therefore, we make a mental estimate after giving the situation some thought, and pick a value so that we end up with the round number, 100.

***Connection factory settings for message-driven beans on z/OS:***

You can tune a variety of connection factory settings to control the creation of connections and sessions for MDB work.

This topic describes the following concepts and considerations that affect your choice of connection factory settings:
- “Connection factory settings” on page 12
- “Connection pool maximum connections settings” on page 12
- “Session pool maximum connections settings” on page 13
- “Should I use a few or many connection factories?” on page 13
- “Connection factories – examples” on page 14

**Connection factory settings**

To attach an application and a server to a particular queue manager with authentication parameters, applications and message listener ports are both bound to connection factories. The server uses the same administrative model to listen for messages arriving for delivery to message-driven beans as the application uses to exploit JMS, in that message listener ports are bound to a queue connection factory or topic connection factory and a corresponding queue or topic.

In addition to specifying the messaging provider settings (for example, queue manager settings) on connection factories, you also specify connection and session pool properties. In particular, for a connection factory, the values of the connection pool property Maximum connections and session pool property Maximum connections must be chosen with different considerations in mind, depending on whether you are using the connection factory for a listener port, for an application, or for both.

**Connection pool maximum connections settings**

First, in order to avoid waiting for a JMS connection, because of reaching a WebSphere Application Server-defined limit, the connection pool Maximum connections should be set to at least the sum of the following values:

1. One for each message-driven bean that is mapped to any listener port mapped onto this connection factory.
2. The maximum number of connections for one dispatch multiplied by the number of work threads in a servant.

   To determine this maximum connection count, first find the maximum number of connections obtained in a single application dispatch (typically ‘1’), checking all applications that use this connection factory. Multiply this maximum number of connections by the number of worker threads in a servant. (See the following note for more information about this value.)

Note that only a single servant’s worker threads are counted here, because each servant gets its own connection factory with connection and session pools, although the Administrator defines only a single set of property values.

**Clarification about part 2 of the preceding Maximum connection count**

**Note:**

A message-driven bean might or might not get a JMS connection in performing its application logic in onMessage(). For example, it might get a connection to forward the message to another destination or send a reply, or it might simply update some log and perform no JMS-related calls of its own.

In either case, we need to count ‘one’ for this message-driven bean in part 1 of the preceding Maximum connection count, as this is the connection used by the listener port. If the message-driven bean onMessage() logic gets one JMS connection, then we then add the number of servant worker threads to the Maximum connection count. If not, then we need not add to the Maximum connection count on behalf of this (MDB) application component.

Of course, other non-MDB application components which use this connection factory to perform JMS calls might cause the administrator to need to include part 2 of the overall Maximum connections count. But, regardless of how many MDB or non-MDB application components use this connection factory, if they each only use one JMS connection per dispatch, then the count in part 2 is only equal to the number of servant worker threads (not the number of applications multiplied by the number of servant worker threads).

**Session pool maximum connections settings**

This calculation is an easier one to make than it might first appear. For MDB listener port processing in all typical cases, the session pool Maximum connections property should be set to the number of worker threads in a single servant.

The reasons are the fact that JMS sessions are not shared across application dispatches or by the listener port infrastructure, even for clients of the same connection factory, along with the fact that there is a unique session pool for each JMS connection obtained from the connection factory (although the pool property settings are specified only once, at the connection factory level).

It is possible to imagine a case for which the same connection factory is used both by a listener port and an application, with the application having a higher Maximum connections requirement on the session pool setting, but this does not merit further discussion here.

**Important:** : It is possible to restrict the concurrent MDB work in a server by setting this session pool Maximum connections to less than the number of servant worker threads. This is not recommended. In such a case, an MDB work request could be dispatched over to a servant, without a session available to process the message. The worker thread at that point would then wait for a session to become available, tying up the valuable worker thread resource.

**Should I use a few or many connection factories?**

Some users may prefer to keep these calculations simple; for example, by creating a separate connection factory for each message-driven bean (in which case the connection pool Maximum connections property value could simply be set to 1). Others may prefer to manage fewer connection factory administrative objects.

The fact that the connections and sessions used for MDB processing cannot be shared (that is, cannot be used by more than one flow at a time) means that the desire to take advantage of pooling should not be considered a reason for using fewer connection factories. In other words, adding another connection factory does not prevent connection pooling that could otherwise be exploited by MDB listener port processing.

**Connection factories – examples**

**Example 1**

**Note:**

The scenario:

- Each servant has 12 worker threads. (The number of servants in the server is not important since each servant gets its own pools).
- Listener port LP1 is mapped to connection factory CF1. Message-driven bean MDB1 is mapped to LP1. The onMessage() application code of MDB1 puts a new message onto a forwarding queue, and so MDB1 has a resource reference which is also resolved to CF1.
- Also, in the same server, listener port LP2 is defined and mapped to connection factory CF2. Message-driven beans MDB2A and MDB2B are defined in the same ejb-jar file and are both mapped to LP2 with complementary JMS selectors. The onMessage() application code of MDB2A and MDB2B each does some logging, but neither message-driven bean makes any JMS API calls of its own.

The solution:

- For connection factory CF1, we count one for MDB1. The MDB1 application (which also uses connection factory CF1 to send its forwarding message) uses one JMS connection, for which we count the number of worker threads (12), multiplied by one. Our total connection pool Maximum connections for connection factory CF1, then, is `13 = 12 + 1`.
- For connection factory CF2, we count one for each of MDB2A and MDB2B. There are no applications using CF2, (only the listener port infrastructure), so we set the connection pool Maximum connections for connection factory CF2 equal to 2.
- For each of the two connection factories, the session pool Maximum connections value is set to 12.

**Example 2**

**Note:**

The scenario:

- Again, each servant has 12 worker threads. In this example we only want to use a single connection factory, CF1.
- Each of two listener ports LP1 and LP2 is mapped to connection factory CF1. The message-driven beans MDB1, MDB2, and MDB3 are part of three unique application EAR files. MDB1 is mapped to LP1, but MDB2 and MDB3 are each mapped to LP2.

The solution:

- Up to this point we count that we need three connections for connection factory CF1. However, there is also a servlet component which puts messages on a queue and it uses the same connection factory CF1. So for connection factory CF1, the connection pool Maximum connections setting is `15 = 3 + 12`.

***Message-driven beans, heterogeneous workloads, and workload management on z/OS:***

A message-driven bean can run on an application server that hosts a heterogeneous workload including other message-driven beans and non-MDB work items. To manage a heterogeneous workload on z/OS, you should use WLM classification and define unique service classes for different-priority work running in the same server.

In the set of topics under “Concepts and considerations for MDB settings on z/OS” on page 8, the explanations assume, for simplicity, that your server hosts a single message-driven bean, and that several MDB instances could be simultaneously running on all servant worker threads. Of course, in reality it is likely that any message-driven bean runs on an application server that hosts a heterogeneous workload including other message-driven beans, enterprise beans accessed through IIOP, work such as servlets and JSPs accessed through HTTP, and more work items.

Although the various MDB-related tuning controls provide a way to exert a fine-grained control over the amount of MDB work performed for a given message-driven bean or set of message-driven beans in a given server, we do not recommend using these settings to prioritize certain MDB work above or below other work in the server. Instead, we recommend using WLM classification and defining unique service classes for different-priority work running in the same server. Be sure to allow for at least one servant per unique service class.

This introduces complications in the case that work moving through a single listener port is configured with different selectors mapping onto different service classes. For more information about workload classification, see Classifying z/OS workload.

There is only one throttle for the listener port. This throttle controls the rate at which differently-prioritized messages get queued up as work records on to the WLM queue. The prioritization only occurs with the help of workload management, when the work records are queued up. It does not apply to the phase in which the messages are browsed by the queue agent thread and work records are queued up.

One strategy you can take in such a case is to raise the high threshold value (the value of the Maximum sessions property of the listener port) above the baseline recommendation of “twice the combined number of worker threads in all the server’s servants”. The rationale would be to make sure the WLM queue is loaded enough to allow workload management to decide which work to process next, rather than cutting off the queuing up of work requests. (Cutting off the queuing up of work requests can leave an overabundance of lower-priority work records on the WLM queue when higher priority messages are waiting to be browsed.)

You can develop other strategies beyond what can be discussed here.

Finally, it is also possible that both the following values can be known to some degree of certainty:

- The average number of servant worker threads processing a given message-driven bean
- The average number of available servants (some number between the minimum and maximum is started at any given time)

This could be computed perhaps using PMI, other monitoring tools, or perhaps even by a high-level understanding of how the message-driven bean fits into a greater application flow within a given server.

Should you adjust the baseline formula setting listener port maximum sessions to become twice the number of worker threads available for the *maximum number* of servants in the scalable server instead of twice the actual number of worker threads available in all servants? Although it is hard to give general recommendations, it may not be worth changing the formula. Lowering the setting introduces the possibility of idle worker threads, and a setting higher than strictly necessary only results in an extra buildup of messages on the WLM queue. The number of extra messages on the queue should still be a small enough number to prevent the serious problem of an overloaded WLM queue causing the server to fail.

## Best practices for debugging MDB throttle support

In order to have minimal impact on performance while still collecting needed debugging information for the MDB throttle support, use the following options:

- Trace option: com.ibm.ejs.jms.listener.MessageReferenceListenerPort=all=enabled

  The above trace option does not need to be specified if the MDB trace option is already set.
- System property: com.ibm.mdb.throttle.trace.enabled

  Disabled, if the property is not defined or set to 0.

  Enabled, if the property is set to 1.
- Dynamic trace support for statistics gathering and presentation.

  To enable, disable, or reset messaging statistics, use the following modify command:

  f *<server>*,mdbstats,[enable | disable | reset]

  The response from the console should be:

  ```
  BBOO0211I MODIFY COMMAND MDBSTATS, [ENABLE | DISABLE | RESET] COMPLETED SUCCESSFULLY
  ```

  The following message appears if statistics gathering is not enabled:

  ```
  BBOO0284I  STATISTICS GATHERING NOT ENABLED FOR <string>
  ```

  To display statistics, use the following modify display command:

  f *<server>*,display,work,mdb,stats

  The displayed information per listener port includes:

  **NAME** The name of the Listener Port for which the statistics are being displayed.

  **TIME** The amount of time, in seconds, since the stats were enabled or reset.

  **TOTAL**
  Total number of message references browsed since the stats were enabled or reset.

  **IN-FLIGHT**
  The current number of in-flight Work Requests.

  **EXCS** The number of exceptions while queuing the requests.

  **BLOCKS**
  The total number of instances the Throttle restricts queuing of work requests.

## Tuning TCP/IP buffer sizes

WebSphere Application Server uses the TCP/IP sockets communication mechanism extensively. For a TCP/IP socket connection, the send and receive buffer sizes define the receive window. The receive window specifies the amount of data that can be sent and not received before the send is interrupted. If too much data is sent, it overruns the buffer and interrupts the transfer. The mechanism that controls data transfer interruptions is referred to as flow control. If the receive window size for TCP/IP buffers is too small, the receive window buffer is frequently overrun, and the flow control mechanism stops the data transfer until the receive buffer is empty.

TCP/IP can be the source of some significant remote method delays. Follow these tips to tune TCP/IP:

- Recycle TCP/IP, then monitor CPU and paging rates to determine if they are within recommended system guidelines.
- First, ensure that you have defined enough sockets to your system and that the default socket time-out of 180 seconds is not too high. To allow enough sockets, update the BPXPRMxx parmlib member:
  1. Set MAXSOCKETS for the AF_INET filesystem high enough.
  2. Set MAXFILEPROC high enough.

     We recommend setting MAXSOCKETS and MAXFILEPROC to at least 5000 for low-throughput, 10000 for medium-throughput, and 35000 for high-throughput WebSphere transaction environments. Setting high values for these parameters should not cause excessive use of resources unless the sockets or files are actually allocated. **Example:**

     ```
     /* Open/MVS Parmlib Member                                      */
     /* CHANGE HISTORY:                                              */
     /*   01/31/02 AEK Increased MAXSOCKETS on AF_UNIX from 10000 to 50000*/
     /*                per request from My Developer                 */
     /*   10/02/01 JAB Set up shared HFS                             */

     /* KERNEL RESOURCES             DEFAULT            MIN MAX        */
     /* ========================     ================== === =========== */
        .
        .
        MAXFILEPROC(65535)          /* 64              3   65535      */

        .
        .
         NETWORK DOMAINNAME(AF_INET) DOMAINNUMBER(2) MAXSOCKETS(30000)
        .
     ```
- Next check the specification of the port in TCPIP profile dataset to ensure that NODELAYACKS is specified as follows:

  ```
  PORT 8082 TCP NODELAYACKS
  ```

  In your runs, changing this could improve throughput by as much as 50% (this is particularly useful when dealing with trivial workloads). This setting is important for good performance when running SSL.
- You should ensure that your DNS configuration is optimized so that lookups for frequently-used servers and clients are being cached.

  Caching is sometimes related to the name server's Time To Live (TTL) value. On the one hand, setting the TTL high will ensure good cache hits. However, setting it high also means that, if the Daemon goes down, it will take a while for everyone in the network to be aware of it.

  A good way to verify that your DNS configuration is optimized is to issue the oping and onslookup USS commands. Make sure they respond in a reasonable amount of time. Often a misconfigured DNS or DNS server name will cause delays of 10 seconds or more.
- Increase the size of the TCPIP send and receive buffers from the default of 16K to at least 64K. This is the size of the buffers including control information beyond what is present in the data that you are sending in your application. To do this specify the following:

```
TCPCONFIG TCPSENDBFRSIZE 65535
          TCPRCVBUFRSIZE 65535
```

**Note:** It is unreasonable, in some cases, to specify 256K buffers.

- Increase the default listen backlog. This is used to buffer spikes in new connections which come with a protocol like HTTP. The default listen backlog is 10 requests. We recommend that you increase this value to something larger. For example:

  ```
  protocol_http_backlog=100
  protocol_https_backlog=100
  protocol_iiop_backlog=100
  protocol_ssl_backlog=100
  ```

- Reduce the finwait2 time. In the most demanding benchmarks you may find that even defining 65K sockets and file descriptors does not give you enough 'free' sockets to run 100%. When a socket is closed abnormally (for example, no longer needed) it is not made available immediately. Instead it is placed into a state called finwait2 (this is what shows up in the netstat -s command). It waits there for a period of time before it is made available in the free pool. The default for this is 600 seconds.

  **Note:** Unless you have trouble using up sockets, we recommend that you leave this set to the default value.

  If you are using z/OS V1.2 or above, you can control the amount of time the socket stays in finwait2 state by specifying the following in the configuration file:

  ```
  FINWAIT2TIME 60
  ```

## Tuning Java virtual machines

WebSphere Application Server is a Java based server and requires a Java virtual machine (JVM) environment to run and support the Java and J2EE applications that run on it. As part of configuring WebSphere Application Server, you can configure the Java runtime environment to tune performance and system resource usage.

On the z/OS platform there is a JVM in both the controller and servant. This information applies to the JVM in the servant. Usually the JVM in the controller does not need to be tuned.

A Java runtime environment provides the execution environment for Java based applications and servers such as WebSphere Application Server. Therefore the Java configuration plays a significant role in determining performance and system resource consumption for WebSphere Application Server and the applications that run on it.

WebSphere Application Server supports Java runtime environments from different JVM providers depending on the platform. This includes

- IBM JVMs.

  The IBM Java 5.0 and newer versions include major improvements in virtual machine technology to provide significant performance and serviceability enhancements over IBM's earlier Java execution technology. See http://www.ibm.com/software/webservers/appserv/was/performance.html for more information about this new technology.

To determine the JVM provider on which your application server is running, issue the java –fullversion command from within your application server *app_server_root*/java/bin directory. In response to this command, the application server writes information about the JVM, including the JVM provider information, into the `SystemOut.log` file.

The following steps provide specific instructions on how to perform the following types of tuning for each JVM. The steps do not have to be performed in any specific order.

- Java memory or heap tuning
- Garbage collection tuning
- Start up versus runtime performance optimization

Other significant tuning options not described here can be found in the Java technology guides from each vendor. Links to vendor specific tuning information are included in the steps.

1. **Configure the heap size**

   Java memory or heap tuning controls the amount of memory that is allocated for use by individual application server instances. The following command line parameters apply to all supported JVMs and are used to adjust the minimum and maximum heap size for each application server instance. The *IBM Developer Kit and Runtime Environment, Java2 Technology Edition, Version 5.0 Diagnostics Guide*, that is available on the developerWorks Web site, provides additional information on tuning the heap size.

   To use the administrative console to configure the heap size:

   a. In the administrative console, click **Servers > Application Servers >** *server*.

   b. Under Server Infrastructure, click **Java and Process Management > Process Definition > Java Virtual Machine**.

   c. Specify a new value in either the Initial heap size or the Maximum heap size field. You can also specify values for both fields if you need to adjust both settings.

   d. Click **OK**.

   e. Save your changes to the master configuration.

   f. Stop and restart the application server.

   You can also use the following command line parameters to adjust these settings:

   - **-Xms**

     This setting controls the initial size of the Java heap. Properly tuning this parameter reduces the overhead of garbage collection, which improves server response time and throughput. For some applications, the default setting for this option might be too low, which causes a high number of minor garbage collections.

| | |
|---|---|
| **Default:** | 50MB |
| **Recommended:** | Workload specific, but higher than the default. |
| **Usage:** | -Xms256m sets the initial heap size to 256 megabytes. |

   - **-Xmx**

     This setting controls the maximum size of the Java heap. Increasing this parameter increases the memory available to the application server, and reduces the frequency of garbage collection. Increasing this setting can improve server response time and throughput. However, increasing this setting also increases the duration of a garbage collection when it does occur. This setting should never be increased above the system memory available for the application server instance. Increasing the setting above the available system memory can cause system paging and a significant decrease in performance.

| | |
|---|---|
| **Default:** | 256MB |
| **Recommended:** | Workload specific, but higher than the default, depending on the amount of physical memory available. |
| **Usage:** | -Xmx512m sets the maximum heap size to 512 megabytes. |

   - **-Xlp**

This setting can be used with the IBM JVM to allocate the heap when using large pages (16MB). However, if you use this setting your operating system must be configured to support large pages. Using large pages can reduce the CPU overhead needed to keep track of heap memory, and might also allow the creation of a larger heap.

See "Tuning operating systems" on page 66 for more information about tuning your operating system.

2. **Tune garbage collection**

   You can use JVM settings to configure the type and behavior of garbage collection. When the JVM cannot allocate an object from the current heap because of lack of contiguous space, the garbage collector is invoked to reclaim memory from Java objects that are no longer being used. Each JVM vendor provides unique garbage collector policies and tuning parameters.

   To adjust your JVM garbage collection settings:

   a. In the administrative console, click **Servers > Application Servers >** *server*.
   b. Under Server Infrastructure, click **Java and Process Management > Process Definition > Java Virtual Machine**.
   c. Enter the `–X` option you want to change in the Generic JVM arguments field.
   d. Click **OK**.
   e. Save your changes to the master configuration.
   f. Stop and restart the application server.

   The following steps describe specific the `–X` options that the different JVM garbage collectors support.

   a. Tune the IBM JVM garbage collector. A complete guide to the IBM Java garbage collector is provided in the *IBM Developer Kit and Runtime Environment, Java2 Technology Edition, Version 5.0 Diagnostics Guide*. This document is available on the developerWorks Web site.

      Use the Java -X option to view a list of memory options.

      - **-Xgcpolicy**

        Starting with Java 5.0, the IBM JVM provides four policies for garbage collection. Each policy provides unique benefits.

        – `optthruput`, which is the default, provides high throughput but with longer garbage collection pause times. During a garbage collection, all application threads are stopped for mark, sweep and compaction, when compaction is needed. `optthruput` is sufficient for most applications.

        – `optavgpause`, which reduces garbage collection pause time by performing the mark and sweep phases of garbage collection concurrently with application execution. This concurrent execution cause a small performance impact to overall throughput.

        – `gencon`, which is new in IBM Java 5.0, is a generational garbage collector for the IBM JVM. The generational scheme attempts to achieve high throughput along with reduced garbage collection pause times. To accomplish this goal, the heap is split into new and old segments. Long lived objects are promoted to the old space while short-lived objects are garbage collected quickly in the new space. The `gencon` policy provides significant benefits for many applications, but is not suited to all applications and is generally more difficult to tune.

        – `subpool`, which can increase performance on multiprocessor systems, that commonly use more then 8 processors. This policy is only available on IBM pSeries and zSeries processors. The `subpool` policy is similar to the `optthruput` policy except that the heap is divided into subpools that provide improved scalability for object allocation.

| | |
|---|---|
| **Default:** | optthruput |
| **Recommended:** | optthruput |
| **Usage:** | Xgcpolicy:optthruput sets the garbage collection to optthruput |

Setting **gcpolicy** to `optthruput` disables concurrent mark. You should get the best throughput results when you use the `optthruput` policy unless you are experiencing erratic application response times, which is an indication that you might have pause time problems

Setting **gcpolicy** to `optavgpause` enables concurrent mark with its default values. This setting alleviates erratic application response times that normal garbage collection causes. However, this option might decrease overall throughput.

- **-Xnoclassgc**

  By default, the JVM unloads a class from memory whenever there are no live instances of that class left. Class unloading can degrade performance. Turning off class garbage collection eliminates the overhead of loading and unloading the same class multiple times.

| | |
|---|---|
| **Default:** | Class garbage collection is enabled. |
| **Recommended:** | Class garbage collection is disabled. |
| **Usage:** | Xnoclassgc disables class garbage collection. |

See *IBM Developer Kit and Runtime Environment, Java2 Technology Edition, Version 5.0 Diagnostics Guide*. on the developerWorks Web site for more information about class garbage collection.

3. **Optimize the startup and runtime performance**

   In some environments, such as a development environment, it is more important to optimize the startup performance of your application server rather than the runtime performance. In other environments, it is more important to optimize the runtime performance. By default, IBM JVMs are optimized for runtime performance, while HotSpot based JVMs are optimized for startup performance.

   The Java JIT compiler has a big impact on whether startup or runtime performance is optimized. The initial optimization level that the compiler uses influences the length of time it takes to compile a class method, and the length of time it takes to start the server. For faster startups, you should reduce the initial optimization level that the compiler uses. However if you reduce the initial optimization level, the runtime performance of your applications might be degraded because the class methods are now compiled at a lower optimization level.

   - **-Xquickstart**

     This setting influences how the IBM JVM uses a lower optimization level for class method compiles. A lower optimization level provides for faster server startups, but lowers runtime performance. If this parameter is not specified, the IBM JVM defaults to starting with a high initial optimization level for compiles, which results in faster runtime performance, but slower server starts.

| | |
|---|---|
| **Default:** | High initial compiler optimization level |
| **Recommended:** | High initial compiler optimization level |
| **Usage:** | -Xquickstart provides faster server startup. |

   JVMs based on the Sun HotSpot technology initially compile class methods with a low optimization level. Use this JVM option to change that behavior:

4. **Share classes in a cache.**

   The share classes option of the IBM Java 2 Runtime Environment (J2RE) Version 1.5.0 lets you share classes in a cache. Sharing classes in a cache can improve startup time and reduce memory footprint. Processes, such as application servers, node agents, and deployment managers, can use the share classes option.

   If you use this option, you should clear the cache when the process is not in use. To clear the cache, either call the *app_server_root*/bin/clearClassCache.bat/sh utility or stop the process and then restart the process.

   If you need to disable the share classes option for a process, specify the generic JVM argument -Xshareclasses:none for that process:

   a. In the administrative console, click **Servers > Application Servers >** *server*.

b. Under Server Infrastructure, click **Java and Process Management > Process Definition > Java Virtual Machine**.
c. Enter `-Xshareclasses:none` in the Generic JVM arguments field.
d. Click **OK**.
e. Save your changes to the master configuration.
f. Stop and restart the application server.

| | |
|---|---|
| **Default:** | The Share classes in a cache option is enabled. |
| **Recommended:** | Leave the share classes in a cache option enabled. |
| **Usage:** | -Xshareclasses:none disables the share classes in a cache option. |

Each Java vendor provides detailed information on performance and tuning. Use the following Web sites to obtain additional tuning information for a specific Java runtime environments:

- For the IBM Developer Kit for Java, see:
  - http://www.ibm.com/developerworks/java/
  - http://www.ibm.com/developerworks/java/jdk/diagnosis/
- For the Sun Java JDK, see http://java.sun.com/docs/performance/.

Also see Java memory tuning tips for additional tuning information.

## Tuning transport channel services

The transport channel services manage client connections and I/O processing for HTTP and JMS requests. These I/O services are based on the non-blocking I/O (NIO) features that are available in Java™. These services provide a highly scalable foundation to WebSphere Application Server request processing. Java NIO based architecture has limitations in terms of performance, scalability and end user usability. Therefore, integration of true asynchronous I/O is implemented. This implementation provides significant benefits in usability, reduces the complexity of I/O processing and reduces that amount of performance tuning you have to perform.

Key features of the new transport channel services include:

- Scalability, which enables the WebSphere Application Server to handle many concurrent requests.
- Asynchronous request processing, which provides a many-to-one mapping of client requests to Web container threads
- Resource sharing and segregation, which enables thread pools to be shared between the Web container and a messaging service.
- Improved usability and
- Incorporation of autonomic tuning and configuration functions.

Changing the default values for settings on one or more of the transport channels associated with a transport chain can improve the performance of that chain.

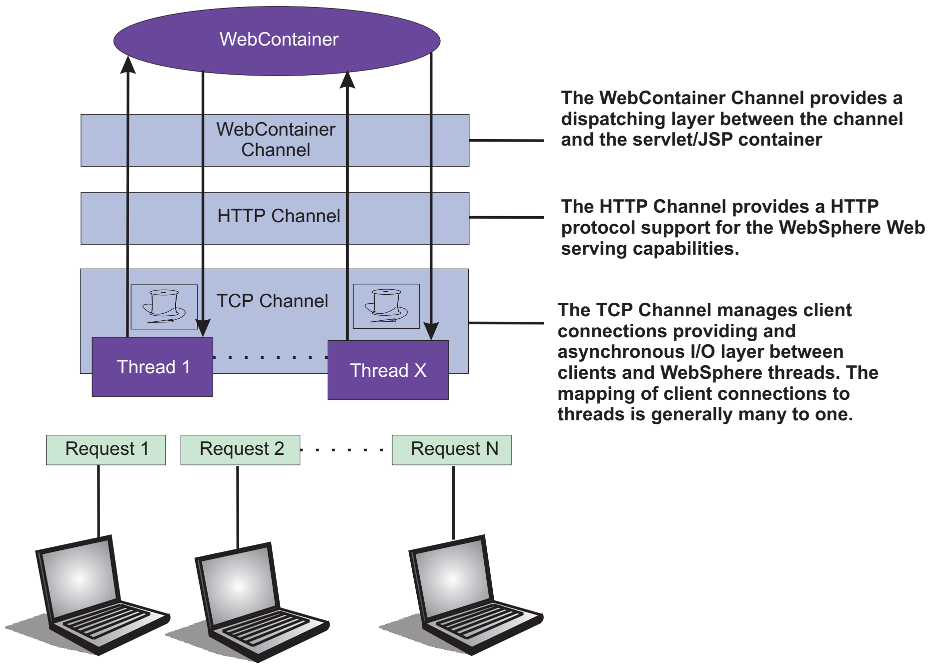
WebContainer
WebContainer Channel
HTTP Channel
TCP Channel
Thread 1
Thread X
Request 1
Request 2
Request N
The WebContainer Channel provides a dispatching layer between the channel and the servlet/JSP container
The HTTP Channel provides a HTTP protocol support for the WebSphere Web serving capabilities.
The TCP Channel manages client connections providing and asynchronous I/O layer between clients and WebSphere threads. The mapping of client connections to threads is generally many to one.

*Figure 1. Transport Channel Service*

- **Adjust TCP transport channel settings.** In the administration console, click **Servers > Application servers >** *server_name* **> Ports**. Then click **View associated transports** for the appropriate port.
  1. Select the transport chain whose properties you are changing.
  2. Click on the TCP transport channel defined for that chain.
  3. Leave the Maximum open connections parameter set to the default value. This parameter controls the maximum number of connections that are available for a server's use. It should be left at the default value of 20000, which is the maximum number of connections allowed. The transport channel service by default manages high client connection counts and requires no tuning.
  4. If client connections are being closed without data being written back to the client, change the value specified for the Inactivity timeout parameter. This parameter controls the maximum number of connections available for a server's use. Upon receiving a new connection, the TCP transport channel waits for enough data to arrive to dispatch the connection to the protocol specific channels above the TCP transport channel. If not enough data is received during the time period specified for the Inactivity timeout parameter, the TCP transport channel closes the connection.

     The default value for this parameter is 60 seconds, which is adequate for most applications. You should increase the value specified for this parameter if your workload involves a lot of connections and all of these connections can not be serviced in 60 seconds.
- **Adjust HTTP transport channel settings.** In the administration console, click **Servers > Application servers >** *server_name* **> Ports**. Then click **View associated transports** for the appropriate port.
  1. Select the transport chain whose properties you are changing.
  2. Click on the HTTP transport channel defined for that chain.
  3. Tune HTTP keep-alive. The Use persistent (keep-alive) connections setting controls whether or not connections are left open between requests. Leaving the connections open can save setup and teardown costs of sockets if your workload has clients that send multiple requests. The default value is true and is the optimal setting in most cases.

     If your clients only send single requests over substantially long periods of time, it is probably better to disable this option and close the connections right away rather than to have the HTTP transport channel setup the timeouts to close the connection at some later time.

4. Change the value specified for the Maximum persistent requests parameter to increase the number of requests that can flow over a connection before it is closed. When the Use persistent connections option is enabled, the Maximum persistent requests parameter controls the number of requests that can flow over a connection before it is closed. The default value is 100. This value should be set to a value such that most, if not all, clients always have an open connection when they make multiple requests during the same session. A proper setting for this parameter helps to eliminate unnecessary setting up and tearing down of sockets.

   For test scenarios in which the client will never close a socket or where sockets are always proxy or Web servers in front of your application server, a value of -1 will disable the processing which limits the number of requests over a single connection. The persistent timeout will still shutdown some idle sockets and protect your server from running out of open sockets.
5. Change the value specified for the Persistent timeout parameter to increase the length of time that a connection is held open before being closed due to inactivity. The Persistent timeout parameter controls the length of time that a connection is held open before being closed because there is no activity on that connection. The default value is 30 seconds This parameter should be set to a value that keeps enough connections open so that most clients can obtain a connection available when they need to make a request.
6. If clients are having trouble completing a request because it takes them more than 60 seconds to send their data, change the value specified for the Read timeout parameter. Some clients pause more than 60 seconds while sending data as part of a request. To ensure they are able to complete their requests, change the value specified for this parameter to a length of time in seconds that is sufficient for the clients to complete the transfer of data. Be careful when changing this value that you still protect the server from clients who send incomplete data and thereby utilize resources (sockets) for an excessive amount of time.
7. If some of your clients require more than 60 seconds to receive data being written to them, change the value specified for the Write timeout parameter. Some clients are slow and require more than 60 seconds to receive data that is sent to them. To ensure they are able to obtain all of their data, change the value specified for this parameter to a length of time in seconds that is sufficient for all of the data to be received. Be careful when changing this value that you still protect the server from malicious clients.

- **Adjust Web container transport channel settings.** In the administration console, click **Servers > Application servers >** *server_name* **> Ports**. Then click **View associated transports** for the appropriate port.
  1. Select the transport chain whose properties need to be changed.
  2. Click on the Web container transport channel defined for that chain.
  3. If multiple writes are required to handle responses to the client, change the value specified for the Write buffer size parameter to a value that is more appropriate for your clients. The Write buffer size parameter controls the maximum amount of data per thread that the Web container buffers before sending the request on for processing. The default value is 32768 bytes, which is sufficient for most applications. If the size of a response is greater than the size of the write buffer, the response is chunked and written back in multiple TCP writes.

     If you need to change the value specified for this parameter, make sure the new value enables most requests to be written out in a single write. To determined an appropriate value for this parameter, look at the size of the pages that are returned and add some additional bytes to account for the HTTP headers.
- **Adjust the settings for the bounded buffer.**

  Even though the default bounded buffer parameters are optimal for most of the environments, you might need to change the default values in certain situations and for some operating systems to enhance performance. Changing the bounded buffer parameters can degrade performance. Therefore, make sure that you tune the other related areas, such as the Web container and ORB thread pools, before deciding to change the bounded buffer parameters.

  To change the bounded buffer parameters:
  1. In the administrative console, click **Servers > Application Servers >** *server*.

2. Under Server Infrastructure, click **Java and Process Management > Process Definition > Java Virtual Machine**.
3. Specify one of the following parameters in the Generic JVM arguments field.
4. Click **Apply** or **OK**.
5. Enter one of the following custom properties in the Name field and an appropriate value in the Value field, and then click **Apply** to save the custom property and its setting.
   – com.ibm.ws.util.BoundedBuffer.spins_take=*value*

     Specifies the number of times a Web container thread is allowed to attempt to retrieve a request from the buffer before the thread is suspended and enqueued. This parameter enables you to trade off the cost of performing possibly unsuccessful retrieval attempts, with the cost to suspending a thread and activating it again in response to a put operation.

| | |
|---|---|
| **Default:** | 4 |
| **Recommended:** | Any non-negative integer value is allowed. In practice an integer between 2 and 8 have shown the best performance results. |
| **Usage:** | com.ibm.ws.util.BoundedBuffer.spins_take=6. Six attempts are made before the thread is suspended. |

   – com.ibm.ws.util.BoundedBuffer.yield_take=true or false

     Specifies that a thread yields the CPU to other threads after a set number of attempts to take a request from the buffer. Typically a lower number of attempts is preferable.

| | |
|---|---|
| **Default:** | false |
| **Recommended:** | The effect of yield is implementation specific for individual platforms. |
| **Usage:** | com.ibm.ws.util.BoundedBuffer.spins_take=*boolean value* |

   – com.ibm.ws.util.BoundedBuffer.spins_put=*value*

     Specifies the number of attempts an InboundReader thread makes to put a request into the buffer before the thread is suspended and enqueued. This value allows to trade off between the cost of repeated, possibly unsuccessful, attempts to put a request into the buffer with the cost to suspend a thread and reactivate it in response to a take operation.

| | |
|---|---|
| **Default:** | 4 |
| **Recommended:** | Any non-negative integer value is allowed. In practice an integer between 2 and 8 have shown the best performance results. |
| **Usage:** | com.ibm.ws.util.BoundedBuffer.spins_put=6. Six attempts are made before the thread is suspended. |

   – com.ibm.ws.util.BoundedBuffer.yield_put=true or false

     Specifies that a thread yields the CPU to other threads after a set number of attempts to put a request into the buffer. Typically a lower number of attempts is preferable.

| | |
|---|---|
| **Default:** | false |
| **Recommended:** | The effect of yield is implementation specific for individual platforms. |
| **Usage:** | com.ibm.ws.util.BoundedBuffer.yield_put=*boolean value* |

   – com.ibm.ws.util.BoundedBuffer.wait=*number of milliseconds*

Specifies the maximum length of time, in milliseconds, that a request might unnecessarily be delayed if the buffer is completely full or if the buffer is empty.

| | |
|---|---|
| **Default:** | 10000 milliseconds |
| **Recommended:** | A value of 10000 milliseconds usually works well. In rare instances when the buffer becomes either full or empty, a smaller value guarantee a more timely handling of requests, but there is usually a performance impact to using a smaller value. |
| **Usage:** | com.ibm.ws.util.BoundedBuffer.wait=8000. A request might unnecessarily be delayed up to 8000 milliseconds. |

- Click **Apply** and then **Save** to save these changes.

## Checking hardware configuration and settings

An optimal hardware configuration enables applications to get the greatest benefit from performance tuning. The hardware speed impacts all types of applications and is critical to overall performance.

The following parameters include considerations for selecting and configuring the hardware on which the application servers run.

- **Optimize disk speed**
  - **Description:** Disk speed and configuration have a dramatic effect on the performance of application servers running applications that are heavily dependent on the database support, using extensive messaging, or processing workflow. The disk input or output subsystems that are optimized for performance, for example Redundant Array of Independent Disks (RAID) array, high-speed drives, and dedicated caches, are essential components for optimum application server performance in these environments.

    Application servers with fewer disk requirements can benefit from a mirrored disk drive configuration that improves reliability and has good performance.
  - **Recommendation:** Spread the disk processing across as many disks as possible to avoid contention issues that typically occur with 1- or 2-disk systems. Placing the database tables on disks that are separate from the disks that are used for the database log files reduces disk contention and improve throughput.
- **Increase processor speed and processor cache**
  - **Description:** In the absence of other bottlenecks, increasing the processor speed often helps throughput and response times. A processor with a larger L2 or L3 cache yields higher throughput, even if the processor speed is the same as a CPU with a smaller L2 or L3 cache.
- **Increase system memory**
  - **Description:** The amount of storage required for z/OS is mostly dependent on the number of servers and the size of the Java Virtual Machine (JVM) heap for each server.
  - **Recommendation:** For a single server with 1 GB JVM heap, allocate a minimum of 1GB of memory.
- **Run network cards and network switches at full duplex**
  - **Description:** Run network cards and network switches at full duplex and use the highest supported speed. Full duplex is much faster than half duplex. Verify that the network speed of adapters, cables, switches, and other devices can accommodate the required throughput. Some Web sites might require multiple gigabit links.
  - **Recommendation** Make sure that the highest speed is in use on 10/100/1000 Ethernet networks.
- An **IBM S/390 or zSeries Model** that supports the software requirement of z/OS V1R2.
- **Storage**
  - Storage requirements are higher than for traditional workloads
  - **Recommendation**

- Virtual storage default should be about 370 MB per servant, which includes a 256 MB default heap size and a default initial LE heap size of 80 MB.
      - Real storage minimum is 512 MB per LPAR for a light load such as the IVP. For most real-world applications, we recommend 2 GB or higher. We have seen applications that require as much as 8 GB of real to operate at peak load.
- **DASD**
  - **Recommendation**
    - To maximize your performance, we recommend a fast DASD subsystem (for example, IBM Shark), running with a high cache read/write hit rate.
- **Networking**
  - **Recommendation**
    - For high bandwidth applications, we recommend at least a 1 Gb Ethernet connection. If your applications have extremely high bandwidth requirements, you may need additional Ethernet connections.

## Tuning operating systems

Use this page to determine your operating system and configure tuning specifications.

The following tuning parameters are specific to operating systems. Because these operating systems are not WebSphere Application Server products, be aware that the products can change and results can vary.

**Note:** Check your operating system documentation to determine how to make the tuning parameters changes permanent and if a reboot is required.

1. Determine your operating system.
2. Select your operating system from the related links section.
3. Configure your settings to optimize performance of Websphere Application Server.

## Tuning the z/OS operating system

Use these steps to tune your z/OS operating system to optimize WebSphere Application Server performance.

1. “Tuning storage” on page 74
2. “z/OS operating system tuning tips”
3. “UNIX System Services (USS) tuning tips for z/OS” on page 72
4. “Workload management (WLM) tuning tips for z/OS” on page 76
5. “Resource Recovery Service (RRS) tuning tips for z/OS” on page 67

### z/OS operating system tuning tips

There are several configuration changes you can make to z/OS system components that might improve WebSphere Application Server performance on the z/OS platform.

You might want to make one or more of the following changes to the indicated z/OS components:

- CTRACE

  The first place to review is your CTRACE configuration. Ensure that all components are either set to MIN or OFF. To display the CTRACE options for all components on your system, issue the following command from the operator console:

  ```
  D TRACE,COMP=ALL
  ```

  To change the setting for an individual component to its minimum tracing value, use the following command, where xxx is the component ID.

  ```
  TRACE CT,OFF,COMP=xxx
  ```

This configuration change eliminates the unnecessary overhead of collecting trace information that is not needed. Often during debug, CTRACE is turned on for a component and not shut off when the problem is resolved.

- SMF

  Ensure that you are not collecting more SMF data than you need. Review the SMFPRM*xx* settings to ensure that only the minimum number of records are collected.

  **Use SMF 92 or 120 only for diagnostics.**

  – SMF Type 92

    SMF Type 92 records are created each time an HFS file is opened, closed, deleted, and so forth. Almost every web server request references HFS files, so thousands of SMF Type 92 records are created. Unless you specifically need this information, turn off SMF Type 92 records. In the following example, we have disabled the collection of SMF type 92 records:

    **Example:**

    ```
    ACTIVE,
    DSNAME(SYS1.&.SYSNAME..SMF.MAN1;SYS1.&SYSNAME..SMF.MAN2;),
    NOPROMPT,
    REC(PERM),
    MAXDORM(3000),
    STATUS(010000),
    JWT(0510),
    SID(&SYSNAME;(1:4)),
    LISTDSN,
    SYS(NOTYPE(19,40,92)),
    INTVAL(30),
    SYNCVAL(00),
    SYS(DETAIL,INTERVAL(SMF,SYNC)),
    SYS(EXITS(IEFACTRT,IEFUJI,IEFU29,IEFU83,IEFU84,IEFU85,IEFUJV,IEFUSI))
    ```

  – SMF Type 120

    You may find that running with SMF 120 records in production is appropriate, since these records give information specific to WebSphere applications such as response time for J2EE artifacts, bytes transferred, and so forth. If you do choose to run with SMF 120 records enabled, we recommend that you use server interval SMF records and container interval SMF records rather than server activity records and container activity records. See the *Troubleshooting and support* PDF for a description of the SMF 120 record.

    The *Troubleshooting and support* PDF also describes the steps involved in controlling collection of SMF 120 records. To enable specific record types, specify the following properties:

    - server_SMF_server_activity_enabled=0 (or server_SMF_server_activity_enabled = false)
    - server_SMF_server_interval_enabled=1 (or server_SMF_server_interval_enabled = true)
    - server_SMF_container_activity_enabled=0 (or false)
    - server_SMF_container_interval_enabled=1 (or true)
    - server_SMF_interval_length=1800

- You might also want to review your DB2 records and the standard RMF written SMF records, and ensure that the SMF data sets are allocated optimally. DB2 SMF records 100, 101 and 102 affect performance and should be used only for monitoring DB2 performance with the DB2 PM tool. If you are not monitoring DB2 performance, you should consider not collecting those SMF records.

## Resource Recovery Service (RRS) tuning tips for z/OS

Use these tips to tune your z/OS operating system to optimize WebSphere Application Server performance.

- For best throughput, use coupling facility (CF) logger.

  DASD logger can limit your throughput because it is I/O-sensitive. The CF logger has much more throughput (in one measurement, the CF logger was six times faster than the DASD logger). Throughput will benefit from moving the RRS logs in logger to a coupling facility (CF) logstream. Doing so will help transactions complete quickly and not require any DASD I/O. If it's not possible to use CF logs, use well performing DASD and make sure the logs are allocated with large CI sizes.

- For best throughput, use coupling facility (CF) logger for the RRS log.

DASD logger can limit your throughput because it is I/O-sensitive. The CF logger has much more throughput (in one measurement, the CF logger was six times faster than the DASD logger). Throughput will benefit from moving the RRS logs in logger to a coupling facility (CF) logstream. Doing so will help transactions complete quickly and not require any DASD I/O. If it's not possible to use CF logs, use well performing DASD and make sure the logs are allocated with large CI sizes.

- Ensure that your CF logger configuration is optimal by using SMF 88 records.

  See the tuning section of *z/OS MVS Setting Up a Sysplex* or the chapter on System Logger Accounting in *z/OS MVS System Management Facilities (SMF)* for details. In any case, you should monitor the logger to ensure that there is a sufficient size in the CF and that offloading is not impacting the overall throughput. The transaction logs are shared I/O-intensive resources in the mainline and can affect throughput dramatically if mistuned.
- Set adequate default values for the LOGR policy.

  Default values of LOGR policy may have an impact on performance. We recommend the default settings in the table below.

*Table 1. Recommended default setting for LOGR*

| Log Stream | Initial Size | Size |
|---|---|---|
| RM.DATA | 1 MB | 1MB |
| MAIN.UR | 5 MB | 50 MB |
| DELAYED .UR | 5 MB | 50 MB |
| RESTART | 1 MB | 5 MB |
| ARCHIVE | 5 MB | 50 MB |

- Review XA Resource Managers log sizes.

  If you are using XA Resource Managers and you have chosen to put the logs in the logger, you may have to review the log sizes. As of this writing, we cannot give specific recommendations.

  You can configure the XA logs in the install dialog to live either in the HFS or in logstreams. If you are not using global transactions involving XA resources, there is no point in putting the log in a logstream. If the XA logs are placed in logstreams, we recommend that they be in the Coupling Facility instead of DASD. The default names are 'HLQ.server.M' and 'HLQ.server.D' where HLQ is a user-defined value between 1-8 characters specified in the install dialog, and 'server' is the server short name. It is the installer's responsibility to ensure that the HLQ + server name is unique across the configuration. If it is not, the server will fail to start because the user data in the existing logstream will not match that of the new server. The logs (and structures, if applicable) are created in job 'BBOLOGSA' in the install dialog. If structures need to be allocated, there is also a step indicating what structure names need to be added to the CFRM policy. We recommend 5MB initial and 20MB max sizes for both of these logstreams.
- Eliminate archive log if not needed.

  If you don't need the archive log, we recommend that you eliminate it since it can introduce extra DASD I/Os. The archive log contains the results of completed transactions. Normally, the archive log is not needed. Following is an example of disabling archive logging.

  **Example:**

```
//STEP1 EXEC PGM=IXCMIAPU
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
    DATA TYPE(LOGR)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.ARCHIVE)

    DELETE LOGSTREAM NAME(ATR.WITPLEX.MAIN.UR)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.RESTART)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.RM.DATA)
    DELETE LOGSTREAM NAME(ATR.WITPLEX.DELAYED.UR)
    DELETE STRUCTURE NAME(RRSSTRUCT1)
/*
```

```
//STEP2 EXEC PGM=IXCMIAPU
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
    DATA TYPE(LOGR)
    DEFINE STRUCTURE NAME(RRSSTRUCT1)
           LOGSNUM(9)


    DEFINE LOGSTREAM NAME(ATR.WITPLEX.MAIN.UR)
           STRUCTNAME(RRSSTRUCT1)
           STG_DUPLEX(YES)
           DUPLEXMODE(UNCOND)
           LS_DATACLAS(SYSPLEX)
           LS_STORCLAS(LOGGER)
           HLQ(IXGLOGR)
           AUTODELETE(YES)
           RETPD(3)
    DEFINE LOGSTREAM NAME(ATR.WITPLEX.RESTART)
           STRUCTNAME(RRSSTRUCT1)
           STG_DUPLEX(YES)
           DUPLEXMODE(UNCOND)
           LS_DATACLAS(SYSPLEX)
           LS_STORCLAS(LOGGER)
           HLQ(IXGLOGR)
           AUTODELETE(YES)
           RETPD(3)
    DEFINE LOGSTREAM NAME(ATR.WITPLEX.RM.DATA)
           STRUCTNAME(RRSSTRUCT1)
           STG_DUPLEX(YES)
           DUPLEXMODE(UNCOND)
           LS_DATACLAS(SYSPLEX)
           LS_STORCLAS(LOGGER)
           HLQ(IXGLOGR)
           AUTODELETE(YES)
           RETPD(3)
    DEFINE LOGSTREAM NAME(ATR.WITPLEX.DELAYED.UR)
           STRUCTNAME(RRSSTRUCT1)
           STG_DUPLEX(YES)
           DUPLEXMODE(UNCOND)
           LS_DATACLAS(SYSPLEX)
           LS_STORCLAS(LOGGER)
           HLQ(IXGLOGR)
           AUTODELETE(YES)
           RETPD(3)
/*

//* DEFINE LOGSTREAM NAME(ATR.WITPLEX.ARCHIVE)
           //* STRUCTNAME(RRSSTRUCT1)
           //* STG_DUPLEX(YES)
           //* DUPLEXMODE(UNCOND)
           //* LS_DATACLAS(SYSPLEX)
           //* LS_STORCLAS(LOGGER)
           //* HLQ(IXGLOGR)
           //* AUTODELETE(YES)
           //* RETPD(3)
```

## LE tuning tips for z/OS

Enabling xplink in the runtime environment and compiling applications with xplink enabled improves performance in z/OS V1R2.

- For best performance, use the LPALSTxx parmlib member to ensure that LE and C++ runtimes are loaded into LPA, as shown in the following example:

  **Example:** sys1.parmlib(LPALSTxx):

```
********************************* Top of Data ********************
USER.LPALIB,
ISF.SISFLPA,                        SDSF
CEE.SCEELPA,
                        LANGUAGE ENVIRONMENT
CBC.SCLBDLL,                        C++ RUNTIME
.
.
.

******************************** Bottom of Data ******************
```

- Ensure that the Language Environment data sets, SCEERUN and SCEERUN2, are authorized to enable xplink.

  For processes that run the client ORB, since they start the JVM, must run with xplink (on). For best performance, compile applications that use JNI services with xplink enabled. Compiling applications with xplink enabled improves performance in z/OS V1R2. As you move from z/OS V1R2 to z/OS V1R6 you should experience additional performance improvements when all of the LE services calls are xplink enabled.
- Ensure that you are **NOT** using the following options during production:
  - RPTSTG(ON)
  - RPTOPTS(ON)
  - HEAPCHK(ON)
- Turn LE heappools on.

  If you are running a client on z/OS, setting the following: SET LEPARM='HEAPP(ON)' in a shell script, turns on LE heappools, which should improve the performance of the client.
- Refer to "Fine tuning the LE heap"

**Customization Note:** Do not modify LE parameters without consulting IBM support. The LE parameters are set internally to ensure the best possible performance of the WebSphere Application Server, which is the main LE application running in the address space. If you need to add or change LE parameters, make sure that you work with the IBM WebSphere support team to ensure that the internally set parameters are not compromised. The appropriate interface for making these changes is through the `PARM=` parameter of the `EXEC PGM=BBOSR` statement in the startup JCL.

***Fine tuning the LE heap:***

Use these steps to tune your z/OS operating system to optimize WebSphere Application Server performance.

The LE Heap is an area of storage management to be concerned with. For servers, IBM has compiled default values for HEAP and HEAPPOOL into the server main programs. These are good starting points for simple applications. To fine tune the LE Heap settings, use the following procedure:

1. Generate a report on storage utilization for your application servers. Use the LE function RPTSTG(ON) in a SET LEPARM= statement in JCL as shown in the example:.

   ```
   SET LEPARM='RPTOPTS(ON),RPTSTG(ON)'
   ```

   Results appear in servant joblog.
2. To bring the server down cleanly, use the following VARY command:

   ```
   VARY WLM,APPLENV=xxxx,QUIESCE
   ```

   The following example shows the servant SYSPRINT DD output from the `RPTSTG(ON)` option.

   **Example:**

   ```
   .    .    .
   0    HEAP statistics:
          Initial size:                              83886080
   ```

```
        Increment size:                                  5242880
        Total heap storage used (sugg. initial size):  184809328

        Successful Get Heap requests:                     426551
        Successful Free Heap requests:                    424262
        Number of segments allocated:                          1
        Number of segments freed:                              0
    .   .   .

   Suggested Percentages for current Cell Sizes:
     HEAPP(ON,8,6,16,4,80,42,808,45,960,5,2048,20)
   Suggested Cell Sizes:
     HEAPP(ON,32,,80,,192,,520,,1232,,2048,)

  . . .
```

3. Take the heap values from the ″Suggested Cell Sizes″ line in the storage utilization report and use them in another RPTSTG(ON) function to get another report on storage utilization as shown below:

   ```
   SET LEPARM='RPTOPTS(ON),RPTSTG(ON,32,,80,,192,,520,,1232,,2048,)'
   ```

   The following example shows the servant joblog output from the RPTOPTS(ON),RPTSTG(ON,32,,80,,192,,520,,1232,,2048,) option.

   **Example:**

   ```
      .   .
   0    HEAP statistics:
          Initial size:                                   83886080

          Increment size:                                  5242880
          Total heap storage used (sugg. initial size):  195803218

          Successful Get Heap requests:                     426551
          Successful Free Heap requests:                    424262
          Number of segments allocated:                          1
          Number of segments freed:                              0
      .   .   .

     Suggested Percentages for current Cell Sizes:
       HEAPP(ON,32,8,80,43,192,48,520,20,1232,5,2048,20)

     Suggested Cell Sizes:
       HEAPP(ON,32,,80,,192,,520,,1232,,2048,)
    . . .
   ```

4. Take the heap values from the ″Suggested Percentages for current Cell Sizes″ line of the second storage utilization report and use them in another RPTSTG(ON) function to get a third report on storage utilization as shown below:

   ```
   SET LEPARM='RPTOPTS(ON),RPTSTG(ON,32,8,80,43,192,48,520,20,1232,5,2048,20)'
   ```

   The following example shows the servant joblog output from the RPTOPTS(ON),RPTSTG(ON,32,8,80,43,192,48,520,20,1232,5,2048,20) option.

   **Example:**

   ```
      .   .
   0    HEAP statistics:
          Initial size:                                   83886080

          Increment size:                                  5242880
          Total heap storage used (sugg. initial size):  198372130

          Successful Get Heap requests:                     426551
          Successful Free Heap requests:                    424262
          Number of segments allocated:                          1
          Number of segments freed:                              0
      .   .   .
   ```

```
   Suggested Percentages for current Cell Sizes:
     HEAPP(ON,32,8,80,43,192,48,520,20,1232,5,2048,20)
   Suggested Cell Sizes:
     HEAPP(ON,32,,80,,192,,520,,1232,,2048,)
 . . .
```

5. On the third storage utilization report, look for the ″Total heap storage used (sugg. initial size):″ line and use this value for your initial LE heap setting. For example, in the report in third report example this value is 198372130.
6. Make sure that you remove RPTSTG since it does incur a small performance penalty to collect the storage use information.
7. For your client programs that run on z/OS or OS/390, we recommend that you at least specify HEAPP(ON) on the proc of your client to get the default LE heappools. LE will be providing additional pools (more than 6) and larger than 2048MB cell size in future releases of z/OS. You may be able to take advantage of these increased pools and cell sizes, if you have that service on your system.
8. If you use LE HEAPCHECK, make sure to turn it off once you have ensured that your code doesn't include any uninitialized storage. HEAPCHECK can be very expensive.

## UNIX System Services (USS) tuning tips for z/OS

Use these tips to tune your z/OS operating system to optimize WebSphere Application Server performance.

WebSphere Application Server for z/OS no longer requires or recommends the shared file system for the configuration files, since it maintains its own mechanism for managing this data in a cluster. However, WebSphere for z/OS does require the shared files system for XA partner logs. Your application may also use the shared file system. This article provides some basic tuning information for the shared file system.

For basic z/OS UNIX System Services performance information, refer to the following web site: http://www.ibm.com/servers/eserver/zseries/ebusiness/perform.html

- Mount the shared file system R/O.

  Special consideration needs to be made to file system access when you run in a sysplex. If you mount the file system R/W in a shared file system environment, only one system will have local access to the files. All other systems have remote access to the files which negatively affects performance. You may choose to put all of the files for WebSphere in their own mountable file system and mount it R/O to improve performance. However, to change your current application or install new applications, the file system must be mounted R/W. You will need to put operational procedures in place to ensure that the file system is mounted R/W when updating or installing applications.
- HFS files caching.

  HFS Files Caching Read/Write files are cached in kernel data spaces. In order to determine what files would be good candidates for file caching you can use SMF 92 records.

  Initial cache size is defined in BPXPRMxx.
- Consider using zFS.

  z/OS has introduced a new file system called zFS which should provide improved file system access. You may benefit from using the zFS for your UNIX file system. See *z/OS UNIX System Services Planning* for more information.
- Use the filecache command.

  High activity, read-only files can be cached in the USS kernel using the filecache command. Access to files in the filecache can be much more efficient than access to files in the shared file system, even if the shared file system files are cached in dataspaces. GRS latch contention, which sometimes is an issue for frequently accessed files shared file system, will not affect files in the filecache.

  To filecache important files at startup, you can add filecache command to your /etc/rc file. Unfortunately, files which are modified after being added to the filecache may not be eligible for caching until the file system is unmounted and remounted, or until the system is re-IPLed. Refer to *z/OS UNIX System Services Command Reference* for more information about the filecache command.

Example of using the filecache command:

```
/usr/sbin/filecache -a /usr/lpp/WebSphere/V5R0M0/
MQSeries/java/samples/base/de_DE/mqsample.html
```

## Tuning the WebSphere Application Server for z/OS runtime

To optimize performance, review the following steps involved in tuning the WebSphere Application Server for z/OS runtime.

- “Review the WebSphere Application Server for z/OS configuration”
- “Internal tracing tips for WebSphere for z/OS”
- “Location of executable programs tips for z/OS” on page 74
- “Security tuning tips” on page 126

### Review the WebSphere Application Server for z/OS configuration

WebSphere Application Server provides tunable settings for its major components to enable you to make adjustments to better match the runtime environment to the characteristics of your application.

Many applications can run successfully without any changes to the default values for these tuning parameters. Other applications might need changes, for example, a larger heap size, to achieve optimal performance.

Before you read a description of WebSphere Application Server for z/OS tuning guidelines, it is important to note that, no matter how well the middleware is tuned, it cannot make up for poorly designed and coded applications. Focusing on the application code can help improve performance. Often, poorly written or designed application code changes will make the most dramatic improvements to overall performance.

1. Review the WebSphere for z/OS configuration. One simple way to do this is to look in your application control and server regions in SDSF.
2. When each server starts, the runtime prints out the current configuration data in the joblog.

### Internal tracing tips for WebSphere for z/OS

WebSphere Application Server traces can be extremely helpful in detecting and diagnosing problems.

By properly setting trace options, you can capture the information needed to detect problems without significant performance overhead.

- Ensure that you are not collecting more diagnostic data than you need.

  You should check your WebSphere for z/OS tracing options to ensure that ras_trace_defaultTracingLevel=0 or 1, and that ras_trace_basic and ras_trace_detail are not set.

  **How to view or set:** Use the WebSphere administrative console:
  1. Click **Environment > Manage WebSphere Variables**.
  2. On the Configuration Tab check for any of these variables in the name field and observe the variable setting in the value field.
  3. To change or set a variable, specify the variable in the name field and specify the setting in the value field. You can also describe the setting in the description field on this tab.
- For the best performance with any level of tracing, including ras_trace_defaultTracingLevel=1, set ras_trace_outputLocation to BUFFER. This trace setting stores the trace data in memory, which is later asynchronously written to a CTRACE data set. Setting ras_trace_outputLocation to SYSPRINT or TRCFILE provides approximately the same level of performance, but significantly less than BUFFER.
- You can use the ras_trace_BufferCount and ras_trace_BufferSize settings to control the amount of storage used for trace buffers. Generally, the larger the buffer allocation, the better the performance. However, specifying a buffer allocation that is too large can cause a decrease in performance due to system paging.

The default settings of ras_trace_BufferCount=4 and ras_trace_BufferSize=1M should perform sufficiently for most applications.

- Make sure you disable JRAS tracing.

  To do this, look for the following lines in the trace.dat file pointed to by the JVM properties file:

  ```
  com.ibm.ejs.*=all=disable

  com.ibm.ws390.orb=all=disable
  ```

  Ensure that both lines are set to **=disable** or delete the two lines altogether.

  **Note:** If ras_trace_outputLocation is set, you may be tracing and not know it.

## Location of executable programs tips for z/OS

If you choose to not put most of the runtime in LPA, you may find that your processor storage gets a bigger workout as the load increases.

IBM recommends that you install as much of the WebSphere Application Server for z/OS code in LPA as is reasonable, and the remainder in the linklist. Also, ensure that you have eliminated any unnecessary STEPLIBs which can affect performance. If you must use STEPLIBs, verify that any `STEPLIB DD`s in the controller and servant procs do not point to any unnecessary libraries. Refer to UNIX System Services (USS) tuning tips for z/OS for USS shared file system tuning considerations.

At a minimum, WebSphere for z/OS will start three address spaces, so that any code that is not shared will load three copies rather than one. As the load increases, many more servants may start and will contribute additional load on processor storage.

Review the `PATH` statement to ensure that only required programs are in the PATH and that the order of the PATH places frequently-referenced programs in the front.

## Tuning storage

WebSphere Application Server for z/OS puts much higher demands on virtual memory than a traditional workload. Ensure that you don't underestimate the amount of virtual storage applied to the WebSphere Application Server for z/OS servers.

Generally, Websphere Application Server for z/OS uses significantly more virtual memory than traditional application servers on z/OS. Since real storage is needed to back the virtual storage, real storage usage may also be high. Ensure that you don't underestimate the amount of D applied to the WebSphere Application Server for z/OS servers.

1. Allocate enough virtual storage. The setting of REGION on the JCL for the proc controls the amount of virtual storage available to a z/OS address space. The default values for the WebSphere Application Server controller and servant are set to zero, which tells the operating system to allocate all the available region (close to 2GB). You can limit the amount of virtual storage allocated by setting the REGION parameter to a value other than zero. The size of the JVM heap is the most important factor when determining the setting of the REGION parameter. You should only need to set the REGION to something other than zero when the JVM heap size is very large. The z/OS operating system allocates user storage from the bottom of the address space, which is where the JVM heap is allocated, and system storage from the top. System abends can occur when the system tries to obtain virtual storage and none is available. A non-zero REGION parameter setting prevents this from occurring by preserving storage at the top of the address space for system use. In almost all cases running with the default REGION will be satisfactory.

   **Note:** See the *Installing your application serving environment* PDF for more information.

2. **Optional:** Allocate enough real storage. Expect a requirement of at least 512MB of real storage for a small configuration. The total amount of real storage used by a WebSphere system is dependent on the number of servers and the size of the JVM heaps for each server.

   **Recommendation:** It can be the case that in a heavy use environment 2GB of central storage is not enough to handle the real storage demands of a high volume Java application. In this case, we recommend that you configure with 64-bit real storage, which will give you the ability to dedicate more central storage to the LPAR. z/OS on a zSeries will always run in 64-bit mode. Running in 64-bit mode gives you the ability to define more than 2GB of central storage When you configure for 64-bit real all of the storage is defined as central storage. For non-zSeries processors, or 31-bit mode, you can minimize paging by defining more expanded storage.
3. "Java virtual machine storage tuning tips for z/OS." Also refer to Best practices for maintaining the run-time environment.

## Java virtual machine storage tuning tips for z/OS

Specifying a sufficient `JVM Heap Size` is important to Java performance.

The JVM has thresholds it uses to manage the JVM's storage. When the thresholds are reached, the garbage collector (GC) gets invoked to free up unused storage. GC can cause significant degradation of Java performance.

Use the administrative console to specify the Initial Heap Size and the Maximum Heap Size for the JVM.

To view this administrative console page, click **Servers > Application Servers >** *server_name* **> Process Definition > Java Virtual Machine**. Access the configuration tab to change these settings.
- In the majority of cases you should set the maximum JVM heap size to value higher than the initial JVM heap size. This allows for the JVM to operate efficiently during normal, steady state periods within the confines of the initial heap but also to operate effectively during periods of high transaction volume by expanding the heap up to the maximum JVM heap size. In some rare cases where absolute optimal performance is required you might want to specify the same value for both the initial and maximum heap size. This will eliminate some overhead that occurs when the JVM needs to expand or contract the size of the JVM heap.Make sure the region is large enough to hold the specified JVM heap.
- Beware of making the Initial Heap Size *too* large. While it initially improves performance by delaying garbage collection, it ultimately affects response time when garbage collection eventually kicks in (because it runs for a longer time).
- Paging activity on your system must also be considered when you set your JVM heap size. If your system is already paging heavily, increasing the JVM heap size might make performance worse rather than better.
- For WebSphere Application Server for zOS the default JVM heap size settings are 128m/256m (initial/maximum) for the controller and 256m/512m for the servant. For the majority of applications these defaults should work quite well. You can monitor the frequency of garbage collection by enabling verbose GC (see below). If garbage collection is occurring too frequently you should increase the size of the JVM heap.
- To determine if you are being affected by garbage collection, you can enable Verbose Garbage Collection on the JVM Configuration tab. The default is not enabled. This will write a report to the output stream each time the garbage collector runs. This report should give you an idea of what is going on with Java GC.

  **Example:** This is an example of a verboseGC report.

```
<af type="tenured" id="7" timestamp="Mon Nov 14 22:58:18 2005" intervalms="3724.328">
  <minimum requested_bytes="72" />
  <time exclusiveaccessms="0.034" />
  <tenured freebytes="4026368" totalbytes="134217728" percent="2" >
    <soa freebytes="0" totalbytes="130191360" percent="0" />
    <loa freebytes="4026368" totalbytes="4026368" percent="100" />
  </tenured>
  <gc type="global" id="7" totalid="7" intervalms="3724.645">
    <refs_cleared soft="0" weak="10" phantom="29578" />
    <finalization objectsqueued="0" />
    <timesms mark="76.550" sweep="6.021" compact="0.000" total="82.939" />
    <tenured freebytes="87995424" totalbytes="134217728" percent="65" >
      <soa freebytes="85311520" totalbytes="131533824" percent="64" />
      <loa freebytes="2683904" totalbytes="2683904" percent="100" />
    </tenured>
  </gc>
  <tenured freebytes="87994768" totalbytes="134217728" percent="65" >
    <soa freebytes="85310864" totalbytes="131533824" percent="64" />
    <loa freebytes="2683904" totalbytes="2683904" percent="100" />
  </tenured>
  <time totalms="83.290" />
</af>
```

Key things to look for in a verboseGC report are:

– Time spent in garbage collection.

  Ideally, you want to be spending less than 5% of the time in GC. To determine percentage of time spent in GC, divide the time it took to complete the collection by the time since the last AF and multiply the result by 100. For example,

  ```
  83.29/3724.32 * 100 = 2.236%
  ```

  If you are spending more than 5% of your time in GC and if GC is occurring frequently, you may need to increase your Java heap size.

– Growth in the allocated heap.

  To determine this, look at the %free. You want to make sure the number is not continuing to decline. If the %free continues to decline you are experiencing a gradual growth in allocated heap from GC to GC which could indicate that your application has a memory leak.

You can also use the MVS console command, *modify display, jvmheap* to display JVM heap information. See ″Modify command″ for details. In addition, you can check the server activity and interval SMF records. The JVM heap size is also made available to PMI and can be monitored using the Tivoli Performance Viewer.

## Workload management (WLM) tuning tips for z/OS

If you are running z/OS 1.2 or higher, you can use the administrative console to provide the job control language (JCL) PROC name for the servant and the JCL Parm for the servant and thereby set up a dynamic application environment. Even if you set up a dynamic application environment, you must set the WLM goals for your environment.

Proper WLM goals can significantly affect your application throughput. The WebSphere Application Server address spaces should be given a fairly high priority. When setting the WLM goals for your z/OS system, you might want to:

- Classify location service daemons and controllers as SYSSTC or high velocity.
- Use STC classification rules to classify velocity goals for application servers.

  Java garbage collection runs under this classification. Java garbage collection is a CPU and storage intensive process. If you set the velocity goal too high garbage collection can consume more of your system resources than desired. If your Java heap is correctly tuned, garbage collection for each servant should run no more than 5% of the time. Also, providing proper priority to garbage collection processing is necessary since other work in the servant is stopped during much of the time that garbage collection is running.

  JavaServer Page file compiles run under this classification. If your system is configured to do these compiles at runtime, setting the velocity goal too low can cause longer delays waiting for JavaServer Page file compiles to complete.

  Application work is classified under the work manager.
- Set up an application environment for work running under servants where:
  – The subsystem type is set to CB.

- The environment is classified based on server name, server instance name, user ID, and transaction class.
- Percentage response time goals are set.

  You should make the response time goals achievable. For example, a goal that 80% of the work will complete in .25 seconds is a typical goal. Velocity goals for application work are not meaningful and should be avoided.
- A high velocity default service class for CB subsystem transactions is provided. The default is SYSOTHER.

  Your goals can be multi-period. This might be useful if you have distinctly short and long running transactions in the same service class. On the other hand, it is usually better to filter this work into a different service class if you can. Being in a different service class will place the work in a different servant which allows WLM much more latitude in managing the goals.

• Define unique WLM report classes for servant regions and for applications running in your application environment. Defining unique WLM report classes enables the resource measurement facility (RMF) to report performance information with more granularity.

• Set your Application environment to **No Limit**.
  - Required if you need more than one servant per application server.
  - Under WLM, you can control how many servants can be started for each server. If you need more than one servant in a server make sure that **No Limit** is selected for the application environment associated with your server. For information about setting up WLM performance goals, see *z/OS MVS Planning: Workload Management*.

    **Example:**

```
Application-Environment  Notes  Options  Help
----------------------------------------------------------------------------
                    Modify an Application Environment
Command ===> ______________________________________________________________

Application Environment Name . : BBOASR2
Description  . . . . . . . . . . WAS.V40.WB02 Application server
Subsystem Type . . . . . . . . . CB
Procedure Name . . . . . . . . . BBOASR2S
Start Parameters . . . . . . . . IWMSSNM=&IWMSSNM
                                 __________________________________________
                                 __________________________________________

Limit on starting server address spaces for a subsystem instance:
1   1.  No limit

    2.  Single address space per system
    3.  Single address space per sysplex
```

    When the WLM configuration is set to no limit, you can use the `wlm_maximumSRCount=x` and `wlm_minimumSRCount=y` variables to control the maximum and minimum number of servants. To specify values for these variables, in the administrative console, click **Severs** > **Application servers** and select the appropriate application server.

    **Important:** If you specify a value for the wlm_maximumSRCount variable, the value must be greater than or equal to the number of service classes defined for this application environment. If the value is less than the number of defined service classes, timeouts might be caused because there is an insufficient number of servants available.
  - Results reported in RMF Postprocessor workload activity report:
    - Transactions per second (not always the same as client tran rate)
    - Average response times (and distribution of response times)
    - CPU time used
    - Percent response time associated with various delays

## Tuning WebSphere applications

This topic provides quick links to information about tuning specific WebSphere application types, and the services and containers that support them.

**Note:** The WebSphere Application Server documentation contains a finite set of tuning topics to which the following table provides links. Installing the documentation plug-ins for additional components, such as Service integration, might add new entries to the information table of contents. The new entries will not be shown in the table. To see the complete set of application tuning topics available in this information center installation, expand **Tuning performance > Tuning WebSphere applications** in the table of contents.

### Product architecture and programming model, at a glance

| **Application serving environment -- See Tuning the application serving environment** | **WebSphere applications** | **WebSphere applications** |
|---|---|---|
| **Servers**<br>• Application servers<br>• Java virtual machines<br>• Java virtual machines<br>• Transport channels<br>• Web servers<br>• More server types<br>• Core groups<br>• Workload balancing<br>**Environment**<br>• Hardware<br>• Operating system<br>• Virtual hosts<br>• Variable settings<br>• Shared libraries<br>• Replication domains<br>**System administration**<br>• Administrative clients<br>• Configuration files<br>• Domains (cells, nodes)<br>**Performance tools**<br>• Monitoring<br>• Tuning performance<br>**Troubleshooting tools**<br>• Diagnostic tools<br>• Support and self-help<br>The product subsystems are discussed in the Product architecture. For the most part, they do not depend on the type of applications being deployed | **Services**<br>• Security<br>• Naming<br>• ORB<br>• Transactions<br>**J2EE applications**<br>• Web applications > Sessions<br>• EJB applications<br>**Clients**<br>• Client applications<br>• Web clients<br>• Web services clients<br>• Administrative clients<br>**Web services**<br>• Web services and Service Oriented Architecture (SOA)<br>• Web services security | **J2EE resources**<br>• Data access resources<br>• Messaging resources<br>• Mail, URLs, and more<br>**WebSphere extensions**<br>• ActivitySessions<br>• Application profiling<br>• Asynchronous beans<br>• Dynamic caching<br>• Dynamic and EJB query<br>• Internationalization<br>• Object pools<br>• Scheduler<br>• Startup beans<br>• Work area |

**Topology planning and performance** Topology can have a significant effect on WebSphere performance. This article describes some of the topology considerations you should be aware of when configuring and installing WebSphere Application Server for z/OS.

- Single server or multiple servers?

  WebSphere for WebSphere Application Server for z/OS gives you the ability to install your application either in a single server or spread it across multiple servers. There are many reasons for partitioning your application. However, for performance, placing your application all in the same server will always provide better performance than partitioning it. If you do choose to partition your application across servers, you will get better performance if there are at least replica servers on each system in the sysplex. The WebSphere for WebSphere Application Server for z/OS runtime will try to keep calls local to the system if it can, which will, for example, use local interprocess calls rather than sockets.
- One tran or multiple trans?

  You also have a choice of running server regions with an isolation policy of one tran per server region or multiple trans per server region. From a performance perspective, running more threads in a server region will consume less memory but at the cost of thread contention. This contention is application-dependent. We generally recommend the use of multiple trans unless you run into contention problems.

  Specify the threads setting using the *server_region_workload_profile*. The variables include:
  – ISOLATE - sets the value to 1 thread.
  – CPUBOUND
  – IOBOUND - default
  – LONGWAIT - 40

  The thread value increases with each variable to the maximum number available with the LONGWAIT setting (40). For more information refer to ORB services advanced settings on the z/OS platform

  **Note:** Please see the ″Servers″ section in the WebSphere Application Server for z/OS information center, access to which can be obtained through the WebSphere Application Server library Web site (http://www-306.ibm.com/software/webservers/appserv/was/library/index.html) for more information on ORB services advanced settings.
- Local client or remote client?

  On a local client, the client and the optimized communication are done on the same system. This has some additional client CPU costs but less communication cost. On a remote client, the client cost is replaced by the additional communication overhead of sockets. The CPU cost on either system is almost equivalent. Latency is better for a local client than for a remote client, meaning you will get better response time with a local client.
- One copy of a server or many clones?

  You can define more than one copy of a server on a system. These copies are called clones. We have found slight improvements in performance when running with a couple of clones as opposed to just one (very large configuration). While there is some benefit, IBM does not recommend, at this time, the creation of replicated control regions for the sole purpose of improving performance. We do, however, recommend them for eliminating a single point of failure and for handling rolling upgrades without introducing an outage.

## Web applications

### Session management tuning

WebSphere Application Server session support has features for tuning session performance and operating characteristics, particularly when sessions are configured in a distributed environment. These options support the administrator flexibility in determining the performance and failover characteristics for their environment.

The table summarizes the features, including whether they apply to sessions tracked in memory, in a database, with memory-to-memory replication, or all. Click a feature for details about the feature. Some features are easily manipulated using administrative settings; others require code or database changes.

| **Feature or option** | **Goal** | **Applies to sessions in memory, database, or memory-to-memory** |
|---|---|---|
| Write frequency | Minimize database write operations. | Database and Memory-to-Memory |
| Session affinity | Access the session in the same application server instance. | All |
| Multirow schema | Fully utilize database capacities. | Database |
| Base in-memory session pool size | Fully utilize system capacity without overburdening system. | All |
| Write contents | Allow flexibility in determining what session data to write | Database and Memory-to-Memory |
| Scheduled invalidation | Minimize contention between session requests and invalidation of sessions by the Session Management facility. Minimize write operations to database for updates to last access time only. | Database and Memory-to-Memory |
| Tablespace and row size | Increase efficiency of write operations to database. | Database (DB2 only) |

***Scheduled invalidation:***

Instead of relying on the periodic invalidation timer that runs on an interval based on the session timeout parameter, you can set specific times for the session management facility to scan for invalidated sessions in a distributed environment. When used with distributed sessions, this feature has the following benefits:
- You can schedule the scan for invalidated sessions for times of low application server activity, avoiding contention between invalidation scans of database or another WebSphere Application Server instance and read and write operations to service HTTP session requests.
- Significantly fewer external write operations can occur when running with the End of Service Method Write mode because the last access time of the session does not need to be written out on each HTTP request. (Manual Update options and Time Based Write options already minimize the writing of the last access time.)

**Usage considerations**
- The session manager invalidates sessions only at the scheduled time, therefore sessions are available to an application if they are requested before the session is invalidated.
- With scheduled invalidation configured, HttpSession timeouts are not strictly enforced. Instead, all invalidation processing is handled at the configured invalidation times.
- HttpSessionBindingListener processing is handled at the configured invalidation times unless the HttpSession.invalidate method is explicitly called.
- The HttpSession.invalidate method immediately invalidates the session from both the session cache and the external store.
- The periodic invalidation thread still runs with scheduled invalidation. If the current hour of the day does not match one of the configured hours, sessions that have exceeded the invalidation interval are removed from cache, but not from the external store. Another request for that session results in returning that session back into the cache.
- When the periodic invalidation thread runs during one of the configured hours, all sessions that have exceeded the invalidation interval are invalidated by removal from both the cache and the external store.
- The periodic invalidation thread can run more than once during an hour and does not necessarily run exactly at the top of the hour.
- If you specify the interval for the periodic invalidation thread using the HttpSessionReaperPollInterval custom property, do not specify a value of more than 3600 seconds (1 hour) to ensure that invalidation processing happens at least once during each hour.

***Configuring write contents:***

In session management, you can configure which session data is written to the database or to another WebSphere instance, depending on whether you are using database persistent sessions or memory to memory replication. This flexibility allows for fewer code changes for the JavaServer Pages (JSP) writer when the application will be operating in a clustered environment. The following options are available in Session Management for tuning what is to be written back:

- Write changed (the default) - Write only session data properties that have been updated through setAttribute method and removeAttribute method calls.
- Write all - Write all session data properties.

The **Write all** setting might benefit servlet and JSP writers who change Java objects' states that reside as attributes in HttpSession and do not call HttpSession.setAttribute method.

However, the use of **Write all** could result in more data being written back than is necessary. If this situation applies to you, consider combining the use of **Write all** with **Time-based write** to boost performance overall. As always, be sure to evaluate the advantages and disadvantages for your installation.

With either Write Contents setting, when a session is first created, complete session information is written, including all of the objects bound to the session. When using database session persistence, in subsequent session requests, what is written to the database depends on whether a single-row or multi-row schema has been set for the session database, as follows:

| Write Contents setting | Behavior with single-row schema | Behavior with multirow schema |
|---|---|---|
| Write changed | If any session attribute is updated, all objects bound to the session are written. | Only the session data modified through setAttribute method or removeAttribute method calls is written. |
| Write all | All bound session attributes are written. | All session attributes that currently reside in the cache are written. If the session has never left the cache, all session attributes are written. |

1. Go to the appropriate level of Session Management. See the *Administering applications and their environment* PDF for more information.
2. Click Distributed Environment Settings
3. Click Custom Tuning Parameters.
4. Select Custom Settings, and click Modify.
5. Select the appropriate write contents setting.

***Configuring write frequency:***

In the Session Management facility, you can configure the frequency for writing session data to the database or to a WebSphere instance, depending on whether you use database distributed sessions or memory-to-memory replication. This flexibility enables you to weigh session performance gains against varying degrees of failover support. The following options are available in the Session Management facility for tuning write frequency:

- **End of service servlet**- Write session data at the end of the servlet service method call.
- **Manual update**- Write session data only when the servlet calls the IBMSession.sync method.
- **Time based** (the default) - Write session data at periodic intervals, in seconds (called the *write interval*).

When a session is first created, session information is always written at the end of the service call.

Using the time based write or manual update options can result in loss of data in failover scenarios since the backup copy of the session in the persistent store (for example, a database or another JVM) may not be in sync with the session in the session cache.

***Base in-memory session pool size:*** The base in-memory session pool size number has different meanings, depending on session support configuration:
- With in-memory sessions, session access is optimized for up to this number of sessions.
- With distributed sessions (meaning, when sessions are stored in a database or in another WebSphere Application Server instance); it also specifies the cache size and the number of last access time updates saved in manual update mode.

For distributed sessions, when the session cache has reached its maximum size and a new session is requested, the Session Management facility removes the least recently used session from the cache to make room for the new one.

General memory requirements for the hardware system, and the usage characteristics of the e-business site, determines the optimum value.

Note that increasing the base in-memory session pool size can necessitate increasing the heap sizes of the Java processes for the corresponding WebSphere Application Servers.

**Overflow in non-distributed sessions**

By default, the number of sessions maintained in memory is specified by base in-memory session pool size. If you do not wish to place a limit on the number of sessions maintained in memory and allow overflow, set overflow to *true*.

Allowing an unlimited amount of sessions can potentially exhaust system memory and even allow for system sabotage. Someone could write a malicious program that continually hits your site and creates sessions, but ignores any cookies or encoded URLs and never utilizes the same session from one HTTP request to the next.

When overflow is disallowed, the Session Management facility still returns a session with the HttpServletRequest getSession(true) method when the memory limit is reached, and this is an invalid session that is not saved.

With the WebSphere Application Server extension to HttpSession, com.ibm.websphere.servlet.session.IBMSession, an isOverflow method returns *true* if the session is such an invalid session. An application can check this status and react accordingly.

***Write operations:***

You can manually control when modified session data is written out to the database or to another WebSphere Application Server instance by using the sync method in the com.ibm.websphere.servlet.session.IBMSession interface. This interface extends the javax.servlet.http.HttpSession interface. By calling the sync method from the service method of a servlet, you send any changes in the session to the external location. When manual update is selected as the write frequency mode, session data changes are written to an external location only if the application calls the sync method. If the sync method is not called, session data changes are lost when a session object leaves the server cache. When end of service servlet or time based is the write frequency mode, the session data changes are written out whenever the sync method is called. If the sync method is not called, changes are written out at the end of service method or on a time interval basis based on the write frequency mode that is selected.

```
IBMSession iSession = (IBMSession) request.getSession();
iSession.setAttribute("name", "Bob");

//force write to external store
iSession.sync( )
```

If the database is down or is having difficulty connecting during an update to session values, the sync method always makes three attempts before it finally creates a BackedHashtable.getConnectionError error. For each connection attempt that fails, the BackedHashtable.StaleConnectionException is created and can be found in the sync method. If the database opens during any of these three attempts, the session data in the memory is then persisted and committed to the database.

However, if the database is still not up after the three attempts, then the session data in the memory is persisted only after the next check for session invalidation. Session invalidation is checked by a separate thread that is triggered every five minutes. The data in memory is consistent unless a request for session data is issued to the server between these events. For example, if the request for session data is issued within five minutes, then the previous persisted session data is sent.

Sessions are not transactional resources. Because the sync method is associated with a separate thread than the client, the exception that is created does not propagate to the client, which is running on the primary thread. Transactional integrity of data can be maintained through resources such as enterprise beans.

***Tuning parameter settings:***

Use this page to set tuning parameters for distributed sessions.

To view this administrative console page, click **Servers > Application servers >** *server_name* **> Web container settings > Session management > Distributed environment settings > Custom tuning parameters**.

*Tuning level:*

Specifies that the session management facility provides certain predefined settings that affect performance.

Select one of these predefined settings or customize a setting. To customize a setting, select one of the predefined settings that comes closest to the setting desired, click **Custom settings**, make your changes, and then click **OK**.

**Very high (optimize for performance)**

| | |
|---|---|
| **Write frequency** | Time based |
| **Write interval** | 300 seconds |
| **Write contents** | Only updated attributes |
| **Schedule sessions cleanup** | true |
| **First time of day default** | 0 |
| **Second time of day default** | 2 |

**High**

| | |
|---|---|
| **Write frequency** | Time based |
| **Write interval** | 300 seconds |
| **Write contents** | All session attributes |
| **Schedule sessions cleanup** | false |

**Medium**

| | |
|---|---|
| **Write frequency** | End of servlet service |
| **Write contents** | Only updated attributes |
| **Schedule sessions cleanup** | false |

**Low (optimize for failover)**

| | |
|---|---|
| **Write frequency** | End of servlet service |
| **Write contents** | All session attributes |
| **Schedule sessions cleanup** | false |

**Custom settings**

| | |
|---|---|
| **Write frequency default** | Time based |
| **Write interval default** | 10 seconds |
| **Write contents default** | All session attributes |
| **Schedule sessions cleanup default** | false |

***Tuning parameter custom settings:***

Use this page to customize tuning parameters for distributed sessions.

To view this administrative console page, click **Servers** > **Application servers** > *server_name* **Web container settings** > **Session management** > **Distributed environment settings** > **Custom tuning parameters** > **Custom settings**.

*Write frequency:*

Specifies when the session is written to the persistent store.

| | |
|---|---|
| **End of servlet service** | A session writes to a database or another WebSphere Application Server instance after the servlet completes execution. |
| **Manual update** | A programmatic sync on the IBMSession object is required to write the session data to the database or another WebSphere Application Server instance. |
| **Time based** | Session data writes to the database or another WebSphere Application Server instance based on the specified Write interval value. Default: 10 seconds |

*Write contents:*

Specifies whether updated attributes are only written to the external location or all of the session attributes are written to the external location, regardless of whether or not they changed. The external location can be either a database or another application server instance.

| | |
|---|---|
| **Only updated attributes** | Only updated attributes are written to the persistent store. |
| **All session attribute** | All attributes are written to the persistent store. |

*Schedule sessions cleanup:*

Specifies when to clean the invalid sessions from a database or another application server instance.

| | |
|---|---|
| **Specify distributed sessions cleanup schedule** | Enables the scheduled invalidation process for cleaning up the invalidated HTTP sessions from the external location. Enable this option to reduce the number of updates to a database or another application server instance required to keep the HTTP sessions alive. When this option is not enabled, the invalidator process runs every few minutes to remove invalidated HTTP sessions. When this option is enabled, specify the two hours of a day for the process to clean up the invalidated sessions in the external location. Specify the times when there is the least activity in the application servers. An external location can be either a database or another application server instance. |
| **First Time of Day (0 - 23)** | Indicates the first hour during which the invalidated sessions are cleared from the external location. Specify this value as a positive integer between 0 and 23. This value is valid only when schedule invalidation is enabled. |
| **Second Time of Day (0 - 23)** | Indicates the second hour during which the invalidated sessions are cleared from the external location. Specify this value as a positive integer between 0 and 23. This value is valid only when schedule invalidation is enabled. |

## EJB applications

### EJB Container tuning

If you use applications that affect the size of the EJB Container Cache, it is possible that the performance of your applications can be impacted by an incorrect size setting. Monitoring Tivoli Performance Viewer (TPV) is a great way to diagnose if the EJB Container Cache size setting is tuned correctly for your application.

If the application has filled the cache causing evictions to occur, TPV will show a very high rate of ejbStores() being called and probably a lower than expected CPU utilization on the application server machine.

All applications using enterprise beans should have this setting adjusted from the default if the following formula works out to more than 2000.

```
EJB_Cache_Size = (Largest number of Option B or C Entity Beans enlisted in a
transaction * maximum number of concurrent transactions) +
(Largest number of unique Option A Entity Beans expected to be accessed during
typical application workload) +
(Number of stateful Session Beans active during typical workload) +
(Number of stateless SessionBean types used during typical workload)

Where:
Option B and C Entity Beans are only held in the EJB cache during the lifetime
of the transaction they are enlisted in. Therefore, the first term in the formula
computes the average EJB cache requirements for these types of beans.

Option A Entity Beans are held in the EJB cache indefinitely, and are only removed
 from the cache if there start to become more beans in the cache than the cache
size has been set to.

Stateful Session Beans are held in the EJB cache until they are removed by the
application, or their session timeout value is reached.

Only a single stateless Session Bean instance for each EJB type is held in the
cache during the time any methods are being executed on that stateless Session
```

```
Bean. If two or more methods are being executed simultaneously on the same
stateless Session Bean type, each method executes on its own bean instance, but
only one cache location is used for all of these instances.
```

This calculates the upper bound on the maximum possible number of enterprise beans active at one time inside the application server. Because the EJB Containers cache is built to contain all these beans for performance optimizations, best performance can be achieved by setting this cache size to be larger than the number resulting from the calculation above.

```
<tuning parameter>
This setting can be found under Servers > Application Servers > serverName >
         EJB Container > EJB Cache Settings
```

Also while adjusting the EJB Cache Size, the EJB Container management thread parameter can be tuned to meet the needs of the application. The management thread is controlled through the Clean Up Interval setting. This setting controls how frequently a daemon thread inside of WebSphere Application Server wakes up and attempts to remove bean instances from the cache that have not been used recently, attempting to keep the number of bean instances at or below the cache size. This allows the EJB container to place and look up items in the cache as quickly as possible. It normally is best to leave this interval set to the default, however, in some cases, it may be worthwhile to see if there is a benefit to reducing this interval.

### EJB Container Pool Size

If the application is using the majority of the instances in the pool, TPV indicates this. When this occurs, then the size of those bean pools that are being exhausted should be increased. This can be done by adding the following parameter in the JVM's custom properties tag .

```
-Dcom.ibm.websphere.ejbcontainer.poolSize=<application_name>#<module_name>#
<enterprisebean_name>=<minSize>,<maxSize>

where:
 <application_name> is the J2EE application name as defined in the application
archive (.ear) file deployment descriptor, for the bean whose pool size is being
set

<module_name> is the .jar file name of the EJB module, for the bean whose pool size
 is being set,

<bean_name> is the J2EE Enterprise Bean name as defined in the EJB module
deployment descriptor, for the bean whose pool size is being set

<minSize> is the number of bean instances the container maintains in the pool,
 irrespective of how long the beans have been in the pool (beans greater than this
 number are cleared from the pool over time to optimize memory usage)

<maxSize> is the number of bean instances in the pool where no more bean instances
 are placed in the pool after they are used (that is, once the pool is at this
 size, any additional beans are discarded rather than added into the pool --
this ensures the number of beans in the pool has an upper limit so memory usage
does not grow in an unbounded fashion).

To keep the number of instances in the pool at a fixed size, minSize and maxSize
can be set to the same number. Note that there is a separate instance pool for
every EJB type running in the application server, and that every pool starts out
with no instances in it - that is, the number of instances grows as beans are
used and then placed in the pool. When a bean instance is needed by the container
and no beans are available in the pool, the container creates a new bean
instance, uses it, then places that instance in the pool (unless there are
already maxSize instances in the pool).

For example, the statement
-Dcom.ibm.websphere.ejbcontainer.poolSize=ivtApp#ivtEJB.jar#ivtEJBObject=125,1327
```

```
would set a minSize of 125 and a maxSize of 1327 on the bean named "ivtEJBObject"
within the ivtEJB.jar file, in the application "ivtApp".
```

Where ivtApp is replaced by the actual application name, ivtEJB.jar is replaced by the jar containing the bean that needs to have its pool size increased, and ivtEJBObject is the bean name of the enterprise bean whose pool size should be increased. The 125,1327 is the minimum and maximum number of beans that will be held in the pool. These should be set so no more evictions occur from the pool and in most cases should be set equal if memory is plentiful because no growth and shrinkage of the pool will occur.

## EJB Container Primary Key Mutation

Application developers and administrators should have a good idea of how their application handles the creation of primary key objects for use by container-managed persistence (CMP) beans and bean-managed persistence (BMP) beans inside of WebSphere Application Server. The IBM EJB Container uses the primary key of an Entity bean as an identifier inside of many internal data structures to optimize performance. However, the EJB Container must copy these primary key objects upon the first access to the bean to ensure that the objects stored in the internal caches are separate from the ones used in an application, in case the application changes or mutates the primary key, to keep the internal structures consistent.

If the application does not mutate any of the primary keys used to create and access entity beans after they are created, then a special flag can be used that allows the EJB Container to skip the copy of the primary key object, thus saving CPU cycles and increasing performance. This mechanism can be enabled *at your own risk* by adding the following –D property to the JVM custom property field.

```
<tuning parameter>
-Dcom.ibm.websphere.ejbcontainer.noPrimaryKeyMutation=true
```

The performance benefit of this optimization depends on the application. If the application uses primitive types for enterprise beans’ primary keys there will be no gain because these objects are already immutable and the copy mechanism takes this into account. If, however, the application uses many complex primary keys (that is, And object for a primary key or multiple fields) then this parameter can yield significant improvements.

## Persistence Manager Deferred Insert on EJB Create

The IBM Persistence manager is used by the EJB Container to persist data to the database from CMP entity beans. When creating entity beans by calling the ejbCreate() method, by default the Persistence manager immediately inserts the empty row with only the primary key in the database. In most cases applications, after creating the bean, modify fields in the bean created or in other beans inside of the same transaction. If the user wishes to postpone the insert into the database until the end of the transaction, so that it will eliminate one trip to the database, they may set this –D flag inside of the JVM custom properties field. The data will still be inserted into the database and consistency will be maintained.

```
<tuning parameter>
-Dcom.ibm.ws.pm.deferredcreate=true
```

The performance benefit of this optimization depends on the application. If the EJB applications transactions are very insert intensive the application could benefit largely from this optimization. If the application performs very few inserts then the benefit of this optimization will be much less.

## Persistence Manager Database Batch Update on EJB Update

When an EJB application accesses multiple CMP beans inside of a single transaction, depending on the operations performed on the beans (updates, inserts, reads), the number of operations issued to the database will correspond directly to the operations performed on the CMP beans. If the database system you are using supports batching of update statements you can enable this flag and gain a performance boost on all interactions with the database that involve more than two updates in a single transaction. This flag will let the persistence manager add all the update statements into one single batch statement which will then be issued to the database. This saves round trips to the database, thus increasing performance. If the user knows their application exhibits the behavior of updating multiple CMP beans in a single transaction and the database supports batch updates they may set this –D flag inside of the JVM custom properties field.

```
<tuning parameter>
-Dcom.ibm.ws.pm.batch=true
```

The performance benefit of this optimization depends on the application. If the application never or infrequently updates CMP beans or only updates a single bean per transaction there will be no performance gain. If the application updates multiple beans per transaction then this parameter will benefit your applications performance.

The following table lists which backend databases support batch update.

*Table 2.*

| **Database** | **Supports Batch update** | **Supports Batch update with Optimistic Concurrency Control** |
|---|---|---|
| DB2 | yes | no |
| Oracle | yes | no |
| DB2 Universal Driver | yes | yes |
| Informix | yes | yes |
| SQLServer | yes | yes |
| Cloudscape | yes | yes |

**Note:** Batch update with OCC cannot be performed for databases that do not support it, even if specified by the access intent.

### Persistence Manager cache Tuning

Persistence Manager has two different types of caching mechanisms available: *legacy cache* and *two-level cache*. Normally two-level cache performs better than legacy cache because of optimizations in this mode. The default is legacy cache, although two-level cache is recommended. Set this configuration through the system property

```
com.ibm.ws.pm.useLegacyCache=false
```

### Persistence Manager Partial Updates Tuning

The partial updates feature enhances the performance of applications with enterprise beans in certain scenarios. Persistence Manager has two different types of caching mechanisms available, legacy cache and two-level cache. Normally, two-level cache performs better than legacy cache because of the optimizations in this mode. In certain applications where you need to perform both batch updates and partial updates, you must configure the following system properties to gain the benefits of both.

```
'com.ibm.ws.pm.grouppartialupdate=true' and 'com.ibm.ws.pm.batch=true'
```

## Setting tuning properties of a messaging engine

Use this task to set the tuning properties for a messaging engine.

You can set the following property to improve the performance of a messaging engine:

| Name | Value |
|---|---|
| sib.trm.retry | The messaging engine to messaging engine connection retry interval in seconds. The retry interval is the time delay left between attempts to contact neighboring messaging engines with which communications contact should exist. The default retry interval is 30 seconds. |

To set the tuning properties for a messaging engine, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses** → **[Content Pane]** *bus_name* → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] Custom properties**.
2. Type the name of the property that you want to set.
3. Type the value that you want to set for that property.
4. Click **OK**.
5. Save your changes to the master configuration.
6. Restart the messaging engine for the changes to take effect.

## Messaging engine failover between v6 and v6.1

It is not permissible to failover a messaging engine using a file store onto a WebSphere Application Server v6 server. If you have a cluster as a bus member that consists of a mixture of v6 and v6.1 servers, you must modify the high availability policy to prevent this.

To prevent failover of a v6.1 messaging engine to a v6 server, the high availability policy for the messaging engine should be modified so that the cluster is effectively divided into sets of servers at the different versions and the messaging engine is restricted to the servers at v6.1.

## Tuning and problem solving for messaging engine data stores

Obtain an overview of improving the performance of messaging engine data stores and understanding problems that can occur with a data store.

For more information about tuning and problem solving for messaging engine data stores, see the following topics:
- “Tuning the JDBC data source of a messaging engine”
- “Controlling the memory buffers used by a messaging engine” on page 90
- Diagnosing problems with data store exclusive access locks
- “Diagnosing problems with your data store configuration” on page 92
- “Avoiding failover problems when you use DB2 v8.2 with HADR as your data store” on page 92

### Tuning the JDBC data source of a messaging engine

The messaging engine needs to have the correct configuration for JDBC data source to achieve messaging performance on a service integration bus.

Consider whether you need to configure the connection pool for the JDBC data source to achieve your requirements for messaging performance.

The messaging engine uses the connection pool to obtain its connections to the database. With a heavy workload, a messaging engine might require a large number of concurrent connections to avoid delays waiting for connections to become available in the pool. For example, a very heavily loaded messaging engine might need 50 or more connections. Perform the following steps to configure the connection pool to meet your performance requirements:

1. Ensure that the configuration of your relational database management system (RDBMS) permits the number of connections that you require. Refer to the documentation for your RDBMS for more information.
2. Use the WebSphere administrative console to set the connection pool parameters. Navigate to the **General properties** for your data source and click **Additional properties** → **Connection pool properties**
   a. Set the **Maximum connections** to the number of connections you require, for example, at least 50. The default number of connections is 10.

      **Tip:** If your messaging engine times out when requesting a database connection, check the error log. If the error log contains error message CWSIS1522E, increase the number of connections and ensure that the configuration of your RDBMS permits that number of connections.
   b. Set the **Purge policy** to *EntirePool*. This policy enables the connection pool to release all connections when the messaging engine stops.

      **Important:** You must set this value if the messaging engine can failover in a cluster.

## Controlling the memory buffers used by a messaging engine

To control the sizes of the memory buffers used by a messaging engine you should follow these instructions and tips. Also learn about how to set the messaging engine to improve its interaction with its data store.

Every messaging engine manages two memory buffers that contain messages and message-related data. You can set the following properties to improve the interaction of a messaging engine with its data store.

| Name | Value |
|---|---|
| sib.msgstore.discardableDataBufferSize | The size in bytes of the data buffer used by the messaging engine to contain data for which the quality of service attribute is best effort nonpersistent. The messaging engine holds this data entirely within this memory buffer and never writes this data to the data store. When the messaging engine adds data to this buffer, for example when the messaging engine receives a best effort nonpersistent message from a client, the messaging engine might discard data already in the buffer to make space. This behavior enables the messaging engine to discard best effort nonpersistent messages.<br><br>The discardable data buffer contains all data for which the quality of service attribute is best effort nonpersistent. That data comprises data both that is involved in active transactions, and any other best effort nonpersistent that the messaging engine has neither discarded nor consumed. The messaging engine can discard only data that is not involved in active transactions.<br>**Tip:** If the messaging engine attempts to add data to the discardable data buffer when insufficient space remains after discarding all the data that is not involved in active transaction, the messaging engine throws a `com.ibm.ws.sib.msgstore.OutOfCacheSpace` exception. Client applications can catch this exception, wrapped inside API-specific exceptions such as javax.jms.JMSException.<br><br>The **sib.msgstore.discardableDataBufferSize** property of the messaging engine controls the size of the discardable data buffer. You specify the value of this property in bytes. The default value is 320000, which is approximately 320 kilobytes. |

| Name | Value |
|---|---|
| sib.msgstore.cachedDataBufferSize | The size in bytes of the data buffer used by the messaging engine to contain data for which the quality of service attribute is *better than* best effort nonpersistent and which is held in the data store. The purpose of the cached data buffer is to optimize the performance of the messaging engine by caching in memory the data that the messaging engine might otherwise need to read from the data store. As it writes data to the data store and reads from the data store, the messaging engine attempts to add that data to the cached data buffer. The messaging engine might discard data already in the buffer to make space.<br><br>The **sib.msgstore.cachedDataBufferSize** property of the messaging engine controls the size of the cached data buffer. You specify the value of this property in bytes. The default value is 320000, which is approximately 320 kilobytes. |
| sib.msgstore.transactionSendLimit | The maximum number of operations that the messaging engine includes in each transaction. For example, each JMS send or receive is an operation that counts towards the transaction send limit. The default value is 100. |

**Attention:** The messaging engine uses approximate calculations to manage the data it holds in the memory buffers. Neither of the **DataBufferSize** properties gives an accurate indication of the amount of memory that the messaging engine consumes in the JVM heap. The messaging engine can consume considerably more heap storage than the **DataBufferSize** properties indicate.

To set the properties of a messaging engine to improve its interaction with its data store, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses** → **[Content Pane]** *bus_name* → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] Custom properties**.
2. Type the name of the property that you want to set.
3. Type the value that you want to set for that property.
4. Click **OK**.
5. Save your changes to the master configuration.

**Remember:** When you change any of these properties, the new values do not take effect until you restart the messaging engine.

## Increasing the number of data store tables to relieve concurrency bottleneck

Service integration technologies enables users to spread the data store for a messaging engine across several tables. In typical use this is unlikely to have a significant influence. However, if statistics suggest a concurrency bottleneck on the *SIBnnn* tables for a data store, you might try to solve the problem by increasing the number of tables.

For more information on the set of tables in a data store see Data store tables

| | |
|---|---|
| SIB000 | contains information about the structure of the data in the other two tables – the “stream table” |
| SIB001 | contains persistent objects – the “permanent item table” |
| SIB002 | contains nonpersistent objects that have been saved to the data store to reduce the messaging engine memory requirement – the “temporary item table” |

Having multiple tables means you can relieve any performance bottleneck you might have in your system. You can modify *SIBnnn* tables of the data store of a messaging engine. You can increase the number of permanent and temporary tables (*SIB001* and *SIB002*), although there is no way to increase the number of stream tables (*SIB000*).

This example illustrates what the *SIBnnn* tables for a data store might look like after modification:

| | |
|---|---|
| SIB000 | contains information about the structure of the data in the other two tables – the "stream table" |
| SIB001 | contains persistent objects – the "permanent item table" |
| SIB002 | contains persistent objects – the "permanent item table" |
| SIB003 | contains persistent objects – the "permanent item table" |
| SIB004 | contains nonpersistent objects that have been saved to the data store to reduce the messaging engine memory requirement – the "temporary item table" |
| SIB005 | contains nonpersistent objects that have been saved to the data store to reduce the messaging engine memory requirement – the "temporary item table" |

For instructions on how to configure the data store to use multiple item table, see the following topics:

## Diagnosing problems with your data store configuration

Find out how to diagnose problems that are caused by your data store configuration and possible solutions to these problems.

The following problems depend on the database that you use with your data store configuration and the level of that database:

- Examine this section if your messaging engine uses an Oracle 9i database for its data store and your messaging engine fails to start. If the messaging engine fails with the following message, where `XXXXXXXX` is the schema for the table, ensure that your Oracle installation is at 9.2.0.4, or higher:

  ```
  CWSIS1530E: The data type, 1,111, was found instead of the expected type, 2,004, for column,
  LONG_DATA, in table, XXXXXXXX.SIB000.
  ```
- Examine this section if your messaging engine uses a Sybase database for its data store. When you create your Sybase server:
  - Ensure that you create the database server with a page size of at least 4k.
  - Ensure that you set the **lock scheme** property on your server to the value *datarows*. This avoids the possibility of a deadlock on the data store tables.
- Examine this section if your messaging engine uses an Informix database for its data store and the messaging engine is unable to access its data store. When you configure your messaging engine to use an Informix database, ensure that you specify the schema name in lower case. For a full description of the configuring procedure, refer to Modify data store configurations.

## Avoiding failover problems when you use DB2 v8.2 with HADR as your data store

Use this task to avoid problems that can occur when a messaging engine that is configured to use DB2 v8.2 with the High Availability Data Recovery (HADR) feature for its data store terminates if the DB2 database fails over.

If you use the High Availability Data Recovery (HADR) feature of DB2, note the following restrictions:

- The messaging engine default messaging provider supports only the synchronous and near-synchronous synchronization modes of HADR. The default messaging provider does not support asynchronous HADR configurations.

- The TAKEOVER BY FORCE command is permitted only when the standby database is in peer state, or in a non-peer state (such as disconnected state) having changed from peer state.

### One-phase commit optimization tuning

If you have configured your messaging engine to use a data store, you can achieve better performance by configuring both the messaging engine and container-managed persistent (CMP) beans.

You need to configure both the CMP and the messaging engine's resource authorization so that they share the same data source.

1. Open the administrative console.
2. Click on **Enterprise Applications** > *servername* > **Map data sources for all 2.x CMP beans**.
3. On the content pane, select the check boxes next to all the CMPs.
4. Select *Per application* in the **Resource authorization** selection list.
5. You can modify the messaging engine's resource authorization to *Per application* by modify the property file `sib.properties` by adding the custom property `sib.msgstore.jdbcResAuthForConnections=Application`.

## Setting tuning properties for a mediation

Use this task to tune a mediation for performance using the administrative console.

Before you begin this task, you should review the guidance on when it is appropriate to tune a mediation for performance in the topic Guidance for tuning mediations for performance.

You can set the following tuning property to improve the performance of a mediation:

| Name | Value |
|---|---|
| sib:SkipWellFormedCheck | Whether you want to omit the well formed check that is performed on messages after they have been processed by the mediation. Either true or false. **Note:** This property is overridden for messages that have the delivery option assured persistent, and a well formed check is always performed. |

To set, or unset, one or more tuning properties for a mediation, use the administrative console to complete the following steps:

1. Display the mediation context information:
   a. In the navigation pane, click **Service integration** → **Buses**
   b. In the content pane, click the name of the service integration bus.
   c. In the content pane, under **Destination resources**, click **Mediations**.
   d. In the content pane, select the name of the mediation for which you want to configure tuning information.
   e. Under **Additional Properties**, click **Context information**.
2. In the content pane, click **New**.
3. Type the name of the property in the **Name** field.
4. Select the type `Boolean` in the list box.
5. Type **true** in the **Context Value** field to set the property, or type **false** to unset the property.
6. Click **OK**.
7. Save your changes to the master configuration.

## Enabling CMP entity beans and messaging engine data stores to share database connections

Use this task to enable container-managed persistence (CMP) entity beans to share the database connections used by the data store of a messaging engine. This has been estimated as a potential performance improvement of 15% for overall message throughput, but can only be used for entity beans connected to the application server that contains the messaging engine.

To enable CMP entity beans to share the database connections used by the data store of a messaging engine, complete the following steps:

1. Configure the data store to use a data source that is not XA-capable. For more information about configuring a data store, see Configuring a JDBC data source.
2. Select the Share data source with CMP option.

   This option is provided on the JMS connection factory or JMS activation specification used to connect to the service integration bus that hosts the bus destination that is used to store and process messages for the CMP bean.

   For example, to select the option on a unified JMS connection factory, complete the following steps:

   a. Display the default messaging provider. In the navigation pane, expand **Resources** → **JMS** → **JMS Providers**.
   b. Select the default provider for which you want to configure a unified connection factory.
   c. **Optional:** Change the **Scope** check box to set the level at which the connection factory is to be visible, according to your needs.
   d. In the content pane, under Additional Properties, click **Connection factories**
   e. **Optional:** To create a new unified JMS connection factory, click **New**.

      Specify the following properties for the connection factory:

      **Name** Type the name by which the connection factory is known for administrative purposes.

      **JNDI name**
      Type the JNDI name that is used to bind the connection factory into the name space.

      **Bus name**
      Type the name of the service integration bus that the connection factory is to create connections to. This service integration bus hosts the destinations that the JMS queues and topics are assigned to.
   f. **Optional:** To change the properties of an existing connection factory, click one of the connection factories displayed. This displays the properties for the connection factory in the content pane.
   g. Select the check box for the Share data source with CMP field
   h. Click **OK**.
   i. Save your changes to the master configuration.

   The JMS connection factory can only be used to connect to a "local" messaging engine that is in the application server on which the CMP beans are deployed.
3. Deploy the CMP beans onto the application server that contains the messaging engine, and specify the same data source as used by the messaging engine. You can use the administrative consoles to complete the following steps:

   a. **Optional:** To determine the data source used by the messaging engine, click **Servers** → **Application servers** → **server_name** → **Messaging engines** → **engine_name** → **Data store** The **Data source name** field displays the name of the data source; by default:

      `jdbc/com.ibm.ws.sib/`*`engine_name`*
   b. Click **Applications** → **Install New Application**
   c. On the first Preparing for application install page, specify the full path name of the source application file (.ear file otherwise known as an EAR file), then click **Next**

   d. On the second Preparing for application install page, complete the following steps:
      1) Select the check box for the Generate Default Bindings property. Data source bindings (for EJB 1.1 JAR files) are generated based on the JNDI name, data source user name password options. This results in default data source settings for each EJB JAR file. No bean-level data source bindings are generated.
      2) Under Connection Factory Bindings, click the check box for the **Default connection factory bindings:** property, then type the JNDI name for the data source and optionally select a **Resource authorization** value.
      3) Click **Next**
4. If your application uses EJB modules that contain Container Managed Persistence (CMP) beans that are based on the EJB 1.x specification, for Step: Provide default data source mapping for modules containing 1.x entity beans, specify a JNDI name for the default data source for the EJB modules. The default data source for the EJB modules is optional if data sources are specified for individual CMP beans.
5. If your application has CMP beans that are based on the EJB 1.x specification, for Step: Map data sources for all 1.x CMP, specify a JNDI name for data sources to be used for each of the 1.x CMP beans. The data source attribute is optional for individual CMP beans if a default data source is specified for the EJB module that contains CMP beans. If neither a default data source for the EJB module nor a data source for individual CMP beans are specified, then a validation error displays after you click Finish (step 13) and the installation is cancelled.
6. Complete other panels as needed.
7. On the Summary panel, verify the cell, node, and server onto which the application modules will install:
   a. Beside Cell/Node/Server, click **Click here**.
   b. Verify the settings on the Map modules to servers page displayed. Ensure that the application server that is specified contains the messaging engine and its data store.
   c. Specify the Web servers as targets that will serve as routers for requests to this application. This information is used to generate the plug-in configuration file (plugin-cfg.xml) for each Web server.
   d. Click **Finish**.

For more information about installing applications, see Installing application files with the console.

## Tuning service integration technologies

Use this task to set tuning properties that control the performance of message-driven beans and other messaging applications deployed to use service integration technologies.

To optimize the performance of messaging with service integration technologies, such as message-driven beans that use the default messaging provider, you can use the following parameters set through the WebSphere administrative console or command line interfaces.

On z/OS, the performance of messaging applications is affected by the number of servants, which can vary dynamically, and the distribution of work between servants. For more information about configuring and managing the number of servants and the distribution of work between servants, see Tuning the application serving environment.

- Viewing the Available Message Count on a destination enables you to determine whether your message consumers are able to cope with your current workload. If the available message count on a given destination is too high, or is increasing over time, you should consider some of the tuning recommendations on this page.
   1. To monitor the available message count for a queue, you need to enable runtime AvailableMessageCount statistics for the queue. If you restart administrative server, you need to enable AvailableMessageCount statistics again because such runtime settings are not preserved when the server is restarted.

To enable AvailableMessageCount statistics using the administrative console, complete the following steps:

a. In the navigation pane, click **Monitoring and Tuning** → **Performance Monitoring Infrastructure (PMI)**

b. In the content pane, click *server_name*

c. Click the Runtime tab.

d. In the Currently monitored statistic set, click **Custom**

e. On the Custom monitoring level panel, click **SIB Service** → **SIB Messaging Engines** → *messageEngine_name* → **Destinations** → **Queues** → *queue_name*

f. Select the AvailableMessageCount option.

g. Click the **Enable** button at the top of the panel.

2. To view the available message count, you can use the administrative console to complete the following steps:

   a. In the navigation pane, click **Monitoring and Tuning** → **Performance Viewer** → **Current activity**

   b. In the content pane, click *server_name*

   c. Click **Performance Modules** → **SIB Service** → **SIB Messaging Engines** → *messageEngine_name* → **Destinations** → **Queues** → *queue_name*

   d. Click the **View Module(s)** button at the top of the Resource Selection panel, located on the left side. This displays the AvailableMessageCount data in the Data Monitoring panel, located on the right side.

      You can use the Data Monitoring panel to manage the collection of monitoring data; for example, you can use the buttons to start or stop logging, or to change the data displayed as either a table or graph.

- Monitoring MDB Thread Pool Size for the Default Message Provider. You may experience a performance bottleneck if there are insufficient threads available for the Message Driven Beans. There is a trade-off between providing sufficient threads to maximize the throughput of messages and configuring excessive threads, which can lead to CPU starvation of the threads in the application server. If you notice that the throughput for express nonpersistent, reliable nonpersistent, or reliable persistent messaging has fallen as a result of increasing the size of the default thread pool, then you should decrease the size of the thread pool and reassess the message throughput.

  1. By default MDBs use the default thread pool. To view or change the number of threads in the default thread pool for an application server, you can use the administrative console to complete the following steps:

     a. In the navigation pane, click **Servers** → **Application servers**

     b. In the content pane, click *server_name*

     c. Under Additional properties, click **Thread Pools** → **Default**. By default the Minimum size value is set to 5 and the Maximum size value is set to 20. The best performance is obtained by setting the Maximum size value to the expected maximum concurrency for all message-driven beans. For high throughput using a single message-driven bean, 41 was found to be the optimal Maximum size value.

     d. To change the Maximum size value, type the new value in the Maximum size field then click **OK**. Finally, save your changes to the master configuration.

  2. As the default thread pool is also used by other WAS components it can be beneficial to define a separate thread pool for the MDBs. This will reduce thread contention for the default thread pool. To create your own thread pool you can use the administrative console to complete the following steps:

     a. In the navigation pane, click **Servers** → **Application servers**

     b. In the content pane, click *server_name*

     c. Under Additional properties, click **Thread Pools**. Create a new thread pool. Create sufficient threads to support the maximum amount of concurrent work for the MDBs.

   d. b. Change the SIB JMS Resource Adapter to use the new thread pool: **Resources** -> **Resource Adapters** -> **Resource Adapters**.
   e. Open **Preferences** and select the **SIB JMS Resource Adapter** with the appropriate scope depending upon the scope of the connection factories. Add the name of the new thread pool in the **Thread pool alias** box. Click **Apply** and save the changes.
- Tuning MDB performance with the default messaging provider.
  1. The maximum concurrent endpoints parameter controls the amount of concurrent work that can be processed by an MDB. The parameter is applicable to MDBs using an activation specification. Increasing the number of concurrent endpoints can improve performance but can increase the number of threads in use at one time. To benefit from a change in this parameter, there should be sufficient threads available in the MDB thread pool to support the concurrent work. If message ordering must be retained across failed deliveries this parameter should be set to 1. This parameter can be set from the administrative console:
     a. Click on **Resources** -> **JMS** -> **Activation Specification**.
  2. 2. Delivering batches of messages to each MDB endpoint can improve performance particularly when used with Acknowledge mode set to Duplicates-ok auto-acknowledge. This parameter is applicable to MDBs using an activation specification. If message-ordering must be retained across failed deliveries, the batch size should be set to 1. This parameter can be set from the administrative console:
     a. Click on **Resources** -> **JMS** -> **Activation Specification**.

     For additional information about tuning the throttling of message-driven beans, including controlling the maximum number of instances of each message-driven bean and the message batch size for serial delivery, see Configuring MDB throttling on the default messaging provider.
- Reducing the number of OutOfCacheSpace errors in the SystemOut.log file.

  OutOfCacheSpace errors in the SystemOut.log file indicate that the discardable data buffer used by the messaging engine is overflowing. For best effort nonpersistent messages, the messaging engine starts to discard messages when this buffer is full. You can increase the size of this data buffer to allow more best effort nonpersistent data to be handled before the messaging engine begins to discard the messages.

  For more information about tuning the size of the discardable data buffer, set by the sib.msgstore.discardableDataBufferSize property of a messaging engine, see "Controlling the memory buffers used by a messaging engine" on page 90.
- Reducing the occurrence of OutOfMemoryError exceptions when processing a large set of messages within a transaction. If the cumulative size of the set of messages being processed within a transaction by the service integration bus is large enough to exhaust the JVM heap, OutOfMemoryError exceptions occur. Consider one of the following options:
  – Increase the heap size for the Java Virtual Machine (JVM) used by the WebSphere Application Server by setting the Initial Heap Size and Maximum Heap Size properties of the application server. To view the administrative console page, click **Servers** → **Application Servers** → ***server_name*** → **Server Infrastructure** → **Process Definition** → **Java Virtual Machine**. For more information about changing the JVM configuration for the application server, see Java virtual machine settings.
  – Reduce the cumulative size of the set of messages being processed within the transaction.
- Changing the maximum connections in a Connection Factory for the default messaging provider. The maximum connections parameter limits the number of local connections. The default is 10. This parameter should be set to a number equal to or greater than the number of threads (enterprise beans) concurrently sending messages. Using the administrative console you can set the Maximum connections property as follows:
  1. Click on **Resources** -> **JMS** -> **Topic Connection Factory** -> *factory_name* -> **Connection pool properties**
  2. Enter the required value in the **Maximum connections** field.
  3. Click **Apply** and save the changes to master configurations.

- Tuning the messaging engine message stores
  - For file store configurations see File stores .
  - For tuning information of JDBC data sources see “Tuning and problem solving for messaging engine data stores” on page 89
- Additional tuning advice for a messaging engine using a JDBC data source.

  To improve the performance of messaging throughput of a messaging engine data store, you can tune the JDBC connection pool and statement cache size. In tests of high throughput MDB workloads, the following changes provided a 10% gain in throughput.

  1. The messaging engine uses a connection pool for managing the JDBC connections to its data store. Tuning the size of the pool can improve the messaging throughput.

     To view or change the size of the connection pool, you can use the administrative console to complete the following steps:

     a. In the navigation pane, click **Resources** → **JDBC Providers**

     b. In the content pane, click *jdbc_provider_name*

     c. Under Additional properties, click **Data sources** → *data_source_name*

     d. Under Additional properties, click **Connection pool properties**

     e. View the Maximum connections property and the Minimum connections property. By default, these properties are set to Maximum connections=10 and Minimum connections=1. Setting the value of both these properties to 50 is recommended. For especially high throughput workloads, setting the value of both these properties up to 100 can be beneficial. You may need to configure the underlying database to accept this many concurrent connections.

     f. To change the value of a property, type a new value in the property field then click **OK**. Finally, save your changes to the master configuration.
  2. The statement cache contains recently used prepared statements to remove the costs associated with repeated preparation of statements. Tuning the size of the cache helps prevent useful entries from being discarded to make room for new entries.

     To view or change the size of the statement cache, you can use the administrative console to complete the following steps:

     a. In the navigation pane, click **Resources** → **JDBC Providers**

     b. In the content pane, click *jdbc_provider_name*

     c. Under Additional properties, click **Data sources** → *data_source_name*

     d. Under Additional properties, click **WebSphere Application Server data source properties**

     e. View the Statement cache size property. By default, the value of this property is set to 10. For high throughput JMS messaging, a value of 40 is recommended.

     f. To change the value of the property, type a new value in the property field then click **OK**. Finally, save your changes to the master configuration.
- Tuning reliability levels for messages.

  The reliability level chosen for the messages has a significant impact on performance. In order of decreasing performance (fastest first), the reliability levels are: Best-Effort Nonpersistent, Express Nonpersistent, Reliable Nonpersistent, Reliable Persistent, and Assured Persistent. For MDB point-to-point messaging, best-effort nonpersistent throughput is more than 6 times greater than assured persistent.

  For more information about reliability levels, see Message reliability levels.
- Tuning MDB performance with the default messaging provider.

  For information about tuning the throttling of message-driven beans, including controlling the maximum number of instances of each message-driven bean and the message batch size for serial delivery, see Configuring MDB throttling on the default messaging provider.

## Tuning the SIBWS

You can use the administrative console or a Jacl script to tune performance settings for the service integration bus Web services enablement (SIBWS).

The SIBWS dynamically selects an optimized route through the code where possible. If you migrate Web services from the WebSphere Application Server Version 5 Web services gateway, and you do not use mediations to support previous Gateway filter applications, then your messages avoid being routed through the internal infrastructure that enables additional SIBWS functionality. This fast-path route through the bus is used if the following criteria are met:

- The inbound port and outbound port for the service are on the same server.
- There are no mediations on the path from the inbound port to the outbound port.

Further optimizations can be made, if your configuration meets the previous two criteria, and also meets the following criteria:

- The inbound template WSDL URI is the same location as the Outbound Target Service WSDL location URI.
- The inbound service template WSDL service name matches the outbound WSDL service name.
- The inbound service template port name matches the outbound WSDL port name.
- The mapping of the namespaces is disabled (that is, you have set the inbound service property **com.ibm.websphere.wsgw.mapSoapBodyNamespace** to `false`).
- Operation-level security is not enabled on the outbound service.

If your Web services use the fast-path route, you need not tune mediations or the service integration bus. However it is good practise to do so, because a typical environment will have at least one non-fast-path (for example, mediated) service.

To improve the performance of the SIBWS, you can tune the following parameters:

- The Java virtual machine heap size. This helps ensure there is enough memory available to process large messages, or messages with large attachments.
- The maximum number of instances of a message-driven bean that are permitted by the activation specification for the service integration technologies resource adapter. This throttles the number of concurrent clients serviced.
- The maximum batch size for batches of messages to be delivered to a client. By default, only a single message is delivered to a message-driven bean instance at one time; you can improve performance by allowing messages to be sent in batches to a message-driven bean.
- The number of threads available to service requests for each client. That is, the number of threads available in the default thread pool, the Web container thread pool and the mediation thread pool for a given application server.
- The number of threads available in the mediation thread pool. This assumes that your mediations use concurrent support where appropriate, as explained in Concurrent mediations.

If you have mediations that act on SOAP headers, you can improve performance by inserting the associated header schemas (`.xsd` files) into the SDO repository.

To tune the SIBWS, complete one of the following two steps:

- Use the administrative console to tune the SIBWS, or
- Use a Jacl script to tune the SIBWS.

If you have mediations that act on SOAP headers, also complete the following step:

- Insert the header schemas into the SDO repository.
- **Optional:** To use the administrative console to tune the SIBWS, complete the following steps:

1. Use the topic Tuning Java virtual machines to set the JVM heap size to a larger value than the default value (256 megabytes). The value should generally be as large as possible without incurring paging.
2. Use the topic Tuning service integration messaging to tune the maximum number of instances of a message-driven bean, the maximum batch size for batches of messages for a bean, and the number of threads available to service requests for a bean.
3. Use the topic Tuning the application serving environment to tune the general application serving environment, in particular the size of the Web Container Thread Pool. In a server which is exclusively serving requests to the SIBWS the default thread pool and the Web Container thread pool should be the same size.
4. Use the topic Configuring the mediation thread pool to configure the number of threads available to concurrent mediations.

- To use a Jacl script to tune the SIBWS, use the wsadmin scripting client to run a script based on the following example:

```
#----------------------------------------------------------------------
# SIBWS WebSphere Tuning Script
#----------------------------------------------------------------------
##
# This script is designed to modify some of the tuning pertinent to a SIBWS
# deployment.
# In order to tune the config parameters, simply change the values
# provided below. This script assumes that all server names in a
# cluster configuration are unique.
#
# To invoke the script, type:
#   wsadmin -f tuneWAS.jacl <scope> <id>
#      scope      - 'cluster' or 'server'
#      id         - name of target object within scope (i.e. servername)
#
# Examples:
#   wsadmin -f tuneWAS.jacl server server1
#   wsadmin -f tuneWAS.jacl cluster WSGWCluster#
#
#----------------------------------------------------------------------
$AdminConfig setValidationLevel NONE

puts "Starting script..."
puts "Reading config parameters..."

#----------------------------------------------------------------------
# COMMON CONFIG PARAMETERS
# - Adjust these parameters based on the intended target system (Defaults in parentheses)
#----------------------------------------------------------------------
# WebContainer Thread Pool (10,50)
set minWebPool     10
set maxWebPool     15

# Default Thread Pool - (Multiprotocol MDB) (10,50)
set minDefaultPool     10
set maxDefaultPool     15

# Mediations Thread Pool (1,5)
set minMediationPool 10
set maxMediationPool 15

# HTTP KeepAlive settings (true, 100)
set keepAliveEnabled      true
set maxPersistentRequests -1

# Inactivity Timeouts for thread pools (3500)
set inactivity     3500

# JVM properties
```

```
set minHeap        1280
set maxHeap        1280
set verboseGC      "false"
set genericArgs    ""

# J2CActivationSpec for the SIB_RA Resource adapter
set SIB_RA_maxConcurrency 15
set SIB_RA_maxBatchSize   5

# Java2 Security (false for 5.1 and true for 6.0)
set j2Security     false

# Parallel server startup
set parallelStart  false


#---------------------------------------------
# Check/Print Usage
#---------------------------------------------

proc printUsageAndExit {} {
   puts " "
   puts "Usage: wsadmin -f tuneWAS.jacl <cluster | server> <name>"
   exit
}

#---------------------------------------------
# Misc Procedures
#---------------------------------------------

proc getName {objectid} {
   set endIndex [expr [string first "(" $objectid] - 1]

   return [string range $objectid 0 $endIndex]
}


#---------------------------------------------
# Parse command line arguments
#---------------------------------------------

puts "Parsing command line arguments..."

if {[llength $argv] < 2} {
   printUsageAndExit
} else {
   set scope [lindex $argv 0]
   puts "Scope:   ${scope}"

   if {$scope == "cluster"} {
      set clustername [lindex $argv 1]
      puts "Cluster: ${clustername}"
   } elseif {$scope == "server"} {
      set servername [lindex $argv 1]
      puts "Server:  ${servername}"
   } else {
      puts "Error: Invalid Argument ($scope)"
      printUsageAndExit
   }
}

#---------------------------------------------
# Obtain server list
#---------------------------------------------

puts ""
puts "Obtaining server list..."
```

```
if {$scope == "cluster"} {
   set cluster [$AdminConfig getid "/ServerCluster:${clustername}/"]
   set temp [$AdminConfig showAttribute $cluster members]
   set memberList [split [string trim $temp "{ }"] " "]
   foreach member $memberList {
      set memberName [getName $member]
      lappend serverList [$AdminConfig getid "/Server:${memberName}/"]
   }
} else {
   set server [$AdminConfig getid "/Server:${servername}/"]
   lappend serverList $server
}


#---------------------------------------------
# Print config properties
#---------------------------------------------

puts ""
puts "WebSphere configuration"
puts "-----------------------"
puts ""
puts "   Enforce Java2 Security:      ${j2Security} "
puts ""

puts "Servers:"
foreach server $serverList {
   puts "   [getName $server]"
}
puts ""
puts " Web --------------------------------------------"
puts "   Min WebContainer Pool Size:  ${minWebPool} "
puts "   Max WebContainer Pool Size:  ${maxWebPool} "
puts " JVM --------------------------------------------"
puts "   Min JVM Heap Size:           ${minHeap} "
puts "   Max JVM Heap Size:           ${maxHeap} "
puts "   Verbose GC:                  ${verboseGC}"
puts ""


#---------------------------------------------
# Modify cell parameters
#---------------------------------------------

# Accessing cell based security config
puts "Accessing security configuration..."
set sec [$AdminConfig list Security]
set attrs [subst {{enforceJava2Security $j2Security}}]
puts "Updating security..."
$AdminConfig modify $sec $attrs



#---------------------------------------------
# Modify server parameters
#---------------------------------------------

foreach server $serverList {
   set servername [getName $server]
   puts ""
   puts "Server: $servername"
   puts ""


   # Accessing server startup config
   puts "Accessing server startup configuration..."
   puts "Parallel Startup (old/new):  [$AdminConfig showAttribute $server parallelStartEnabled]/
$parallelStart"
   set attrs [subst {{parallelStartEnabled $parallelStart}}]
   puts "Updating server startup..."
   puts ""
   $AdminConfig modify $server $attrs

   # Accessing web container thread pool config
   puts "Accessing web container thread pool configuration..."
   set tpList [$AdminConfig list ThreadPool $server]

   set oI [lsearch -glob $tpList "*WebContainer*"]
   set webPool [lindex $tpList $oI]
   puts "ThreadPool MaxSize (old/new):             [$AdminConfig showAttribute $webPool maximumSize
]/$maxWebPool"
   puts "ThreadPool MinSize (old/new):             [$AdminConfig showAttribute $webPool minimumSize
]/$minWebPool"
   puts "ThreadPool Inactivity Timeout (old/new):  [$AdminConfig showAttribute $webPool
inactivityTimeout]/$inactivity"
   set attrs [subst {{maximumSize $maxWebPool} {minimumSize $minWebPool} {inactivityTimeout
$inactivity}}]
   puts "Updating web container thread pool..."
   puts " "
   $AdminConfig modify $webPool $attrs

   # Accessing default thread pool config
   puts "Accessing default thread pool configuration..."
   set tpList [$AdminConfig list ThreadPool $server]

   set oI [lsearch -glob $tpList "*Default*"]
   set webPool [lindex $tpList $oI]
   puts "ThreadPool MaxSize (old/new):             [$AdminConfig showAttribute $webPool maximumSize
]/$maxDefaultPool"
   puts "ThreadPool MinSize (old/new):             [$AdminConfig showAttribute $webPool minimumSize
]/$minDefaultPool"
   puts "ThreadPool Inactivity Timeout (old/new):  [$AdminConfig showAttribute $webPool
inactivityTimeout]/$inactivity"
   set attrs [subst {{maximumSize $maxDefaultPool} {minimumSize $minDefaultPool} {inactivityTimeout
$inactivity}}]
   puts "Updating default thread pool..."
   puts " "
   $AdminConfig modify $webPool $attrs

   # Creating Mediations Thread Pool
   puts "Creating Mediations thread pool"
   set me [$AdminConfig list SIBMessagingEngine]
   set mtpName [$AdminConfig showAttribute $me name]-mediationThreadPool
   set tpAttrs [subst {{name $mtpName} {minimumSize $minMediationPool} {maximumSize
$maxMediationPool}}]
   puts "ThreadPool Name    : $mtpName"
   puts "ThreadPool MaxSize : $maxMediationPool"
   puts "ThreadPool MinSize : $minMediationPool"
   $AdminConfig create ThreadPool $me $tpAttrs mediationThreadPool
   puts "Mediations Thread Pool Created"
   puts " "

   # Accessing HTTP keepalive config
   puts "Accessing HTTP KeepAlive configuration..."
   set HTTPInbound [$AdminConfig list HTTPInboundChannel $server]

   set oI [lsearch -glob $HTTPInbound "*HTTP_2*"]
   set http2 [lindex $HTTPInbound $oI]
   puts "KeepAlive Enabled (old/new):        [$AdminConfig showAttribute $http2 keepAlive]/
$keepAliveEnabled"
   puts "Max Persistent Requests (old/new):  [$AdminConfig showAttribute $http2
maximumPersistentRequests]/$maxPersistentRequests"
   set attrs [subst {{keepAlive $keepAliveEnabled} {maximumPersistentRequests
$maxPersistentRequests}}]
   puts "Updating HTTP KeepAlives..."
   puts " "
   $AdminConfig modify $http2 $attrs

   # Accessing JVM config
   puts "Accessing JVM configuration..."
   set jvm [$AdminConfig list JavaVirtualMachine $server]
   puts "Initial Heap Size (old/new):  [$AdminConfig showAttribute $jvm initialHeapSize]/$minHeap"
   puts "Maximum Heap Size (old/new):  [$AdminConfig showAttribute $jvm maximumHeapSize]/$maxHeap"
   puts "VerboseGC Enabled (old/new):  [$AdminConfig showAttribute $jvm
verboseModeGarbageCollection]/$verboseGC"
   set attrs [subst {{initialHeapSize $minHeap} {maximumHeapSize $maxHeap}
{verboseModeGarbageCollection $verboseGC} }]
   puts "Updating JVM..."
   puts " "
   $AdminConfig modify $jvm $attrs

   # Accessing J2CActivationSpec for the SIB Resource Adapter
   puts "Modifying the J2CActivationSpec for the SIB Resource Adapter"
   set actSpec [$AdminConfig getid /J2CActivationSpec:SIBWS_OUTBOUND_MDB/]
   set propSet [$AdminConfig showAttribute $actSpec resourceProperties]

   set propSet [lindex $propSet 0]

   set maxConcurrency [list value $SIB_RA_maxConcurrency]
   set maxConcurrency [list $maxConcurrency ]

   set maxBatchSize   [list value $SIB_RA_maxBatchSize]
   set maxBatchSize   [list $maxBatchSize]

   foreach propId $propSet {
      if { [string compare [$AdminConfig showAttribute $propId name] maxConcurrency] == 0} {
    $AdminConfig modify $propId $maxConcurrency
    puts "Custom property changed : [$AdminConfig showall $propId] "
      }
      if { [string compare [$AdminConfig showAttribute $propId name] maxBatchSize] == 0} {
    $AdminConfig modify $propId $maxBatchSize
    puts "Custom property changed : [$AdminConfig showall $propId] "
      }
   }
   puts "J2CActivationSpec modifications complete"


}


puts ""
puts "Script completed..."
puts "Saving config..."
$AdminConfig save
```

- **Optional:** If you have mediations that act on SOAP headers, insert the associated schemas (`.xsd` files) into the SDO repository as described in “Including SOAP header schemas in the SDO repository.”

## Including SOAP header schemas in the SDO repository

Mediations accessing SOAP headers should ensure that the SOAP header schema is made available to the SDO repository. This simplifies access to the header fields (see Web Services code example) and can provide a significant performance benefit. Normally the schema (`.xsd` file) for a SOAP header is already available to the application developer.

Here is an example of a header (used for routing) that is passed in the SOAP message:

```
<soapenv:Header>
<hns0:myClientToken xmlns:hns0="http://www.ibm.com/wbc">
           <UseRoutingId>true</ UseRoutingId >
           <RoutingID>5</ RoutingID >
       </hns0: myClientToken >
</soapenv:Header>
```

Here is an example of an associated header schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.ibm.com/wbc"
           elementFormDefault="unqualified">
<xs:element name=" myClientToken">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="UseRoutingId" type="xs:string"/>
      <xs:element name="RoutingID" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

To insert the schema into the SDO repository, complete the following steps:

1. Create a Jacl script called `sdoXSDImport.jacl` that contains the following code:

```
#
set xsdFile [lindex $argv 0]
set xsdKey  [lindex $argv 1]
set sdoRep [$AdminControl queryNames *,type=SdoRepository,node=[$AdminControl getNode]]
puts [$AdminControl invoke $sdoRep importResource [list $xsdKey $xsdFile]]
```

**Note:** To create an equivalent script for removing a resource from the SDO repository, take a copy of this script and modify the final line as follows:

```
$AdminControl invoke $sdoRep removeResource [list $xsdKey false]
```

2. Use the wsadmin scripting client to insert the schema into the SDO repository by entering the following command:

```
wsadmin -f sdoXSDImport.jacl your_header.xsd your_header_namespace
```

where

- *your_header*`.xsd` is the name of the file that contains your header schema.
- *your_header_namespace* is the target namespace for the header. For example `http://yourCompany.com/yourNamespace`.

## Setting tuning properties for service integration

Use this task to set tuning properties for the service integration environment.

The service integration environment includes properties that you can set to improve the performance of a messaging engine or the component of the messaging engine that manages the data store. These properties are known collectively as “tuning properties”. You can set these properties either with the WebSphere administrative console or by editing the sib.properties file.

**Tip:** Properties set with the WebSphere administrative console take precedence over properties set in the sib.properties file.

To set tuning properties using the administrative console, click the relevant link from the following list:

- “Setting tuning properties of a messaging engine” on page 88.
- “Controlling the memory buffers used by a messaging engine” on page 90.

You can also use the administrative console to tune the data source. Refer to "Tuning the JDBC data source of a messaging engine" on page 89.

To set tuning properties for any of the components mentioned above by editing the sib.properties file, refer to Setting tuning properties by editing the sib.properties file.

**Sub-topics**

## Setting tuning properties of a messaging engine

Use this task to set the tuning properties for a messaging engine.

You can set the following property to improve the performance of a messaging engine:

| Name | Value |
|---|---|
| sib.trm.retry | The messaging engine to messaging engine connection retry interval in seconds. The retry interval is the time delay left between attempts to contact neighboring messaging engines with which communications contact should exist. The default retry interval is 30 seconds. |

To set the tuning properties for a messaging engine, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses** → **[Content Pane]** *bus_name* → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] Custom properties**.
2. Type the name of the property that you want to set.
3. Type the value that you want to set for that property.
4. Click **OK**.
5. Save your changes to the master configuration.
6. Restart the messaging engine for the changes to take effect.

## Controlling the memory buffers used by a messaging engine

To control the sizes of the memory buffers used by a messaging engine you should follow these instructions and tips. Also learn about how to set the messaging engine to improve its interaction with its data store.

Every messaging engine manages two memory buffers that contain messages and message-related data. You can set the following properties to improve the interaction of a messaging engine with its data store.

| Name | Value |
|---|---|
| sib.msgstore.discardableDataBufferSize | The size in bytes of the data buffer used by the messaging engine to contain data for which the quality of service attribute is best effort nonpersistent. The messaging engine holds this data entirely within this memory buffer and never writes this data to the data store. When the messaging engine adds data to this buffer, for example when the messaging engine receives a best effort nonpersistent message from a client, the messaging engine might discard data already in the buffer to make space. This behavior enables the messaging engine to discard best effort nonpersistent messages.<br><br>The discardable data buffer contains all data for which the quality of service attribute is best effort nonpersistent. That data comprises data both that is involved in active transactions, and any other best effort nonpersistent that the messaging engine has neither discarded nor consumed. The messaging engine can discard only data that is not involved in active transactions.<br>**Tip:** If the messaging engine attempts to add data to the discardable data buffer when insufficient space remains after discarding all the data that is not involved in active transaction, the messaging engine throws a `com.ibm.ws.sib.msgstore.OutOfCacheSpace` exception. Client applications can catch this exception, wrapped inside API-specific exceptions such as javax.jms.JMSException.<br><br>The **sib.msgstore.discardableDataBufferSize** property of the messaging engine controls the size of the discardable data buffer. You specify the value of this property in bytes. The default value is 320000, which is approximately 320 kilobytes. |
| sib.msgstore.cachedDataBufferSize | The size in bytes of the data buffer used by the messaging engine to contain data for which the quality of service attribute is *better than* best effort nonpersistent and which is held in the data store. The purpose of the cached data buffer is to optimize the performance of the messaging engine by caching in memory the data that the messaging engine might otherwise need to read from the data store. As it writes data to the data store and reads from the data store, the messaging engine attempts to add that data to the cached data buffer. The messaging engine might discard data already in the buffer to make space.<br><br>The **sib.msgstore.cachedDataBufferSize** property of the messaging engine controls the size of the cached data buffer. You specify the value of this property in bytes. The default value is 320000, which is approximately 320 kilobytes. |
| sib.msgstore.transactionSendLimit | The maximum number of operations that the messaging engine includes in each transaction. For example, each JMS send or receive is an operation that counts towards the transaction send limit. The default value is 100. |

**Attention:** The messaging engine uses approximate calculations to manage the data it holds in the memory buffers. Neither of the **DataBufferSize** properties gives an accurate indication of the amount of memory that the messaging engine consumes in the JVM heap. The messaging engine can consume considerably more heap storage than the **DataBufferSize** properties indicate.

To set the properties of a messaging engine to improve its interaction with its data store, use the administrative console to complete the following steps:

1. In the navigation pane, click **Service integration** → **Buses** → **[Content Pane]** *bus_name* → **[Topology] Messaging engines** → ***engine_name*** → **[Additional Properties] Custom properties**.
2. Type the name of the property that you want to set.
3. Type the value that you want to set for that property.
4. Click **OK**.

5. Save your changes to the master configuration.

**Remember:** When you change any of these properties, the new values do not take effect until you restart the messaging engine.

## Tuning the JDBC data source of a messaging engine

The messaging engine needs to have the correct configuration for JDBC data source to achieve messaging performance on a service integration bus.

Consider whether you need to configure the connection pool for the JDBC data source to achieve your requirements for messaging performance.

The messaging engine uses the connection pool to obtain its connections to the database. With a heavy workload, a messaging engine might require a large number of concurrent connections to avoid delays waiting for connections to become available in the pool. For example, a very heavily loaded messaging engine might need 50 or more connections. Perform the following steps to configure the connection pool to meet your performance requirements:

1. Ensure that the configuration of your relational database management system (RDBMS) permits the number of connections that you require. Refer to the documentation for your RDBMS for more information.
2. Use the WebSphere administrative console to set the connection pool parameters. Navigate to the **General properties** for your data source and click **Additional properties** → **Connection pool properties**
   a. Set the **Maximum connections** to the number of connections you require, for example, at least 50. The default number of connections is 10.

      **Tip:** If your messaging engine times out when requesting a database connection, check the error log. If the error log contains error message CWSIS1522E, increase the number of connections and ensure that the configuration of your RDBMS permits that number of connections.
   b. Set the **Purge policy** to *EntirePool*. This policy enables the connection pool to release all connections when the messaging engine stops.

      **Important:** You must set this value if the messaging engine can failover in a cluster.

## Workload classification for service integration work

This topic describes the syntax for z/OS workload manager classification for inbound message-driven bean and mediation work for service integration technologies.

### Usage notes

You use the syntax described in this topic to classify service integration work in the workload classification document for the z/OS workload manager. This topic provides a service integration subset of the workload classification document syntax, which is provided in Workload classification file. For more information about using this syntax to create a workload classification document, see Classifying z/OS workload when you use the workload classification document.

### Required elements

**<?xml version=″1.0″ encoding=″UTF-8″>**
: Indicates that the workload classification document must be saved in ASCII to be processed by the application server. This statement is required.

**<!DOCTYPE Classification SYSTEM ″Classification.dtd″>**
: Gives the XML parser with the name of the DTD document provided by WebSphere Application Server for z/OS that validates the workload classification document. The workload classification

document that you write must follow the rules that are described in this DTD. You must add this statement to the workload classification document.

**Classification**

```
<Classification schema_version="1.0">
```

Indicates the root of the workload classification document. Every workload classification document must begin and end with this element. The schema_version attribute is required. The only supported schema_version is 1.0. For inbound service integration work, the Classification element can contain up to two SibClassification elements, as described in this topic. The Classification element can also contain one or more InboundClassification elements for IIOP, HTTP, or EJB 2.0 message-driven bean work. For more information about InboundClassification elements, see Workload classification file.

**SibClassification**

```
<SibClassification type="jmsra | destinationmediation" schema_version="1.0"
default_transaction_class="value">
```

Use the following rules when using the SibClassification element:

- The **type** attribute is required. The value must be jmsra or destinationmediation. There can be at most one SibClassification element in the document for each type. The types do not have to be specified in a certain order in your classification document.
- The **schema_version** attribute is required. The value must be set to 1.0.
- The **default_transaction_class** attribute must be specified, and defines the default transaction class for work flows of the specified type. The string value must be a valid WLM transaction class, a null string (such as "") or a string that contains eight or fewer blanks (such as " ").
- The SibClassification elements cannot be nested. Each SibClassification element must end before the next InboundClassification element or SibClassification element can begin.
- If a message does not match any sib_classification_info element in an enclosing SibClassification element, the message is assigned the default classification from the SibClassification element.

  If a message does not match any sib_classification_info element in *any* SibClassification element, or if *no* SibClassification elements are defined, all work receives a built-in default classification with the value “SIBUS”. You must perform z/OS Workload Manager actions that are required to use the TCLASS value “SIBUS”, as described in Classifying z/OS workload.

The rules and XML statements for classifying different types of work are very similar, but there is slightly different syntax for each type. For more information about the syntax for each type of work, see the following sections:

- “JMS RA classification”
- “Mediation classification” on page 111

## JMS RA classification

The SibClassification element with the attribute type="jmsra" defines the section of the document that applies to work for message-driven beans (MDBs) deployed against JCA 1.5-compliant resources for use with the JCA resource adapter (RA) of the default messaging provider. An example of this element follows:

```
<SibClassification  type="jmsra"
                    schema_version="1.0"
                    default_transaction_class="a">
```

Each SibClassification element can contain one or more sib_classification_info elements. An example of a classification entry element follows:

```
<sib_classification_info  selector="&apos;East&apos;"
                      transaction_class="sibb"
                      selector="user.Location=&apos;East&apos;"
                      bus="bigrred"
                      destination="abusqueue"
                      description="Some words" />
```

**selector**
Use the selector attribute of the sib_classification_info element to assign a transaction class to a message-driven bean that has a selector clause in its deployment descriptor. Use the following rules when defining your sib_classification_info elements:

- The value of the selector attribute must match exactly to the selector clause in the MDB deployment descriptor. It is an SQL expression that selects a message according to the values of the message properties. The syntax is that of a message selector in the JMS 1.1 specification, but it can operate on SIMessage messages (more than JMS messages). The syntax can select on system properties (including JMS headers, JMSX properties, and JMS_IBM_properties) and user properties (which must be prefixed by ".user" - for example, for the user property "Location", the selector would specify "user.Location" as shown in the preceding example). For more information, see Working with the message properties.
- The value of the selector attribute must have the correct syntax for an XML document. You must replace the < and > symbols with the entity references `&lt;` and `&gt;`, respectively. Similarly, if you use an apostrophe or quotation mark, use the `&apos;` and `&quot;` entity references.

**bus** The name of the service integration bus on which the target destination is assigned. The classification applies to the bus named by this property, or to any bus if you do not specify this property. The destinations to which the classification applies depends on your use of the destination property.

**destination**
The name of the target bus destination to which the message has been delivered. This is the name of a queue or topic space. The classification applies to the destination named by this property, or any destination if you do not specify this property. The service integration buses to which the classification applies depends on your use of the bus property.

**discriminator**
The property applies only when the destination property names a topic space. This discriminator value is then an XPath expression that selects one or more topics within the topic space.

**description**
Although the description field is optional, you should use it on all the sib_classification_info elements. The description is displayed when you monitor the transaction classes in the MVS console.

Each sib_classification_info element can contain one or more of these properties as needed to classify the work for a message. A sib_classification_info element cannot contain more than one instance of each property.

If a message matches several sib_classification_info elements, the element that appears first is used. For example, consider the following specifications:

```
<sib_classification_info bus="MyBus" transaction_class="a" />
<sib_classification_info destination="MyDest" transaction_class="b" />
```

A message that arrives at destination MyDest from the service integration bus MyBus is assigned the classification "a". A message that arrives at MyDest from another bus is assigned the classification "b".

## Mediation classification

The SibClassification element with the attribute type="destinationmediation" defines the section of the document that applies to work for mediations assigned to destinations on a service integration bus. An example of this element follows:

```
    <SibClassification type="destinationmediation"
                       schema_version="1.0"
                       default_transaction_class="b">
```

Each SibClassification element can contain one or more sib_classification_info elements. An example of a classification entry element follows:

```
<sib_classification_info
                     transaction_class="e"
                     selector="user.Location=&apos;East&apos;"
                     destination="themoon"
                     discriminator="sides/dark"
                     description="n" />
```

**selector**
Use the selector attribute of the sib_classification_info element to assign a transaction class to a mediation that has a selector clause in its deployment descriptor. Use the following rules when defining your sib_classification_info elements:

- The value of the selector attribute must match exactly to the selector clause in the mediation deployment descriptor. It is an SQL expression that selects a message according to the values of the message properties. The syntax is that of a message selector in the JMS 1.1 specification, but it can operate on SIMessage messages (more than JMS messages). The syntax can select on system properties (including JMS headers, JMSX properties, and JMS_IBM_properties) and user properties (which must be prefixed by ".user" - for example, for the user property "Location", the selector would specify "user.Location" as shown in the preceding example). For more information, see Working with the message properties.
- The value of the selector attribute must have the correct syntax for an XML document. You must replace the < and > symbols with the entity references `&lt;` and `&gt;`, respectively. Similarly, if you use an apostrophe or quotation mark, use the `&apos;` and `&quot;` entity references.

**bus** The name of the service integration bus within which the mediation is running.

**destination**
The name of a bus destination to which the message has been delivered. This is the name of a queue or topic space (on the service integration bus named by the bus property).

**discriminator**
The property applies only when the destination property names a topic space. This discriminator value is then an XPath expression that selects one or more topics within the topic space.

**description**
Although the description field is optional, you should use it on all the sib_classification_info elements. The description is displayed when you monitor the transaction classes in the MVS console.

Each sib_classification_info element can contain one or more of these properties as needed to classify the work for a message. A sib_classification_info element cannot contain more than one instance of each property.

If a message matches several sib_classification_info elements, the element that appears first is used. For example, consider the following specifications:

```
<sib_classification_info transaction_class="e" destination="themoon" description="n" />
<sib_classification_info transaction_class="f" description="n" />
```

A message that arrives at the mediated destination themoon is assigned the classification “e”. A message that arrives at another mediated destination is assigned the classification “f”.

If a message does not match any sib_classification_info element, the message is assigned the default classification from the enclosing SibClassification element. If no SibClassification elements are defined, all work receives a built-in default classification with the value “SIBUS”. You must perform z/OS Workload Manager actions that are required to use the TCLASS value “SIBUS”, as described in Classifying z/OS workload.

## Data access resources

### EJB Container tuning

If you use applications that affect the size of the EJB Container Cache, it is possible that the performance of your applications can be impacted by an incorrect size setting. Monitoring Tivoli Performance Viewer (TPV) is a great way to diagnose if the EJB Container Cache size setting is tuned correctly for your application.

If the application has filled the cache causing evictions to occur, TPV will show a very high rate of ejbStores() being called and probably a lower than expected CPU utilization on the application server machine.

All applications using enterprise beans should have this setting adjusted from the default if the following formula works out to more than 2000.

```
EJB_Cache_Size = (Largest number of Option B or C Entity Beans enlisted in a
transaction * maximum number of concurrent transactions) +
(Largest number of unique Option A Entity Beans expected to be accessed during
typical application workload) +
(Number of stateful Session Beans active during typical workload) +
(Number of stateless SessionBean types used during typical workload)

Where:
Option B and C Entity Beans are only held in the EJB cache during the lifetime
of the transaction they are enlisted in. Therefore, the first term in the formula
computes the average EJB cache requirements for these types of beans.

Option A Entity Beans are held in the EJB cache indefinitely, and are only removed
 from the cache if there start to become more beans in the cache than the cache
size has been set to.

Stateful Session Beans are held in the EJB cache until they are removed by the
application, or their session timeout value is reached.

Only a single stateless Session Bean instance for each EJB type is held in the
cache during the time any methods are being executed on that stateless Session
Bean. If two or more methods are being executed simultaneously on the same
stateless Session Bean type, each method executes on its own bean instance, but
only one cache location is used for all of these instances.
```

This calculates the upper bound on the maximum possible number of enterprise beans active at one time inside the application server. Because the EJB Containers cache is built to contain all these beans for performance optimizations, best performance can be achieved by setting this cache size to be larger than the number resulting from the calculation above.

```
<tuning parameter>
This setting can be found under Servers > Application Servers > serverName >
         EJB Container > EJB Cache Settings
```

Also while adjusting the EJB Cache Size, the EJB Container management thread parameter can be tuned to meet the needs of the application. The management thread is controlled through the Clean Up Interval setting. This setting controls how frequently a daemon thread inside of WebSphere Application Server wakes up and attempts to remove bean instances from the cache that have not been used recently, attempting to keep the number of bean instances at or below the cache size. This allows the EJB container to place and look up items in the cache as quickly as possible. It normally is best to leave this interval set to the default, however, in some cases, it may be worthwhile to see if there is a benefit to reducing this interval.

### EJB Container Pool Size

If the application is using the majority of the instances in the pool, TPV indicates this. When this occurs, then the size of those bean pools that are being exhausted should be increased. This can be done by adding the following parameter in the JVM's custom properties tag .

```
-Dcom.ibm.websphere.ejbcontainer.poolSize=<application_name>#<module_name>#
<enterprisebean_name>=<minSize>,<maxSize>

where:
 <application_name> is the J2EE application name as defined in the application
archive (.ear) file deployment descriptor, for the bean whose pool size is being
set

<module_name> is the .jar file name of the EJB module, for the bean whose pool size
 is being set,

<bean_name> is the J2EE Enterprise Bean name as defined in the EJB module
deployment descriptor, for the bean whose pool size is being set

<minSize> is the number of bean instances the container maintains in the pool,
 irrespective of how long the beans have been in the pool (beans greater than this
 number are cleared from the pool over time to optimize memory usage)

<maxSize> is the number of bean instances in the pool where no more bean instances
 are placed in the pool after they are used (that is, once the pool is at this
 size, any additional beans are discarded rather than added into the pool --
this ensures the number of beans in the pool has an upper limit so memory usage
does not grow in an unbounded fashion).

To keep the number of instances in the pool at a fixed size, minSize and maxSize
can be set to the same number. Note that there is a separate instance pool for
every EJB type running in the application server, and that every pool starts out
with no instances in it - that is, the number of instances grows as beans are
used and then placed in the pool. When a bean instance is needed by the container
and no beans are available in the pool, the container creates a new bean
instance, uses it, then places that instance in the pool (unless there are
already maxSize instances in the pool).

For example, the statement
-Dcom.ibm.websphere.ejbcontainer.poolSize=ivtApp#ivtEJB.jar#ivtEJBObject=125,1327

would set a minSize of 125 and a maxSize of 1327 on the bean named "ivtEJBObject"
within the ivtEJB.jar file, in the application "ivtApp".
```

Where ivtApp is replaced by the actual application name, ivtEJB.jar is replaced by the jar containing the bean that needs to have its pool size increased, and ivtEJBObject is the bean name of the enterprise bean whose pool size should be increased. The 125,1327 is the minimum and maximum number of beans that will be held in the pool. These should be set so no more evictions occur from the pool and in most cases should be set equal if memory is plentiful because no growth and shrinkage of the pool will occur.

### EJB Container Primary Key Mutation

Application developers and administrators should have a good idea of how their application handles the creation of primary key objects for use by container-managed persistence (CMP) beans and bean-managed persistence (BMP) beans inside of WebSphere Application Server. The IBM EJB Container uses the primary key of an Entity bean as an identifier inside of many internal data structures to optimize performance. However, the EJB Container must copy these primary key objects upon the first access to the bean to ensure that the objects stored in the internal caches are separate from the ones used in an application, in case the application changes or mutates the primary key, to keep the internal structures consistent.

If the application does not mutate any of the primary keys used to create and access entity beans after they are created, then a special flag can be used that allows the EJB Container to skip the copy of the primary key object, thus saving CPU cycles and increasing performance. This mechanism can be enabled *at your own risk* by adding the following –D property to the JVM custom property field.

```
<tuning parameter>
-Dcom.ibm.websphere.ejbcontainer.noPrimaryKeyMutation=true
```

The performance benefit of this optimization depends on the application. If the application uses primitive types for enterprise beans' primary keys there will be no gain because these objects are already immutable and the copy mechanism takes this into account. If, however, the application uses many complex primary keys (that is, And object for a primary key or multiple fields) then this parameter can yield significant improvements.

### Persistence Manager Deferred Insert on EJB Create

The IBM Persistence manager is used by the EJB Container to persist data to the database from CMP entity beans. When creating entity beans by calling the ejbCreate() method, by default the Persistence manager immediately inserts the empty row with only the primary key in the database. In most cases applications, after creating the bean, modify fields in the bean created or in other beans inside of the same transaction. If the user wishes to postpone the insert into the database until the end of the transaction, so that it will eliminate one trip to the database, they may set this –D flag inside of the JVM custom properties field. The data will still be inserted into the database and consistency will be maintained.

```
<tuning parameter>
-Dcom.ibm.ws.pm.deferredcreate=true
```

The performance benefit of this optimization depends on the application. If the EJB applications transactions are very insert intensive the application could benefit largely from this optimization. If the application performs very few inserts then the benefit of this optimization will be much less.

### Persistence Manager Database Batch Update on EJB Update

When an EJB application accesses multiple CMP beans inside of a single transaction, depending on the operations performed on the beans (updates, inserts, reads), the number of operations issued to the database will correspond directly to the operations performed on the CMP beans. If the database system you are using supports batching of update statements you can enable this flag and gain a performance boost on all interactions with the database that involve more than two updates in a single transaction. This flag will let the persistence manager add all the update statements into one single batch statement which will then be issued to the database. This saves round trips to the database, thus increasing performance. If the user knows their application exhibits the behavior of updating multiple CMP beans in a single transaction and the database supports batch updates they may set this –D flag inside of the JVM custom properties field.

```
<tuning parameter>
-Dcom.ibm.ws.pm.batch=true
```

The performance benefit of this optimization depends on the application. If the application never or infrequently updates CMP beans or only updates a single bean per transaction there will be no performance gain. If the application updates multiple beans per transaction then this parameter will benefit your applications performance.

The following table lists which backend databases support batch update.

Table 3.

| Database | Supports Batch update | Supports Batch update with Optimistic Concurrency Control |
|---|---|---|
| DB2 | yes | no |
| Oracle | yes | no |
| DB2 Universal Driver | yes | yes |
| Informix | yes | yes |
| SQLServer | yes | yes |
| Cloudscape | yes | yes |

**Note:** Batch update with OCC cannot be performed for databases that do not support it, even if specified by the access intent.

### Persistence Manager cache Tuning

Persistence Manager has two different types of caching mechanisms available: *legacy cache* and *two-level cache*. Normally two-level cache performs better than legacy cache because of optimizations in this mode. The default is legacy cache, although two-level cache is recommended. Set this configuration through the system property

```
com.ibm.ws.pm.useLegacyCache=false
```

### Persistence Manager Partial Updates Tuning

The partial updates feature enhances the performance of applications with enterprise beans in certain scenarios. Persistence Manager has two different types of caching mechanisms available, legacy cache and two-level cache. Normally, two-level cache performs better than legacy cache because of the optimizations in this mode. In certain applications where you need to perform both batch updates and partial updates, you must configure the following system properties to gain the benefits of both.

```
'com.ibm.ws.pm.grouppartialupdate=true' and 'com.ibm.ws.pm.batch=true'
```

## Database performance tuning

Database performance tuning can dramatically affect the throughput of your application. For example, if your application requires high concurrency (multiple, simultaneous interactions with backend data), an improperly tuned database can result in a bottleneck. Database access threads accumulate in a backlog when the database is not configured to accept a sufficient number of incoming requests.

Tuning parameters vary according to the type of database you are using. “DB2 tuning tips for z/OS” on page 3 are provided for your convenience. Because DB2 is not a WebSphere Application Server product and can change, consider these descriptions as suggestions.

***DB2 tuning parameters:*** DB2 has many parameters that you can configure to optimize database performance. For complete DB2 tuning information, refer to the *DB2 UDB Administration Guide: Performance* document.

**DB2 logging**

- **Description:** DB2 has corresponding log files for each database that provides services to administrators, including viewing database access and the number of connections. For systems with multiple hard disk drives, you can gain large performance improvements by setting the log files for each database on a different hard drive from the database files.
- **How to view or set:** At a DB2 command prompt, issue the command: `db2 update db cfg for [database_name] using newlogpath [fully_qualified_path]`.

- **Default value:** Logs reside on the same disk as the database.
- **Recommended value:** Use a separate high-speed drive, preferably performance enhanced through a redundant array of independent disk (RAID) configuration.

### DB2 configuration advisor

Located in the DB2 Control Center, this advisor calculates and displays recommended values for the DB2 buffer pool size, the database, and the database manager configuration parameters, with the option of applying these values. See more information about the advisor in the online help facility within the Control Center.

### Number of connections to DB2 - MaxAppls and MaxAgents

When configuring the data source settings for the databases, confirm the DB2 MaxAppls setting is greater than the maximum number of connections for the data source. If you are planning to establish clones, set the MaxAppls value as the maximum number of connections multiplied by the number of clones. The same relationship applies to the session manager number of connections. The MaxAppls setting must be equal to or greater than the number of connections. If you are using the same database for session and data sources, set the MaxAppls value as the sum of the number of connection settings for the session manager and the data sources.

For example, MaxAppls = (number of connections set for the data source + number of connections in the session manager) multiplied by the number of clones.

After calculating the MaxAppls settings for the WebSphere Application Server database and each of the application databases, verify that the MaxAgents setting for DB2 is equal to or greater than the sum of all of the MaxAppls values. For example, MaxAgents = sum of MaxAppls for all databases.

### DB2 buffpage

- **Description:** Improves database system performance. Buffpage is a database configuration parameter. A buffer pool is a memory storage area where database pages containing table rows or index entries are temporarily read and changed. Data is accessed much faster from memory than from disk.
- **How to view or set:** To view the current value of buffpage for database *x*, issue the DB2 command `get db cfg for x` and look for the value **BUFFPAGE**. To set **BUFFPAGE** to a value of *n*, issue the DB2 command `update db cfg for x` using **BUFFPAGE *n*** and set **NPAGES** to -1 as follows:

```
db2   <-- go to DB2 command mode, otherwise the following "select" does not work as is
    connect to x    <-- (where x is the particular DB2 database name)
    select * from syscat.bufferpools
       (and note the name of the default, perhaps: IBMDEFAULTBP)
       (if NPAGES is already -1, there is no need to issue following command)
    alter bufferpool IBMDEFAULTBP size -1
    (re-issue the above "select" and NPAGES now equals -1)
```

  You can collect a snapshot of the database while the application is running and calculate the buffer pool hit ratio as follows:

  1. Collect the snapshot:
     a. Issue the **update monitor switches using bufferpool on** command.
     b. Make sure that bufferpool monitoring is on by issuing the **get monitor switches** command.
     c. Clear the monitor counters with the **reset monitor all** command.
  2. Run the application.
  3. Issue the **get snapshot for all databases** command before all applications disconnect from the database, otherwise statistics are lost.
  4. Issue the **update monitor switches using bufferpool off** command.
  5. Calculate the hit ratio by looking at the following database snapshot statistics:
     – Buffer pool data logical reads
     – Buffer pool data physical reads
     – Buffer pool index logical reads
     – Buffer pool index physical reads

- **Default value:** 250
- **Recommended value:** Continue increasing the value until the snapshot shows a satisfactory hit rate.

The buffer pool hit ratio indicates the percentage of time that the database manager did not need to load a page from disk to service a page request. That is, the page is already in the buffer pool. The greater the buffer pool hit ratio, the lower the frequency of disk input and output. Calculate the buffer pool hit ratio as follows:
- P = buffer pool data physical reads + buffer pool index physical reads
- L = buffer pool data logical reads + buffer pool index logical reads
- Hit ratio = (1-(P/L)) * 100%

**DB2 query optimization level**
- **Description:** Sets the amount of work and resources that DB2 puts into optimizing the access plan. When a database query runs in DB2, various methods are used to calculate the most efficient access plan. The range is from 0 to 9. An optimization level of 9 causes DB2 to devote a lot of time and all of its available statistics to optimizing the access plan.
- **How to view or set:** The optimization level is set on individual databases and can be set with either the command line or with the DB2 Control Center. Static SQL statements use the optimization level that is specified on the **prep** and **bind** commands. If the optimization level is not specified, DB2 uses the default optimization as specified by the dft_queryopt setting. Dynamic SQL statements use the optimization class that is specified by the current query optimization special register, which is set using the SQL Set statement. For example, the following statement sets the optimization class to 1:

  ```
  Set current query optimization = 1
  ```

  If the current query optimization register is not set, dynamic statements are bound using the default query optimization class.
- **Default value:** 5
- **Recommended value:** Set the optimization level for the needs of the application. Use high levels only when there are very complicated queries.

**DB2 reorgchk**
- **Description:** Obtains the current statistics for data and rebinding. Use this parameter because SQL statement performance can deteriorate after many updates, deletes or inserts.
- **How to view or set:** Use the DB2 **reorgchk update statistics on table all** command to perform the **runstats** operation on all user and system tables for the database to which you are currently connected. Rebind packages using the **bind** command. If statistics are available, issue the **db2 -v ″select tbname, nleaf, nlevels, stats_time from sysibm.sysindexes″** command on DB2 CLP. If no statistic updates exist, nleaf and nlevels are -1, and stats_time has an empty entry (for example: ″-″). If the runstats command was previously run, the real-time stamp from completion of the runstats operation also displays under stats_time. If you think the time shown for the previous runstats operation is too old, run the runstats command again.
- **Default value:** None
- **Recommended value:** None

**DB2 locktimeout**
- **Description:** Specifies the number of seconds that an application waits to obtain a lock. Setting this property helps avoid global deadlocks for applications.
- **How to view or set:** To view the current value of the lock timeout property for database *xxxxxx*, issue the DB2 **get db cfg for *xxxxxx*** command and look for the value, LOCKTIMEOUT. To set LOCKTIMEOUT to a value of *n*, issue the DB2 **update db cfg for *xxxxxx*** command using **LOCKTIMEOUT *n***, where *xxxxxx* is the name of the application database and *n* is a value between 0 and 30 000 inclusive.
- **Default value:** -1, meaning lock timeout detection is turned off. In this situation, an application waits for a lock if one is not available at the time of the request, until either of the following events occurs:
  - The lock is granted
  - A deadlock occurs

- **Recommended value:** If your database access pattern tends toward a majority of writes, set this value so that it gives you early warning when a timeout occurs. A setting of 30 seconds suits this purpose. If your pattern tends toward a majority of reads, either accept the default lock timeout value, or set the property to a value greater than 30 seconds.

**DB2 maxlocks**
- **Description:** Specifies the percentage of the lock list that is reached when the database manager performs escalation, from row to table, for the locks held by the application. Although the escalation process does not take much time, locking entire tables versus individual rows decreases concurrency, and potentially decreases overall database performance for subsequent attempts to access the affected tables.
- **How to view or set:** To view the current value of the maxlocks property for database *xxxxxx*, issue the DB2 **get db cfg for *xxxxxx*** command and look for the MAXLOCKS value. To set MAXLOCKS to a value of *n*, issue the DB2 **update db cfg for *xxxxxx*** command using **MAXLOCKS *n***, where *xxxxxx* is the name of the application database and *n* is a value between 1 and 100 inclusive.
- **Default value:** Refer to the current database information for property default values per operating system.
- **Recommended value:** If lock escalations are causing performance concerns, you might need to increase the value of this parameter or the locklist parameter, which is described in the following paragraph. You can use the database system monitor to determine if lock escalations are occurring.

**DB2 locklist**
- **Description:** Specifies the amount of storage that is allocated to the lock list.
- **How to view or set:** To view the current value of the locklist property for database *xxxxxx*, issue the DB2 **get db cfg for *xxxxxx*** command and look for the LOCKLIST value . To set LOCKLIST to a value of *n*, issue the DB2 **update db cfg for *xxxxxx*** command using **LOCKLIST *n***, where *xxxxxx* is the name of the application database and *n* is a value between 4 and 60 000 inclusive.
- **Default value:** Refer to the current database information for property default values per operating system.
- **Recommended value:** If lock escalations are causing performance concerns, you might need to increase the value of this parameter or the maxlocks parameter, which is described in the previous paragraph. You can use the database system monitor to determine if lock escalations are occurring. Refer to the *DB2 Administration Guide: Performance* document for more details.

## Tuning parameters for data access resources

For better application performance, you can tune some data access resources through the WebSphere Application Server administrative console.

Tune these properties of data sources and connection pools to optimize the performance of transactions between your application and datastore. See the *Administering applications and their environment* PDF for more information.

### Data source tuning

To view the administrative console page where you configure the following properties, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data sources** > *data_source* > **WebSphere Application Server connection properties**.

**Enable JMS one phase optimization support**

If your application does not use JMS messaging, **do not** select this option. Activating this support enables the Java Message Service (JMS) to get optimized connections from the data source. Activating this support also *prevents* JDBC applications from obtaining connections from the data source. For further explanation of JMS one phase support, refer to the article entitled ″Sharing connections to benefit from one phase commit optimization″ in this information center.

**Statement cache size**

Specifies the number of statements that can be cached per connection.

The WebSphere Application Server data source optimizes the processing of *prepared statements* and *callable statements* by caching those statements that are not being used in an active connection. Both statement types help reduce overhead for transactions with backend data.

- A prepared statement is a precompiled SQL statement that is stored in a `PreparedStatement` object. Application Server uses this object to run the SQL statement multiple times, as required by your application run time, with values that are determined by the run time.
- A callable statement is an SQL statement that contains a call to a stored procedure, which is a series of precompiled statements that perform a task and return a result. The statement is stored in the `CallableStatement` object. Application Server uses this object to run a stored procedure multiple times, as required by your application run time, with values that are determined by the run time.

In general, the more statements your application has, the larger the cache should be. **Be aware**, however, that specifying a larger statement cache size than needed wastes application memory and *does not* improve performance.

Determine the value for your cache size by adding the number of uniquely prepared statements and callable statements (as determined by the SQL string, concurrency, and the scroll type) for each application that uses this data source on a particular server. This value is the maximum number of possible statements that can be cached on a given connection over the life of the server. See the *Administering applications and their environment* PDF for more information.

Default: For most databases the default is 10. Zero means there is no cache statement.

## Connection pool tuning

To view the administrative console page where you configure the following properties, click **Resources** > **JDBC Providers** > *JDBC_provider* > **Data sources** > *data_source* > **Connection pool settings**.

**Maximum connections**

Specifies the maximum number of physical connections that can be created in this pool. These are the physical connections to the backend datastore. When this number is reached, no new physical connections are created; requestors must wait until a physical connection that is currently in use is returned to the pool.

For optimal performance, set the value for the connection pool lower than the value for the Web container threadpool size. Lower settings, such as 10 to 30 connections, might perform better than higher settings, such as 100. See the *Administering applications and their environment* PDF for more information.

Default: 10

**Minimum connections**

Specifies the minimum number of physical connections to maintain. Until this number is exceeded, the pool maintenance thread does not discard physical connections.

If you set this property for a higher number of connections than your application ultimately uses at run time, you do not waste application resources. WebSphere Application Server does not create additional connections to achieve your minimum setting. Of course, if your application requires more connections than the value you set for this property, application performance diminishes as connection requests wait for fulfillment. See the *Administering applications and their environment* PDF for more information.

Default: 1

## Messaging resources

### Tuning JMS destinations

Use this task to configure the properties of a JMS destination to optimize performance of applications that use the WebSphere Application Version 5 default messaging provider or WebSphere MQ as a JMS provider.

To optimize performance, configure destination properties to best fit your applications. You should also consider queue attributes of the JMS server that are associated with the queue name. For more information, see the following topics:

- "Performance considerations for WebSphere Version 5 queue destinations"
- "Performance considerations for WebSphere Version 5 topic destinations"
- "Performance considerations for WebSphere MQ queue destinations" on page 121
- "Performance considerations for WebSphere MQ topic destinations" on page 121

***Performance considerations for WebSphere Version 5 queue destinations:***

To optimize performance, configure the queue destination properties to best fit your applications. For example, setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of messages that can be queued.

To ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for your application usage.

For queue destinations configured on a WebSphere Application Server version 5 node, you should also consider queue attributes of the internal JMS server that are associated with the queue name. Inappropriate queue attributes can reduce the performance of WebSphere operations.

**BOQNAME**
: The excessive backout requeue name. This can be set to a local queue name that can hold the messages which were rolled back by the WebSphere applications. This queue name can be a system dead letter queue.

**BOTHRESH**
: The backout threshold and can be set to a number once the threshold is reached, the message will be moved to the queue name specified in BOQNAME.

For more information about using these properties, see:

- "Handling poison messages" in the *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

***Performance considerations for WebSphere Version 5 topic destinations:***

To optimize performance, configure the JMS destination properties to best fit your applications.

For example, setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of messages that can be queued.

For JMS destinations configured on a WebSphere Application Server version 5 node, ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for your application usage.

- Ensure the queue attribute, INDXTYPE is set to MSGID for the following system queues:
  - SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
  - SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- Ensure the queue attribute, INDXTYPE is set to CORRELID for the following system queues:

– SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
– SYSTEM.JMS.D.SUBSCRIBER.QUEUE

For more information about using these properties, see:
- The *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

***Performance considerations for WebSphere MQ queue destinations:***

To optimize performance, configure the queue destination properties to best fit your message-driven beans or other applications that use the queue destinations.

For example:
- When MDB applications are configured to WebSphere MQ queues on z/OS, the INDEX by MSGID is very important.
- Setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of messages that can be queued.

To ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for use by your message-driven beans or other applications that use the queue.

You should also consider queue attributes of the internal JMS server that are associated with the queue name. Inappropriate queue attributes can reduce the performance of WebSphere operations.

You should also consider the queue attributes associated with the queue name you created with WebSphere MQ. Inappropriate queue attributes can reduce the performance of WebSphere operations. You can use WebSphere MQ commands to change queue attributes for the queue name.

**BOQNAME**
The excessive backout requeue name. This can be set to a local queue name that can hold the messages which were rolled back by the WebSphere applications. This queue name can be a system dead letter queue.

**BOTHRESH**
The backout threshold and can be set to a number once the threshold is reached, the message will be moved to the queue name specified in BOQNAME.

**INDXTYPE**
Set this to MSGID. This causes an index of message identifiers to be maintained, which can improve WebSphere MQ retrieval of messages.

**DEFSOPT**
Set this to SHARED (for shared input from the queue).

**SHARE**
This must be specified (so that multiple applications can get messages from this queue).

For more information about using these properties, see:
- For BOQNAME and BOTHRESH, see “Handling poison messages” in the *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

***Performance considerations for WebSphere MQ topic destinations:***

To optimize performance, configure the topic destination properties to best fit your applications. For example, setting the Expiry property to SPECIFIED and the Specified Expiry property to 30000 milliseconds for the expiry timeout, reduces the number of messages that can be queued.

To ensure that there are enough underlying WebSphere MQ resources available for the queue, you must ensure that you configure the queue destination properties adequately for your application usage.

- Ensure the queue attribute, INDXTYPE is set to MSGID for the following system queues:
  - SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
  - SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- Ensure the queue attribute, INDXTYPE is set to CORRELID for the following system queues:
  - SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
  - SYSTEM.JMS.D.SUBSCRIBER.QUEUE

For more information about using these properties, see:

- The *WebSphere MQ Using Java* book
- The *WebSphere MQ Script (MQSC) Command Reference* book

## Security

### SSL considerations for WebSphere Application Server administrators

Consider the following items regarding the Secure Sockets Layer.

The Resource Access Control Facility (RACF) customization jobs create an SSL keyring owned by the WebSphere Application Server for z/OS administrator containing the digital certificate needed to communicate with WebSphere Application Server. However, additional customization is required for administration by other MVS user IDs.

Note that the MVS user ID in the description below is the MVS user ID under which the **wsadmin** process is running, not the user ID specified in the wsadmin request.

In the example below:

- `yyyyy` is the user ID of the new WebSphere Application Server for z/OS administrator
- `xxxxx` is the name of the keyring that is specified in `soap.client.props` in the *profile_root*`/properties` directory.
- `zzzzz` is the label name used in the BBOSBRAK jobs to specify which certificate authority certificate was used to generate server keys

1. If the new administrator is not a member of the WebSphere Application Server for z/OS administrative group, make sure that the new user ID has access to the appropriate RACF keyrings and digital certificates. For example:
   ```
   PERMIT IRR.DIGTCERT.LIST CLASS(FACILITY) ID(yyyyy) ACC(READ)
   PERMIT IRR.DIGTCERT.LISTRING CLASS(FACILITY) ID(yyyyy) ACC(READ)
   ```
2. Use the setup completed by the customization jobs as a model for the additional steps. This information is in the BBOCBRAK member of the <HLQ>.DATA data set generated during the customization process. The BBOCBRAK job contains the set of RACF commands that were used:
   ```
     /* Generating SSL keyrings for WebSphere administrator                    */
   RACDCERT ADDRING(xxxxx) ID( yyyyyy )

    /* Connect WebSphere Application Server CA Certificates to Servers keyring    */
   "RACDCERT ID(yyyyy) CONNECT (RING(xxxxxx) LABEL('zzzzzzz')   CERTAUTH"

   SETROPTS RACLIST(FACILITY) REFRESH"
   ```

### Setting up SSL connections for Java clients

Follow these steps to configure SSL for use between Java clients running on a workstation and the WebSphere Application Server for z/OS Java 2 Platform, Enterprise Edition (J2EE) server.

1. Determine what SSL repertoire the server is using. For example: `WASKeyring`.
2. Determine the user ID the server is running. For example: `CBSYMSR1`.

3. Export the certificate authority from RACF. For example:

```
RACDCERT CERTAUTH EXPORT(LABEL('WebSphereCA')) DSN('IBMUSER.WAS.CA') FORMAT(CERTDER)
```

4. Move the file to the workstation. (Note that the FTP transfer must use binary.) For example: `\tmp` directory
5. Add the digital certificate to the TrustStore used by the client. For example, for the `DummyClientTrustFile.jks` file, type:

```
keytool -import -file \tmp\IBMUSER.WAS.CA -keystore DummyClientTrustFile.jks]
```

## Security cache properties

The following Java virtual machine (JVM) security cache custom properties determine whether the authentication cache is enabled or disabled. If the authentication cache is enabled, as recommended, these custom properties specify the initial size of the primary and secondary hash table caches, which affect the frequency of rehashing and the distribution of the hash algorithms.

**Important:** The com.ibm.websphere.security.util.tokenCacheSize and com.ibm.websphere.security.util.LTPAValidationCacheSize properties were replaced with the com.ibm.websphere.security.util.authCacheSize property.

You can specify these system properties by completing the following steps:
1. Click **Servers > Application servers >** *server_name*.
2. Click **Java and Process Management > Process Definition**.
3. Under Additional properties, click **Java Virtual Machine**.
4. Specify the property name and its value in the Generic JVM arguments field. You can specify multiple property name and value pairs delimited by a space.

WebSphere Application Server includes the following security cache custom properties:

**com.ibm.websphere.security.util.authCacheEnabled**

Specifies whether to disable the authentication cache. It is recommended that you leave the authentication cache enabled for performance reasons. However, you can disable the authentication cache for debug or measurement purposes.

| | |
|---|---|
| Default: | True |

**com.ibm.websphere.security.util.authCacheSize**

Specifies the initial size of the primary and secondary hash table caches. A higher number of available hash values might decrease the occurrence of hash collisions. A hash collision results in a linear search for the hash bucket, which might decrease the retrieval time. If several entries compose a hash table cache, you create a table with a larger capacity that supports more efficient hash entries instead of allowing automatic rehashing determine the growth of the table. Rehashing causes every entry to move each time.

| | |
|---|---|
| Default: | 200 |
| Type: | Integer |

## Tuning, hardening, and maintaining

After you have installed WebSphere Application Server, there are several considerations for tuning, strengthening, and maintaining your security configuration.

The following topics are covered in this section:
- Tuning security configurations
- Hardening security configurations
- Changing keys and passwords

- Securing passwords in files

***Tuning security configurations:***
Performance issues typically involve trade-offs between function and speed. Usually, the more function and the more processing that are involved, the slower the performance. Consider what type of security is necessary and what you can disable in your environment. For example, if your application servers are running in a Virtual Private Network (VPN), consider whether you can disable Secure Sockets Layer (SSL). If you have a lot of users, can they be mapped to groups and then associated to your Java 2 Platform, Enterprise Edition (J2EE) roles? These questions are things to consider when designing your security infrastructure.

- Consider the following recommendations for tuning general security.
  - Consider disabling Java 2 security manager if you know exactly what code is put onto your server and you do not need to protect process resources. Remember that in doing so, you put your local resources at some risk.
  - Consider propagating new security settings to all nodes before restarting the deployment manager and node agents, to change the new security policy.

    If your security configurations are not consistent across all servers, you get access denied errors. Therefore, you must propagate new security settings when enabling or disabling administrative security.

    Configuration changes are generally propagated using configuration synchronization. If auto-synchronization is enabled, you can wait for the automatic synchronization interval to pass, or you can force synchronization before the synchronization interval expires. If you are using manual synchronization, you must synchronize all the nodes.

    If the cell is in a configuration state and the security policy is mixed with nodes that have security enabled and disabled, you can use the syncNode utility to synchronize the nodes where the new settings are not propagated.

    For more detailed information about enabling security in a distributed environment, see Enabling security for the realm.
  - Consider increasing the cache and token timeout if you feel your environment is secure enough. By increasing these values, you have to re-authenticate less often. This action supports subsequent requests to reuse the credentials that already are created. The downside of increasing the token timeout is the exposure of having a token hacked and providing the hacker more time to hack into the system before the token expires. You can use security cache properties to determine the initial size of the primary and secondary hashtable caches, which affect the frequency of rehashing and the distribution of the hash algorithms.
  - Consider changing your administrative connector from Simple Object Access Protocol (SOAP) to Remote Method Invocation (RMI) because RMI uses stateful connections while SOAP is completely stateless. Run a benchmark to determine if the performance is improved in your environment.
  - Use the wsadmin script to complete the access IDs for all the users and groups to speed up the application startup. Complete this action if applications contain many users or groups, or if applications are stopped and started frequently. WebSphere Application Server maps user and group names to unique access IDs in the authorization table. The exact format of the access ID depends on the repository. The access ID can only be determined during and after application deployment. Authorization tables created during assembly time do not have the proper access IDs. See Commands for the AdminApp object for more information about how to update access IDs.
  - If using SSL, enable the SSL session tracking mechanism option as described in the article, Session management settings.
- Consider the following steps to tune Common Secure Interoperability version 2 (CSIv2).
  - Consider using Secure Sockets Layer (SSL) client certificates instead of a user ID and password to authenticate Java clients. Because you are already making the SSL connection, using mutual authentication adds little overhead while it removes the service context that contains the user ID and password completely.

- If you send a large amount of data that is not very security sensitive, reduce the strength of your ciphers. The more data you have to bulk encrypt and the stronger the cipher, the longer this action takes. If the data is not sensitive, do not waste your processing with 128-bit ciphers.
- Consider putting only an asterisk (*) in the trusted server ID list (meaning trust all servers) when you use identity assertion for downstream delegation. Use SSL mutual authentication between servers to provide this trust. Adding this extra step in the SSL handshake performs better than having to fully authenticate the upstream server and check the trusted list. When an asterisk (*) is used, the identity token is trusted. The SSL connection trusts the server through client certificate authentication.
- Ensure that stateful sessions are enabled for CSIv2. This is the default, but requires authentication only on the first request and on any subsequent token expirations.
- V6.0.x If you are communicating only with WebSphere Application Server Version 5 or higher servers, make the Active Authentication Protocol CSI, instead of CSI and SAS. This action removes an interceptor invocation for every request on both the client and server sides.

  **Important:** SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

- Consider the following steps to tune Lightweight Directory Access Protocol (LDAP) authentication.
  1. In the administration console, click **Security > Secure administration, applications, and infrastructure**.
  2. Under User account repository, click the **Available realm definitions** drop-down list, select**Standalone LDAP registry** and click **Configure**.
  3. Select the **Ignore case for authorization** option in the standalone LDAP registry configuration, when case-sensitivity is not important.
  4. Select the **Reuse connection** option.
  5. Use the cache features that your LDAP server supports.
  6. Choose either the IBM Tivoli Directory Server or SecureWay directory type, if you are using an IBM Tivoli Directory Server. The IBM Tivoli Directory Server yields improved performance because it is programmed to use the new group membership attributes to improve group membership searches. However, authorization must be case insensitive to use IBM Tivoli Directory Server.
  7. Choose either iPlanet Directory Server (also known as Sun ONE) or Netscape as the directory if you are an iPlanet Directory user. Using the iPlanet Directory Server directory can increase performance in group membership lookup. However, use **Role** only for group mechanisms.
- Consider the following steps to tune Web authentication.
  - Increase the cache and token timeout values if you feel your environment is secure enough. The Web authentication information is stored in these caches and as long as the authentication information is in the cache, the login module is not invoked to authenticate the user. This supports subsequent requests to reuse the credentials that are already created. A disadvantage of increasing the token timeout is the exposure of having a token stolen and providing the thief more time to hack into the system before the token expires.
  - Enable single sign-on (SSO). To configure SSO, click **Security > Secure administration, applications, and infrastructure**. Under Web security, click **Single sign-on (SSO)**.

    SSO is only available when you configure **LTPA** as the authentication mechanism in the Authentication mechanisms and expiration panel. Although you can select Simple WebSphere Authentication Mechanism (SWAM) as the authentication mechanism on the Authentication mechanisms and expiration panel, SWAM is deprecated in Version 6.1 and does not support SSO. When you select SSO, a single authentication to one application server is enough to make requests to multiple application servers in the same SSO domain. Some situations exist where SSO is not a desirable and you do not want to use it in those situations.
  - Disable or enabling the **Web Inbound Security Attribute Propagation** option on the Single sign-on (SSO) panel if the function is not required. In some cases, having the function enabled can improve performance. This improvement is most likely for higher volume cases where a considerable number

of user registry calls reduces performance. In other cases, having the feature disabled can improve performance. This improvement is most likely when the user registry calls do not take considerable resources.

- Consider the following steps to tune authorization.
  - Map your users to groups in the user registry. Associate the groups with your Java 2 Platform, Enterprise Edition (J2EE) roles. This association greatly improves performance when the number of users increases.
  - Judiciously assign method-permissions for enterprise beans. For example, you can use an asterisk (*) to indicate all the methods in the method-name element. When all the methods in enterprise beans require the same permission, use an asterisk (*) for the method-name to indicate all methods. This indication reduces the size of deployment descriptors and reduces the memory that is required to load the deployment descriptor. It also reduces the search time during method-permission match for the enterprise beans method.
  - Judiciously assign security-constraints for servlets. For example, you can use the `*.jsp` URL pattern to apply the same authentication data constraints to indicate all JavaServer Pages (JSP) files. For a given URL, the exact match in the deployment descriptor takes precedence over the longest path match. Use the `*.jsp`, `*.do`, `*.html` extension match if no exact matches exist and longest path matches exist for a given URL in the security constraints.

You always have a trade off between performance, feature, and security. Security typically adds more processing time to your requests, but for a good reason. Not all security features are required in your environment. When you decide to tune security, create a benchmark before making any change to ensure that the change is improving performance.

In a large scale deployment, performance is very important. Running benchmark measurements with different combinations of features can help you to determine the best performance versus the benefit of configuration for your environment. Continue to run benchmarks if anything changes in your environment, to help determine the impact of these changes.

*Security tuning tips:*

As a general rule, two things happen when you increase security: the cost per transaction increases and throughput decreases. Consider the following security information when you configure WebSphere Application Server.

**SAF class**

When a System Authorization Facility (SAF) (Resource Access Control Facility (RACF) or equivalent) class is active, the number of profiles in a class will affect the overall performance of the check. Placing these profiles in a (RACLISTed) memory table will improve the performance of the access checks. Audit controls on access checks also affect performance. Usually, you audit failures and not successes. Audit events are logged to Direct Access Storage Device (DASD) and will increase the overhead of the access check. Because all of the security authorization checks are done with SAF (RACF or equivalent), you can choose to enable and disable SAF classes to control security. A disabled class will cost a negligible amount of overhead.

**EJBROLEs on methods**

Use a minimum number of EJBROLEs on methods. If you are using EJBROLEs, specifying more roles on a method will lead to more access checks that need to be executed and a slower overall method dispatch. If you are not using EJBROLEs, do not activate the class.

**Java 2 Security**

If you do not need Java 2 security, disable it. For instructions on how to disable Java 2 security, refer to Protecting system resources and APIs (Java 2 security).

**Level of authorization**

Use the lowest level of authorization consistent with your security needs. You have the following options when dealing with authentication:

- **Local authentication:** Local authentication is the fastest type because it is highly optimized.
- **UserID and password authentication:** Authentication that utilizes a userID and password has a high first-call cost and a lower cost with each subsequent call.
- **Kerberos security authentication:** We have not adequately characterized the cost of kerberos security yet.
- **SSL security authentication:** SSL security is notorious in the industry for its performance overhead. Luckily, there is a lot of assists available from hardware to make this reasonable on z/OS.

**Level of encryption with SSL**

If using Secure Sockets Layer (SSL), select the lowest level of encryption consistent with your security requirements. WebSphere Application Server enables you to select which cipher suites you use. The cipher suites dictate the encryption strength of the connection. The higher the encryption strength, the greater the impact on performance.

**RACF tuning**

Follow these guidelines for RACF tuning:

- Use the RACLIST to place into memory those items that can improve performance. Specifically, ensure that you RACLIST (if used):
  - CBIND
  - EJBROLE
  - SERVER
  - STARTED
  - FACILITY
  - SURROGAT

  **Example:**

  ```
  RACLIST (CBIND, EJBROLE, SERVER, STARTED, FACILITY, SURROGAT)
  ```
- Using SSL has drawbacks. If you are a heavy SSL user, ensure that you have appropriate hardware, such as Peripheral Component Interconnect (PCI) crypto cards, to speed up the handshake process.
- Here’s how you define the BPX.SAFFASTPATH facility class profile. This profile allows you to bypass SAF calls which can be used to audit successful shared file system accesses.
  - Define the facility class profile to RACF.

    ```
    RDEFINE FACILITY BPX.SAFFASTPATH UACC(NONE)
    ```
  - Activate this change by doing one of the following:
    - re-IPL
    - invoke the SETOMVS or SET OMVS operator commands.

  **Note:** Do not use this option if you need to audit successful HFS accesses or if you use the IRRSXT00 exit to control HFS access.
- Use VLF caching of the UIDs and GIDs as shown in the example COFVLFxx parmlib member below:

  **Example:** sys1.parmlib(COFVLFxx):

```
********************************* Top of Data ********************.
.

CLASS NAME(IRRGMAP) EMAJ(GMAP)
CLASS NAME(IRRUMAP) EMAJ(UMAP)
CLASS NAME(IRRGTS) EMAJ(GTS)
CLASS NAME(IRRACEE) EMAJ(ACEE)
.
******************************** Bottom of Data ******************
```

To avoid a costly scan of the RACF databases, make sure all HFS files have valid GIDs and UIDs.

**Related tasks**

Protecting system resources and APIs (Java 2 security)
Java 2 security is a programming model that is very pervasive and has a huge impact on application development.

**Related information**

Session management settings
Use this page to manage HTTP session support. This support includes specifying a session tracking mechanism, setting maximum in-memory session count, controlling overflow, and configuring session timeout.

*Resource Access Control Facility Tips for customizing WebSphere Application Server:*

It is important to understand the security mechanisms used to protect the server resources using the CBIND, SERVER, and STARTED classes in RACF (or your security product). This paper describes these mechanisms along with some techniques for managing the security environment.

It is important to understand the security mechanisms used to protect the server resources using the `CBIND`, `SERVER`, and `STARTED` classes in RACF (or your security product). This article describes these mechanisms and some additional techniques for managing the security environment.

The first part of this article provides details about the RACF profiles used to protect the WebSphere servers and resources using the following classes:

- `CBIND`: Access to servers, and access to objects in the servers
- `SERVER`: Access to controller regions by servant regions
- `STARTED`: Associate user-ids and groups to started procedures (STCs)

The next part of the article describes adding the required RACF profiles and permissions for another server in your cell.

The last part of the article shows how you can define the minimal set of users, groups, and profiles for a testing environment (where security of individual servers is not the main focus or concern).

**RACF Profiles (`CBIND`, `SERVER`, and `STARTED`)**: Basic information about the RACF profiles used by WebSphere can be found in the ../ae/csec_safauth.dita. This section adds some additional details about the `CBIND`, `SERVER`, and `STARTED` class profiles.

**User IDs and Group IDs**: As part of the WebSphere customization dialogs, the `BBOCBRAJ` job generates RACF commands that then can be run with the `BBOWBRAK` job. Key:

```
CR = Controller Region
SR = Servant Region
CFG = Configuration (group)
server = server short name
cluster = generic server (short) name (also called cluster transition name)
```

First, six users and six groups are defined as follows, which are shown symbolically to help you understand how they are used in the various permissions later on:

```
&lt;CR_userid> &lt;CR_groupid>, &lt;CFG_groupid>
&lt;SR_userid> &lt;SR_groupid>, &lt;CFG_groupid>
&lt;demn_userid> &lt;demn_groupid>, &lt;CFG_groupid>
&lt;admin_userid> &lt;CFG_groupid>
&lt;client_userid> &lt;client_groupid>
&lt;ctracewtr_userid> &lt;ctracewtr_groupid>
```

Below are the various profiles used to protect the WebSphere servers and resources, along with the permissions and access levels.

**CBIND Class Profiles**: There are two formats and levels of CBIND class profiles for protecting access to application servers and objects in those servers:

```
CBIND Class profiles - access to generic servers
CB.BIND.<cluster> UACC(READ); PERMIT <CR_group> ACC(CONTROL)

CBIND Class profiles - access to objects in servers
CB.<cluster> UACC(READ) PERMIT <CR_group> ACC(CONTROL)
```

**SERVER Class Profiles**: There are currently two formats of the `SERVER` class profiles for protecting access to the server controller regions. You must define a single format SERVER profile, depending upon whether or not Dynamic Application Environment (DAE) support is enabled. This is done using the WLM DAE APAR OW54622, which is applicable to z/OS V1R2 or higher.

In the customization dialog, both formats are predefined, and one of these is actually required at runtime. The required format is determined dynamically by the WebSphere Application Server for z/OS Runtime based on the availability of Dynamic Application Environment (DAE) support.

- The following command provides access to controllers using static Application Environments (without the APAR support): `RDEFINE CB.&<server>.&<cluster> UACC(NONE); PERMIT &<SR_userid> ACC(READ)` For this example, `server = server name`, `cluster = cluster name` or `cluster transition name` if a cluster has not yet been created, and `SR = the MVS user ID of the Server Region`.
- The following command provides access to controllers using dynamic Application Environments (with the WLM DAE APAR support): `CB.&<server>.&<cluster>.&lt;cell> UACC(NONE); PERMIT &<SR_userid> ACC(READ)` For this example, `server = server name`, `cluster = cluster name` or `cluster transition name` if a cluster has not yet been created, `cell = cell short name`, and `SR = the MVS user ID of the Server Region`.

**STARTED Class Profiles**: There are two formats of `STARTED` class profiles used to assign user and group IDs to controller regions and other STCs based on whether the started task is started with the `MGCRE` interface or the address space create (`ASCRE`) interface used by Workload Manager (WLM) to start servant regions:

```
STARTED Class profiles - (MGCRE)
<&lt;CR_proc>.&lt;CR_jobname> STDATA(USER(CR_userid) GROUP(CFG_groupid))
&lt;demn_proc>.* STDATA(USER(demn_userid) GROUP(CFG_groupid))

STARTED Class profiles - (ASCRE)
&lt;SR_jobname>.&lt;SR_jobname> STDATA(USER(SR_userid) GROUP(CFG_groupid))

STARTED Class profiles for IJP - (MGCRE)
&lt;MQ_ssname>.* STDATA(USER(IJP_userid) GROUP(CFG_groupid))
```

**Generating new user IDs and Profiles for a new Server**: If you want to use unique user IDs for each new application server, you must define these users, groups, and profiles in the RACF database.

One technique is to edit a copy of the `BBOWBRAK` member in the ISPF customization dialog target `.DATA` partitioned data set, and change the following entries to the new users, groups, and unique `New_server` name, and `New_cluster` name profiles:

- If unique user IDs for the new servers are desired, define three new users and connect them to the following groups:

  ```
  &lt;New_CR_userid> &lt;CR_groupid>, &lt;CFG_groupid>
  &lt;New_SR_userid> &lt;SR_groupid>, &lt;CFG_groupid>
  &lt;New_client_userid> &lt;client_groupid>
  ```
- CBIND class profiles for the new cluster (generic server short name):

  ```
  CB.BIND.&lt;New_cluster>
  CB.&lt;New_cluster>
  ```
- SERVER class profiles for the new server and cluster:

  ```
  CB.&lt;New_server>.&lt;New_cluster>
  CB.&lt;New_server>.&lt;New_cluster>.&lt;cell>
  ```
- STARTED class profiles for the new server's controller and servant's regions:

  ```
  &lt;CR_proc>.&lt;New_CR_jobname> STDATA(USER(New_CR_userid)
                                      GROUP(CFG_groupid))
  &lt;New_SR_jobname>.* STDATA(USER(New_SR_userid) GROUP(CFG_groupid))
  ```

**Minimalist Profiles**: To minimize the number of users, groups, and profiles in the RACF data set, you can use one user ID, one group ID, and very generic profiles so they cover multiple servers in the same cell. Here is an example of profiles with one user (T5USR), one group (T5GRP), and a set of servers in the T5CELL having server short names starting with T5SRV* and generic server names starting with T5CL*. This technique can also be used with Integral JMS provider (IJP) and Network Deployment (ND) configurations.

```
/* CBIND Class profiles (UACC) - access to generic servers */
CB.BIND.T5CL* UACC(READ); PERMIT ID(T5GRP) ACC(CONTROL)

/* CBIND Class profiles (UACC) - access to objects in servers */
CB.T5CL* UACC(READ); PERMIT ID(T5GRP) ACC(CONTROL)

/* SERVER Class profiles - access to controllers (old style) */
CB.*.T5CL* UACC(NONE); PERMIT ID(T5USR) ACC(READ)

/* SERVER Class profiles - acc to controllers (new style) */
CB.*.*.T5CELL UACC(NONE); PERMIT ID(T5USR) ACC(READ)

/* STARTED Class profiles - (MGCRE) - for STCs, except servants */
T5ACR.* STDATA(USER(T5USR) GROUP(T5GRP)) /* controller*/
T5DMN.* STDATA(USER(T5USR) GROUP(T5GRP)) /* daemon */
T5CTRW.* STDATA(USER(T5USR) GROUP(T5GRP)) /* CTrace WTR*/
WMQX*.* STDATA(USER(T5USR) GROUP(T5GRP)) /* IJP */

/* STARTED Class profiles - (ASCRE - for servants) */
T5SRV*.* STDATA(USER(T5USR) GROUP(T5GRP)) /* servant */
```

*Tuning security:*

Use the following procedures to tune the performance, without compromising your security settings.

Enabling security decreases performance. The following tuning parameters provide ways to minimize this performance impact.

While it is not practical to run WebSphere Application Server for z/OS without security enabled, it is possible to perform certain tuning techniques to make the Application Server run better on z/OS. These techniques are documented in detail in "Security tuning tips" on page 126.

- Fine-tune the **Authentication cache timeout** value on the Authentication mechanisms and expiration panel in the administrative console. For more information, see the Secure administration, applications, and infrastructure settings topic.
- Configure the security cache properties. For more information, see the "Security cache properties" on page 123 topic.

- Enable the **Enable SSL ID tracking** option on the Session management panel in the administrative console. For more information, see the Session management settings topic.
- Modify the RACF security settings as documented in the “Security tuning tips” on page 126 article.
- Read the “Tuning security configurations” on page 124 article for more information.

***Hardening security configurations:***

There are several methods that you can use to protect the WebSphere Application Server infrastructure and applications from different forms of attack. Several different techniques can help with multiple forms of attack. Sometimes a single attack can leverage multiple forms of intrusion to achieve the end goal.

For example, in the simplest case, network sniffing can be used to obtain passwords and those passwords can then be used to mount an application-level attack. The following issues are discussed in IBM WebSphere Developer Technical Journal: WebSphere Application Server V5 advanced security and system hardening:
- Take preventative measures to protect the infrastructure.
- Make applications less vulnerable to attack.

***Securing passwords in files:***

Password encoding and encryption deters the casual observation of passwords in server configuration and property files.

The following topics are covered in this section:
- Password encoding and encryption
- Encoding passwords in files
- Enabling custom password encryption

*Encoding password in files:*

Use the **PropFilePasswordEncoder** utility to encode your passwords in the files. WebSphere Application Server does not provide a utility for decoding the passwords.

WebSphere Application Server contains several encoded passwords that are not encrypted. WebSphere Application Server provides the **PropFilePasswordEncoder** utility, which you can use to encode these passwords. However, the utility does not encode passwords that are contained within XML or XMI files. Instead, WebSphere Application Server automatically encodes the passwords in the following XML or XMI files.

*Table 4. XML and XMI files that contain encoded passwords*

| **File name** | **Additional information** |
|---|---|
| `profile_root/config/cells/cell_name/security.xml` | The following fields contain encoded passwords:<br>• LTPA password<br>• JAAS authentication data<br>• User registry server password<br>• LDAP user registry bind password<br>• Keystore password<br>• Truststore password |
| `war/WEB-INF/ibm_web_bnd.xml` | Specifies the passwords for the default basic authentication for the resource-ref bindings within all the descriptors, except in the Java cryptography architecture |

*Table 4. XML and XMI files that contain encoded passwords (continued)*

| File name | Additional information |
|---|---|
| `ejb jar/META-INF/ibm_ejbjar_bnd.xml` | Specifies the passwords for the default basic authentication for the resource-ref bindings within all the descriptors, except in the Java cryptography architecture |
| `client jar/META-INF/ibm-appclient_bnd.xml` | Specifies the passwords for the default basic authentication for the resource-ref bindings within all the descriptors, except in the Java cryptography architecture |
| `ear/META-INF/ibm_application_bnd.xml` | Specifies the passwords for the default basic authentication for the run as bindings within all the descriptors |
| *profile_root*/config/cells/*cell_name* /nodes/*node_name*/servers/ *server_name*/security.xml | The following fields contain encoded passwords: • Keystore password • Truststore password • Session persistence password • DRS client data replication password |
| *profile_root*/config/cells/*cell_name* /nodes/*node_name*/servers/ *server_name*/resources.xml | The following fields contain encoded passwords: • WAS40Datasource password • mailTransport password • mailStore password • MQQueue queue mgr password |
| `ibm-webservices-bnd.xmi` | |
| `ibm-webservicesclient-bnd.xmi` | |

You can use the **PropFilePasswordEncoder** utility to encode the passwords that are located in the following files.

*Table 5. Files that you can encode using the PropFilePasswordEncoder utility*

| File name | Additional information |
|---|---|
| *app_server_root* /properties/sas.client.props | Specifies the passwords for the following files: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword |
| *app_server_root* /properties/soap.client.props | Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.SOAP.loginPassword |
| *app_server_root* /properties/sas.tools.properties | Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword |
| *app_server_root* /properties/sas.stdclient.properties | Specifies passwords for: • com.ibm.ssl.keyStorePassword • com.ibm.ssl.trustStorePassword • com.ibm.CORBA.loginPassword |
| *app_server_root* /properties/wsserver.key | |

To encode a password again in one of the previous files, complete the following steps:

1. Access the file using a text editor and type over the encoded password. The new password is shown is no longer encoded and must be re-encoded.
2. Use the `PropFilePasswordEncoder.bat` or the `PropFilePasswordEncode.sh` file in the *app_server_root*/profiles/*profile_name*/bin directory to encode the password again.

   V6.0.x If you are encoding the z/SAS properties files again, type: `PropFilePasswordEncoder "file_name" -sas`. The `PropFilePasswordEncoder.bat` file encodes the known z/SAS properties.

   **Important:** z/SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

   If you are encoding files that are not z/SAS properties files, type `PropFilePasswordEncoder "file_name" password_properties_list`

   where:

   "*file_name*" is the name of the z/SAS properties file, and *password_properties_list* is the name of the properties to encode within the file.

   **Note:** Only the password should be encoded in this file using the **PropFilePasswordEncoder** tool.

   Use the **PropFilePasswordEncoder** utility to encode WebSphere Application Server password files only. The utility cannot encode passwords that are contained in XML files or other files that contain open and close tags.

If you reopen the affected files, the passwords are encoded. WebSphere Application Server does not provide a utility for decoding the passwords.

The reliance on passwords in configuration files can be minimized on WebSphere Application Server for z/OS by taking advantage of z/OS-specific features:

- Use a System Authorization Facility (SAF) registry to remove the requirement for a user registry server password.
- Select SAF authorization and delegation so role-to-user binding passwords are removed.
- Use a RACF keyring for all SSL repertoires, and trust and key file passwords are no longer required.
- Use native connectors, and configure sync-to-thread to possibly remove the need for Java Authentication and Authorization Service (JAAS) authentication data.

*PropFilePasswordEncoder command reference:*

The **PropFilePasswordEncoder** command encodes passwords that are located in plain text property files. This command encodes both Secure Authentication Server (SAS) property files and non-SAS property files. After you encode the passwords, a decoding command does not exist.

To encode passwords, you must run this command from the directory:

- V6.0.x *app_server_root*/bin

**Important:** SAS is supported only between Version 6.0.x and previous version servers that have been federated in a Version 6.1 cell.

### Syntax

The command syntax is as follows:

```
PropFilePasswordEncoder "file_name"
```

### Parameters

The following option is available for the **PropFilePasswordEncoder** command:

V6.0.x **-SAS**
This parameter is required if you are encoding passwords in the sas.client.props file.

**-help or -?**
If you specify this parameter, the script ignores all other parameters and displays usage text.

V6.0.x The following examples demonstrate the correct syntax:

PropFilePasswordEncoder *"file_name" password_properties_list*
PropFilePasswordEncoder *"file_name"* -SAS

*Enabling custom password encryption:*

After creating the server profile, perform this task to better protect passwords contained in configuration.

Create your custom class for encrypting passwords. For more information, see Plug point for custom password encryption.

Complete the following steps to enable custom password encryption.

1. Add the following system properties for every server and client process. For server processes, update the `server.xml` file for each process. Add these properties as a genericJvmArgument argument preceded by a **-D** prefix.
   ```
   com.ibm.wsspi.security.crypto.customPasswordEncryptionClass=
          com.acme.myPasswordEncryptionClass
   com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=true
   ```
   **Tip:** If the custom encryption class name is com.ibm.wsspi.security.crypto.CustomPasswordEncryptionImpl, it is automatically enabled when this class is present in the classpath. Do not define the system properties that are listed previously when the custom implementation has this package and class name. To disable encryption for this class, you must specify `com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=false` as a system property.
2. Add the Java archive (JAR) file containing the implementation class to the *app_server_root*`/classes` directory so that the WebSphere Application Server runtime can load the file.
3. Restart all server processes.
4. Edit each configuration document that contains a password and save the configuration. All password fields are then run through the **WSEncoderDecoder** utility, which calls the plug point when it is enabled. The `{custom:alias}` tags are displayed in the configuration documents. The passwords, even though they are encrypted, are still Base64-encoded. They seem similar to encoded passwords, except for the tags difference.
5. Encrypt any passwords that are in client-side property files using the **PropsFilePasswordEncoder** (.bat or .sh) utility. This utility requires that the properties listed previously are defined as system properties in the script to encrypt new passwords instead of encoding them.
6. To decrypt passwords from client Java virtual machines (JVMs), add the properties listed previously as system properties for each client utility.
7. Ensure that all nodes have the custom encryption classes in their class paths prior to enabling this function. The order in which enablement occurs is important. When adding a new node to a cell that contains password encryption, the new node must contain the custom encryption classes prior to using the **addNode** command. Consider the following Network Deployment enablement scenarios:
   a. The StandAloneProfile profile is encrypting passwords with a different key prior to federation to a deployment manager cell. For this scenario, you must uninstall custom password encryption to ensure that the configuration has `{xor}` tags preceding the passwords prior to running the **addNode** command. The same implementation of the plug point must be in the `/classes` directory prior to

running the **addNode** command, and the proper configuration properties are set so that the new node can recognize the encrypted password format of the `security.xml` file after federation completes.

b. The StandAloneProfile profile does not have password encryption configured prior to federation to a deployment manager cell. The same implementation of the plug point must be in the `/classes` directory prior to running the **addNode** command, and the proper configuration properties are set so that the new node can recognize the encrypted password format of the `security.xml` file after federation completes.

c. If enabling custom password encryption in a cell with multiple nodes present, update the correct configuration properties and have the custom password encryption implementation class located on all nodes. Stop all processes in the cell, and then start the deployment manager. Use the administrative console to edit the security configuration and then save it. Verify that the passwords are encrypted by looking at the `security.xml` file to see if the passwords are preceded by `{custom:alias}` tags.

d. Run the **syncNode** command on each node, and start each one individually. If any nodes fail to start, make sure that they have custom password encryption enabled properly in each `security.xml` file and that the implementation class is in the appropriate `/classes` directory for the platform.

Custom password encryption is enabled.

If custom password encryption fails or is no longer required, see “Disabling custom password encryption.”

*Disabling custom password encryption:*

If custom password encryption fails or is no longer required, perform this task to disable custom password encryption.

Enable custom password encryption.

Complete the following steps to disable custom password encryption.

1. Change the com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled property to be `false` in the `security.xml` file, but leave the com.ibm.wsspi.security.crypto.customPasswordEncryptionClass property configured. Any passwords in the model that still have the `{custom:alias}` tag are decrypted by using the customer password encryption class.
2. If an encryption key is lost, any passwords that are encrypted with that key cannot be retrieved. To recover a password, retype the password in the password field in plaintext and save the document. The new password must be written out using encoding with the {xor} tag with scripting or from the administrative console.

   ```
   com.ibm.wsspi.security.crypto.customPasswordEncryptionClass=
          com.acme.myPasswordEncryptionClass
   com.ibm.wsspi.security.crypto.customPasswordEncryptionEnabled=false
   ```
3. Restart all processes to make the changes effective.
4. Edit each configuration document that contains an encrypted password and save the configuration. All password fields are then run through the **WSEncoderDecoder** utility, which calls the plug point in the presence of the `{custom:alias}` tag. The `{xor}` tags display in the configuration documents again after the documents are saved.
5. Decrypt and encode any passwords that are in client-side property files using the **PropsFilePasswordEncoder** (.bat or .sh) utility. If the encryption class is specified, but custom encryption is disabled, running this utility converts the encryption to encoding and causes the `{xor}` tags to display again.
6. Disable custom password encryption from the client Java virtual machines (JVMs) by adding the system properties listed previously to all client scripts. This action enables the code to decrypt

passwords, but this action is not used to encrypt them again. The {xor} algorithm becomes the default for encoding. Leave the custom password encryption class defined for a time in case any encrypted passwords still exist in the configuration.

Custom password encryption is disabled.

## SPNEGO trust association interceptor (TAI) troubleshooting tips

Presented here is a list of trouble shooting tips useful in diagnosing Simple and Protected GSS-API Negotiation (SPNEGO) TAI problems and exceptions.

The IBM Java Generic Security Service (JGSS) and IBM SPNEGO providers use a Java virtual machine (JVM) custom property to control trace information. The SPNEGO TAI uses the JRas facility to allow an administrator to trace only specific classes. To debug the TAI using tracing, the following important trace specifications or JVM customer should be used:

*Table 6. SPNEGO TAI trace specifications*

| Trace | Use |
|---|---|
| *com.ibm.security.jgss.debug* | Set this **JVM Custom Property** to `all` to trace through JGSS code. Messages appear in the `trace.log` file, and **SystemOut.log**. |
| *com.ibm.security.krb5.Krb5Debug* | Set this **JVM Custom Property** to `all` to trace through the Kerberos5-specific JGSS code. Messages appear in the `trace.log` file, and **SystemOut.log**. |
| *com.ibm.ws.security.spnego.** | Set this trace on using the **administrative console > troubleshooting > Logging and Tracing > server1 > Change Log Detail Levels > com.ibm.ws.security.spnego.***. Messages appear in the `trace.log` file. |

***Problem: WebSphere Application Server and the Active Directory (AD) Domain Controller's time are not synchronized within 5 minutes:***

The time is not synchronized between WebSphere Application Server and AD Domain Controller.

```
[2/24/06 13:12:46:093 CST] 00000060 Context       2 com.ibm.ws.security.spnego.Context
    begin GSSContext accepted
[2/24/06 13:12:46:093 CST] 00000060 Context       E com.ibm.ws.security.spnego.Context
    begin
CWSPN0011E: An invalid SPNEGO token has been encountered while authenticating a
    HttpServletRequest:
0000:  60820160 06062b06 01050502 a1820154    `..` ..+. .... ...T
0010:  30820150 a0030a01 01a10b06 092a8648    0..P .... .... .*.H
0020:  82f71201 0202a282 013a0482 01366082    .... .... .:.. .6`.
0030:  01320609 2a864886 f7120102 0203007e    .2.. *.H. .... ...~
0040:  82012130 82011da0 03020105 a1030201    ..!0 .... .... ....
0050:  1ea41118 0f323030 36303232 34313931    .... .200 6022 4191
0060:  3234365a a5050203 016b48a6 03020125    246Z .... .kH. ...%
0070:  a9161b14 57535345 432e4155 5354494e    .... WSSE C.AU STIN
0080:  2e49424d 2e434f4d aa2d302b a0030201    .IBM .COM .-0+ ....
0090:  00a12430 221b0448 5454501b 1a773230    ..$0 "..H TTP. .w20
00a0:  30337365 63646576 2e617573 74696e2e    03se cdev .aus tin.
00b0:  69626d2e 636f6dab 81aa1b81 a76f7267    ibm. com. .... .org
00c0:  2e696574 662e6a67 73732e47 53534578    .iet f.jg ss.G SSEx
00d0:  63657074 696f6e2c 206d616a 6f722063    cept ion,  maj or c
00e0:  6f64653a 2031302c 206d696e 6f722063    ode:  10,  min or c
00f0:  6f64653a 2033370a 096d616a 6f722073    ode:  37. .maj or s
0100:  7472696e 673a2044 65666563 74697665    trin g: D efec tive
0110:  20746f6b 656e0a09 6d696e6f 72207374     tok en.. mino r st
0120:  72696e67 3a20436c 69656e74 2074696d    ring : Cl ient  tim
```

```
0130:  65204672 69646179 2c204665 62727561    e Fr iday , Fe brua
0140:  72792032 342c2032 30303620 61742031    ry 2 4, 2 006  at 1
0150:  3a31323a 34352050 4d20746f 6f20736b    :12: 45 P M to o sk
0160:  65776564                               ewed
```

**Solution:** You can fix this in one of two ways. The preferred way is to synchronize the WebSphere Application Server system time to within 5 minutes of the AD server's time. A best practice is to use a time server to keep all systems synchronized. Or you can add or adjust the clockskew parameter in the Kerberos configuration file.

**Note:** The default for the clockskew parameter is 300 seconds ( or 5 minutes).

***Problem: Getting exception: No factory available to create a name for mechanism 1.3.6.1.5.5.2:***

There apparently is no factory available to process the creation of a name for the specific mechanism.

The `systemout.log` file displays something like this:

```
[4/8/05 22:51:24:542 EDT] 5003e481 SystemOut     O [JGSS_DBG_PROV] Provider
        IBMJGSSProvider version 1.01 does not support mech 1.3.6.1.5.5.2
[4/8/05 22:51:24:582 EDT] 5003e481 ServerCredent >
        com.ibm.ws.security.spnego.ServerCredential initialize ENTRY
SPNEGO014: Kerberos initialization Failure: org.ietf.jgss.GSSException, major code: 2,
        minor code: 0
 major string: Unsupported mechanism
 minor string: No factory available to create name for mechanism 1.3.6.1.5.5.2
 at com.ibm.security.jgss.i18n.I18NException.throwGSSException
        (I18NException.java:30)
 at com.ibm.security.jgss.GSSManagerImpl.a(GSSManagerImpl.java:36)
 at com.ibm.security.jgss.GSSCredentialImpl.add(GSSCredentialImpl.java:217)
 at com.ibm.security.jgss.GSSCredentialImpl.<init>(GSSCredentialImpl.java:264)
```

**Solution:** Check the `java.security` file to ensure it contains the IBMSPNEGO security provider and that the provider is defined correctly. The `java.security` file should contain a line similar to:

```
security.provider.6=com.ibm.security.jgss.mech.spnego.IBMSPNEGO
```

***Problem: Getting an exception:***

An exception has occurred when reporting to the client.

You get the following display.

```
Error authenticating request. Reporting to client
Major code = 11, Minor code = 31
org.ietf.jgss.GSSException, major code: 11, minor code: 31
 major string: General failure, unspecified at GSSAPI level
 minor string: Kerberos error while decoding and verifying token:
              com.ibm.security.krb5.internal.KrbException, status code: 31
 message: Integrity check on decrypted field failed
```

as the JGSS library is trying to process the SPNEGO token.

**Cause:** This exception is the result of encoding the ticket using one key and attempting to decode it using a different key. There are number of possible reasons for this condition:

1. The Kerberos keytab file has not been copied to the server machine after it has been regenerated.
2. The Kerberos configuration points to the wrong Kerberos keytab file.
3. The Kerberos service principal name (SPN) has been defined to the Active Directory more than once; this can occur because you have another userid with a similarly defined SPN (either exactly the same name, or one having a different name but with a port defined part of the SPN).

**Solution**: If the problem is with the Kerberos keytab file, then fix it. If the problem is with multiple SPN definitions, then remove the extra or conflicting SPN, confirm that the SPN is no longer registered with the Active Directory, and then add the SPN. The Active Directory may need to be searched for other entries with SPNs defined that clash with the SPN.

To confirm that the SPN is not registered, the command:

```
setspn –l userid
```

should return with the following response:

```
Cannot find account userid
```

***Problem: Single sign-on is not occurring.:***

When trace is turned on, the following message appears:

```
[2/27/06 14:28:04:191 CST] 00000059 SpnegoHandler <
         com.ibm.ws.security.spnego.SpnegoHandler handleRequest: Received a
         non-SPNEGO Authorization Header RETURN
```

**Cause:** The client is returning an NT LAN manager (NTLM) response to the authorize challenge, not a SPNEGO token. This condition can be occur due to any of the following reasons:

- The client has not been configured properly.
- The client is not using a supported browser. For example, when using Microsoft Internet Explorer 5.5, SP1 responds with a non-SPNEGO authentication header.
- The user has not logged into the Active Directory domain, or into a trusted domain, or the client used does not support integrated authentication with Windows – in this case, the SPNEGO TAI is working properly.
- The user is accessing a service defined on the same machine upon which the client is running (local host). Microsoft Internet Explorer resolves the host name of the URL to `http://localhostsomeURL` instead of a fully qualified name.
- The SPN is not found in the Active Directory. The SPN must be of the format `HTTP/server.realm.com`. The command to add the SPN is

  ```
  setspn –a HTTP/server.realm.com userid
  ```

If the SPN is defined incorrectly as `HTTP/server.realm.com@REALM.COM` with the addition of `@REALM.COM`, then delete the user, redefine the user, and redefine the SPN.

***Problem: Credential Delegation is not working:***

An invalid option is detected. When trace is turned on, the following message is displayed:

```
com.ibm.security.krb5.KrbException, status code: 101 message: Invalid option in
ticket request
```

**Cause:** The Kerberos configuration file is not properly configured.

**Solution:** Ensure that neither renewable, nor proxiable are set to true.

***Problem: Unable to get SSO working using RC4-HMAC encryption.:***

When trace is turned on, you get the following message in the trace:

```
com.ibm.security.krb5.internal.crypto.KrbCryptoException, status code: 0
 message: Checksum error; received checksum does not match computed checksum
```

**Cause:** RC4-HMAC encryption is not supported with a Microsoft Windows 2000 Kerberos key distribution center (KDC). To confirm this condition, examine the trace and identify where the exception is thrown. The content of the incoming ticket should be visible in the trace. Although the incoming ticket is encrypted, the

SPN for the service is readable. If a Microsoft Windows 2000 KDC is used and the system is configured to use RC4-HMAC, the string representing the ticket for `userid@REALM` ( instead of the expected `HTTP/hostname.realm@REALM`) is displayed. For example, this is beginning of the ticket received from a Microsoft Windows 2000 KDC:

```
0000: 01 00 6e 82 04 7f 30 82  04 7b a0 03 02 01 05 a1  ..n...0.........
0010: 03 02 01 0e a2 07 03 05  00 20 00 00 00 a3 82 03  ...............
0020: a5 61 82 03 a1 30 82 03  9d a0 03 02 01 05 a1 0a  .a...0..........
0030: 1b 08 45 50 46 44 2e 4e  45 54 a2 18 30 16 a0 03  ...REALM.COM.0..
0040: 02 01 01 a1 0f 30 0d 1b  0b 65 70 66 64 77 61 73  .....0...userid
0050: 75 6e 69 74 a3 82 03 6e  30 82 03 6a a0 03 02 01  .a.f...n0..j....
```

The realm is REALM.COM. The service name is userid. A correctly formed ticket for the same SPN is:

```
0000: 01 00 6e 82 04 56 30 82  04 52 a0 03 02 01 05 a1  ..n..V0..R......
0010: 03 02 01 0e a2 07 03 05  00 20 00 00 00 a3 82 03  ...............
0020: 82 61 82 03 7e 30 82 03  7a a0 03 02 01 05 a1 0a  .a...0..z.......
0030: 1b 08 45 50 46 44 2e 4e  45 54 a2 2a 30 28 a0 03  ..REALM.COM.0...
0040: 02 01 02 a1 21 30 1f 1b  04 48 54 54 50 1b 17 75  .....0...HTTP..u
0050: 73 31 30 6b 65 70 66 77  61 73 73 30 31 2e 65 70  serid.realm.com.
0060: 66 64 2e 6e 65 74 a3 82  03 39 30 82 03 35 a0 03  ...n.....90..5..
```

**Solution:** To correct the problem, either use the Single data encryption standard (DES) or use a Microsoft Windows 2003 Server for a KDC. Remember to regenerate the SPN, and the Kerberos keytab file.

***Problem: User receives the following message when accessing a protected URL through the SPNEGO SSO:***

```
Bad Request

Your browser sent a request that this server could not understand.
Size of request header field exceeds server limit.

Authorization: Negotiate YII......
```

**Cause:** This message is generated by the Apache/IBM HTTP Server. This server is indicating that the authorization header returned by the user's browser is too large. The long string that follows the word Negotiate (in the error message above) is the SPNEGO token. This SPNEGO token is a wrapper of the Microsoft Windows Kerberos token. Microsoft Windows includes the user's PAC information in the Kerberos token. The more security groups that the user belongs to, the more PAC information is inserted in the Kerberos token, and the larger the SPNEGO becomes. IBM HTTP Server 2.0 (also Apache 2.0 and IBM HTTP Server 6.0) limit the size of any acceptable HTTP header to be 8K. In Microsoft Windows domains having many groups, and with user membership in many groups, the size of the user's SPNEGO token may exceed the 8K limit.

**Solution:** If possible, reduce the number of security groups the user is a member of. IBM HTTP Server 2.0.47 cumulative fix PK01070 allows for HTTP header sizes up to and beyond the Microsoft limit of 12K. WebSphere Application Server Version 6.0 users can obtain this fix in fixpack 6.0.0.2.

**Note:** Non-Apache based Web servers may require differing solutions.

***Problem: Even with JGSS tracing disabled, some KRB_DBG_KDC messages appear in the SystemOut.log:***

**Cause:** While most of the JGSS tracing is controlled by the com.ibm.security.jgss.debug property, a small set of messages are controlled by the com.ibm.security.krb5.Krb5Debug property. The com.ibm.security.krb5.Krb5Debug property has a default value to put some messages to the **SystemOut.log**.

**Solution:** To remove all KRB_DBG_KDC messages from the **SystemOut.log**, set the JVM property as follows:

```
-Dcom.ibm.security.krb5.Krb5Debug=none
```

.

***Problem:* HTTP Post parameters are lost during interaction with the SPNEGO TAI, when stepping down to userid/password login.:**

**Cause:** The Microsoft Internet Explorer maintains state during a user's request. If a request was given the response of an ″HTTP 401 Authenticate Negotiate″, and the browser responds with a NTLM token obtained through a userid/password challenge, the browser resubmits the request. If this second request is given a response of an HTML page containing a redirection to the same URL but with new arguments (via Javascript) then the browser does not resubmit the POST parameters. To avoid this problem, it is critical to NOT perform the automatic redirection. If the user clicks on a link, the problem does not occur. See section 5.2 Client Returns NTLM Token to SPNEGO Challenge for a resolution to the problem,

**Solution:** The browser responds to the Authenticate/Negotiate challenge with an NTLM token, not an SPNEGO token. The SPNEGO TAI sees the NTLM, and returns back a HTTP 403 response, along with the HTML page. When the browser runs the Javascript redirTimer function, any POST of GET parameters that were present on the original request are lost.

By leveraging the SPN<id>.NTLMTokenReceivedPage property, an appropriate message page can be returned to the user. The default message that is returned (in the absence of a user defined property) is:

```
"<html><head><title>An NTLM Token was Received.</title></head>"
 + "<body>Your browser configuration is correct, but you have not logged into
          a supported Windows Domain."
 + "<p>Please login to the application using the normal login page.</html>";
```

Using the SPN<id>.NTLMTokenReceivedPage property, you can customize the exact response. It is critical that the returned HTML not perform a redirection.

When the SPNEGO TAI has been configured to use the shipped default HTTPHeaderFilter class as the SPN<id>.filterClass, then the SPN<id>.filter can be used to allow the second request to flow directly to the normal WebSphere Application Server security mechanism. In this way, the user experiences the normal authentication mechanism.

An example of such a configuration follows. The required SPNEGO TAI properties necessary and the HTML file content are presented.

*Table 7. SPNEGO TAI properties and HTML*

| SPNEGO TAI Property Name | HTML File Content |
|---|---|
| com.ibm.ws.security.spnego.SPN1.hostName | server.wasteched30.torolab.ibm.com |
| com.ibm.ws.security.spnego.SPN1.filterClass | com.ibm.ws.security.spnego.HTTPHeaderFilter |
| com.ibm.ws.security.spnego.SPN1.filter | request-url!=noSPNEGO |
| com.ibm.ws.security.spnego.SPN1.NTLMTokenReceivedPage | File:///C:/temp/NTLM.html |

**Note:** Observe that the filter property instructs the SPNEGO TAI to NOT intercept any HTTP request that contains the string “noSPNEGO”.

Here is an example of a generating a helpful response.

```
<html>
<head>
<title>NTLM Authentication Received </title>
<script language="javascript">
 var purl=""+document.location;
```

```
if (purl.indexOf("noSPNEGO")<0) {
  if(purl.indexOf('?')>=0) purl+="&noSPNEGO";
  else purl+="?noSPNEGO";
 }
</script>
</head>
<body>
<p>An NTLM token was retrieved in response to the SPNEGO challenge. It is likely that
you are not logged into a Windows domain.<br>
Click on the following link to get the requested website.
<script language="javascript">
 document.write("<a href='"+purl+"'>");
 document.write("Open the same page using the normal authentication
  mechanism.");
 document.write("</a><br>");
</script>
You will not automatically be redirected.
</body>
</html>
```

## Transactions

### CICS tuning tips for z/OS

These recommendations only apply to WebSphere applications that access CICS.

The LGDFINT system initialization parameter specifies the log defer interval used by CICS log manager when determining how long to delay a forced journal write request before invoking the MVS system logger. The value is specified in milliseconds. Performance evaluations of typical CICS transaction workloads have shown that the default setting of 5 milliseconds gives the best balance between response time and central processor cost. Be aware that CICS performance can be adversely affected by a change to the log defer interval value. Too high a value will delay CICS transaction throughput due to the additional wait before invoking the MVS system logger. An example of a scenario where a reduction in the log defer interval might be beneficial to CICS transaction throughout would be where many forced log writes are being issued, and little concurrent task activity is occurring. Such tasks will spend considerable amounts of their elapsed time waiting for the log defer period to expire. In such a situation, there is limited advantage in delaying a call to the MVS system logger to write out a log buffer, since few other log records will be added to the buffer during the delay period.

- Set the LGDFINT system initialization parameter to 5.

  While CICS is running, you can use the CEMT SET SYSTEM[LOGDEFER(*value*)] command to alter the LGDFINT setting dynamically.
- Set the CICS RECEIVECOUNT value high enough to handle all concurrent EXCI pipes on the system.

  The default value is 4. You set this value in the EXCI sessions resource definition.

For more detailed information on CICS, refer to the *CICS Performance Guide*.

### GRS tuning tips for z/OS

WebSphere for z/OS uses GRS to communicate information between servers in a sysplex. When there are multiple servers defined in a system or a sysplex, a request may end up on the wrong server. To determine where the transaction is running WebSphere uses GRS. Therefore, if you are using global transactions, WebSphere will issue an enqueue for that transaction at the start of the transaction and hold on to that enqueue until the transaction ends. WebSphere for z/OS uses GRS enqueues for the following:

- Two-phase commit transactions involving more than one server
- HTTP sessions in memory
- Stateful EJBs
- ″Sticky″ transactions to keep track of pseudo-conversational states.

- If you **are not** in a sysplex, you should configure GRS=NONE.
- If you **are** in a sysplex, we strongly recommend GRS=STAR.

This requires configuring GRS to use the coupling facility. All of the WebSphere enqueues are issued with RNL=NO, which prevents misconfiguring the GRSRNLxx with inappropriate values. See the GRS documentation for details on setting this up.

## Learn about WebSphere programming extensions

Use this section as a starting point to investigate the WebSphere programming model extensions for enhancing your application development and deployment.

See Learn about WebSphere applications: Overview and new features for a brief description of each WebSphere extension.

In addition, now your applications can use the Eclipse extension framework. Your applications are extensible as soon as you define an extension point and provide the extension processing code for the extensible area of the application. You can also plug an application into another extensible application by defining an extension that adheres to the target extension point requirements. The extension point can find the newly added extension dynamically and the new function is seamlessly integrated in the existing application. It works on a cross Java 2 Platform, Enterprise Edition (J2EE) module basis.

The application extension registry uses the Eclipse plug-in descriptor format and application programming interfaces (APIs) as the standard extensibility mechanism for WebSphere applications. Developers that build WebSphere application modules can use WebSphere Application Server extensions to implement Eclipse tools and to provide plug-in modules to contribute functionality such as actions, tasks, menu items, and links at predefined extension points in the WebSphere application. For more information about this feature, see Application extension registry.

### Application profiling

***Application profiling performance considerations:***

Application profiling enables assembly configuration techniques that improve your application run time, performance and scalability. You can configure tasks that identify incoming requests, identify access intents determining concurrency and other data access characteristics, and profiles that map the tasks to the access intents.

The capability to configure the application server can improve performance, efficiency and scalability, while reducing development and maintenance costs. The application profiling service has no tuning parameters, other than a checkbox for disabling the service if the service is not necessary. However, the overhead for the application profile service is small and should not be disabled, or unpredictable results can occur.

Access intents enable you to specify data access characteristics. The WebSphere runtime environment uses these hints to optimize the access to the data, by setting the appropriate isolation level and concurrency. Various access intent hints can be grouped together in an access intent policy.

In WebSphere Application Server, it is recommended that you configure bean level access intent for loading a given bean. Application profiling enables you to configure multiple access intent policies on the entity bean, if desired. Some callers can load a bean with the intent to read data, while others can load the bean for update. The capability to configure the application server can improve performance, efficiency, and scalability, while reducing development and maintenance costs.

Access intents enable the EJB container to be configured providing optimal performance based on the specific type of enterprise bean used. Various access intent hints can be specified declaratively at deployment time to indicate to WebSphere resources, such as the container and persistence manager, to provide the appropriate access intent services for every EJB request.

The application profiling service improves overall entity bean performance and throughput by fine tuning the run time behavior. The application profiling service enables EJB optimizations to be customized for multiple user access patterns without resorting to ″worst case″ choices, such as pessimistic update on a bean accessed with the findByPrimaryKey method, regardless of whether the client needs it for read or for an update.

Application profiling provides the capability to define the following hierarchy: **Container-Managed Tasks** > **Application Profiles** > **Access Intent Policies** > **Access Intent Overrides**. Container-managed tasks identify units of work (UOW) and are associated with a method or a set of methods. When a method associated with the task is invoked, the task name is propagated with the request. For example, a UOW refers to a unique path within the application that can correspond to a transaction or ActivitySession. The name of the task is assigned declaratively to a J2EE client or servlet, or to the method of an enterprise bean. The task name identifies the starting point of a call graph or subgraph; the task name flows from the starting point of the graph downstream on all subsequent IIOP requests, identifying each subsequent invocation along the graph as belonging to the configured task. As a best practice, wherever a UOW starts, for example, a transaction or an ActivitySession, assign a task to that starting point.

The application profile service associates the propagated tasks with access intent policies. When a bean is loaded and data is retrieved, the characteristics used for the retrieval of the data are dictated by the application profile. The application profile configures the access intent policy and the overrides that should be used to access data for a specific task.

Access intent policies determine how beans are loaded for specific tasks and how data is accessed during the transaction. The access intent policy is a named group of access intent hints. The hints can be used, depending on the characteristics of the database and resource manager. Various access intent hints applied to the data access operation govern data integrity. The general rule is, the more data integrity, the more overhead. More overhead causes lower throughput and the opportunity for simultaneous data access from multiple clients.

If specified, access intent overrides provide further configuration for the access intent policy.

**Best practices**

Application profiling is effective in a variety of different scenarios. The following are example situations where application profiling is useful

- **The same bean is loaded with different data access patterns**

  The same bean or set of beans can be reused across applications, but each of those applications has differing requirements for the bean or for beans within the invocation graph. One application can require that beans be loaded for update, while another application requires beans be loaded for read only. Application profiling enables deploy time configuration for beans to distinguish between EJB loading requirements.
- **Different clients have different data access requirements**

  The same bean or set of beans can be used for different types of client requests. When those clients have different requirements for the bean, or for beans within the invocation graph, application profiling can be used to tailor the bean loading characteristics to the requirements of the client. One client can require beans be loaded for update, while another client requires beans be loaded for read only. Application profiling enables deploy time configuration for beans to distinguish between EJB loading requirements.

**Monitoring tools**

You can use the Tivoli Performance Viewer, database and logs as monitoring tools.

You can use the Tivoli Performance Viewer to monitor various metrics associated with beans in an application profiling configuration. The following sections describe at a high level the Tivoli Performance Viewer metrics that reflect changes when access intents and application profiling are used:

- **Collection scope**

  The enterprise beans group contains EJB life cycle information, either a cumulative value for a group of beans, or for specific beans. You can monitor this information to determine the difference between using the ActivitySession scope versus the transaction scope. For the transaction scope, depending on how the container transactions are defined, activates and passivates can be associated with method invocations. The application could use the ActivitySession scope to reduce the frequency of activates and passivates. For more information, see ″Using the ActivitySession service.″
- **Collection increment**

  The enterprise beans group contains EJB life cycle information, either a cumulative value for a group of beans, or for specific beans. You can monitor *Num Activates* to watch the number of enterprise beans activated for a particular findByPrimaryKey operation. For example, if the collection increment is set to 10, rather than the default 25, the *Num Activates* value shows 25 for the initial findByPrimaryKey, before any result set iterator runs. If the number of activates rarely exceeds the collection increment, consider reducing the collection increment setting.
- **Resource manager prefetch increment**

  The resource manager prefetch increment is a hint acted upon by the database engine to depend upon the database. The Tivoli Performance Viewer does not have a metric available to show the effect of the resource manager prefetch increment setting.
- **Read ahead hint**

  The enterprise beans group contains EJB life cycle information, either a cumulative value for a group of beans, or for specific beans. You can monitor *Num Activates* to watch the number of enterprise beans activated for a particular request. If a read ahead association is not in use, the *Num Activates* value shows a lower initial number. If a read ahead association is in use, the *Num Activates* value represents the number of activates for the entire call graph.

**Database tools** are helpful in monitoring the different bean loading characteristics that introduce contention and concurrency issues. These issues can be solved by application profiling, or can be made worse by the misapplication of access intent policies.

Database tools are useful for monitoring locking and contention characteristics, such as locks, deadlocks and connections open. For example, for locks the DB2 Snapshot Monitor can show statistics for lock waits, lock time-outs and lock escalations. If excessive lock waits and time-outs are occurring, application profiling can define specific client tasks that require a more string level of locking, and other client tasks that do not require locking. Or, a different access intent policy with less restrictive locking could be applied. After applying this configuration change, the snapshot monitor shows less locking behavior. Refer to information about the database you are using on how to monitor for locking and contention.

The **application server logs** can be monitored for information about rollbacks, deadlocks, and other data access or transaction characteristics that can degrade performance or cause the application to fail.

## Dynamic cache

***Managing cache entries stored on a disk:***

Use this page to set Java virtual machine (JVM) custom properties to maintain cache entries that are saved to disk.

**Steps for this task**

You can set the custom properties globally to affect all cache instances, or you can set the custom property on a single cache instance. In most cases, set the properties on the individual cache instances.

To set the custom properties on the default cache instance, use the global option. If you set the same property both globally and on a cache instance, the value that is set on the cache instance overrides the global value.

To configure the custom properties on a single object cache instance or servlet cache instance, perform the following steps:

1. In the administrative console, click one of the following paths:
   - To configure a servlet cache instance, click **Resources** > **Cache instances** > **Servlet cache instances** > *servlet_cache_instance_name* > **Custom properties** > **New**.
   - To configure an object cache instance, click **Resources** > **Cache instances** > **Object cache instances** > *object_cache_instance_name* > **Custom properties** > **New**.
2. Type the name of the custom property. When configuring these custom properties on a single cache instance, you do not use the full property path. For example, type `explictBufferLimitOnStop` to configure the com.ibm.ws.cache.CacheConfig.explictBufferLimitOnStop custom property.
3. Type a valid value for the property in the **Value** field.
4. Save the property and restart WebSphere Application Server.

To configure the custom property globally across all configured cache instances, perform the following steps:

1. In the administrative console, click **Servers** > **Application servers** > *server_name* > **Java and process management** > **Process definition** > **Java virtual machine** > **Custom properties** > **New**.
2. Type the name of the custom property (for example, com.ibm.ws.cache.CacheConfig.explictBufferLimitOnStop) in the **Name** field.
3. Type a valid value for the property in the **Value** field.
4. Save the property and restart WebSphere Application Server.

**com.ibm.ws.cache.CacheConfig.htodCleanupFrequency**

Use this property to change the amount of time between disk cache cleanup.

**Important:** Setting this custom property manually is deprecated for V6.1. Therefore, you should use the administrative console to set this property. To set this property in the administrative console, click one of the following paths:

- To configure a servlet cache instance, click **Resources** > **Cache instances** > **Servlet cache instances** > *servlet_cache_instance_name*.
- To configure an object cache instance, click **Resources** > **Cache instances** > **Object cache instances** > *object_cache_instance_name*.

Then:

1. Under Disk Cache setting, select the Enable disk offload field if it is not already selected.
2. Under Performance Settings, select Balanced performance and balanced memory usage or Custom.
3. In the Disk cache cleanup frequency field, specify an appropriate length of time, in minutes.

By default, the disk cache cleanup is scheduled to run at midnight to remove expired cache entries and cache entries that have not been accessed in the past 24 hours. However, if you have thousands of cache entries that might expire within one or two hours, the files that are in the disk cache can grow large and become unmanageable. Use the com.ibm.ws.cache.CacheConfig.htodCleanupFrequency custom property to change the time interval between disk cache cleanup.

| | |
|---|---|
| Units | minutes<br><br>For example, a value of 60 means 60 minutes between each disk cache cleanup. |
| Default | 0<br><br>The disk cache cleanup occurs at midnight every 24 hours. |

**com.ibm.ws.cache.CacheConfig.htodDelayOffloadEntriesLimit**

Use this property to specify the number of different cache IDs that can be saved in memory for the dependency ID and template buffers. Consider increasing this value if you have a lot of memory in your server and you want to increase the performance of your disk cache.

**Important:** Setting this custom property manually is deprecated for V6.1. Therefore, you should use the administrative console to set this property. To set this property in the administrative console, click one of the following paths:

- To configure a servlet cache instance, click **Resources** > **Cache instances** > **Servlet cache instances** > *servlet_cache_instance_name*.
- To configure an object cache instance, click **Resources** > **Cache instances** > **Object cache instances** > *object_cache_instance_name*.

Then:

1. Under Disk Cache setting, select the Enable disk offload field, if it is not already selected.
2. Under Disk Cache settings, select Limit disk cache size in entries, if it is not already selected.
3. In the Disk cache size field, specify the number of cache IDs that can be saved in memory for the dependency ID and template buffers.

| | |
|---|---|
| Units | number of cache IDs<br><br>For example, a value of 1000 means that each dependency ID or template ID can have up to 1000 different cache IDs in memory. |
| Default | 1000 |
| Minimum | 100 |

**Tune the delay offload function**

Use these properties to tune the delay offload function for the disk cache.

**Important:** Setting these custom properties manually is deprecated for V6.1. You should use the administrative console to set these properties. The individual property descriptions include information on how to use the administrative console to set these properties.

The delay offload function uses extra memory buffers for dependency IDs and templates to delay the disk offload and minimize the input and output operations. However, if most of your cache IDs are longer than 100 bytes, the delay offload function might use too much memory. Use any combination of the following properties to tune your configuration:

- To increase or decrease the in-memory limit of cache IDs for dependency ID and template buffers, use the com.ibm.ws.cache.CacheConfig.htodDelayOffloadEntriesLimit custom property.

- To disable the disk cache delay offload function, use the com.ibm.ws.cache.CacheConfig.htodDelayOffload custom property. Disabling this property saves all cache entries to disk immediately after removing them from the memory cache.

**com.ibm.ws.cache.CacheConfig.explictBufferLimitOnStop**

Use this custom property when the flush-to-disk-on-stop feature is enabled. When the server is stopping, offloads are limited to the value specified for this property, pending removal of entries in the explicit invalidation buffer. If this property is set to `0`, there is no limit to the number of offloads that can occur. Only positive integers are accepted as values for this property. If the number of entries in the explicit invalidation buffer is greater then the specified limit, all of the disk files for this specified cache instance are deleted after the server stops.

**Important:** You cannot use the administrative console to set this property.

***Tuning dynamic cache with the cache monitor:***

Use this task to interpret cache monitor statistics to improve the performance of the dynamic cache service.

Verify that dynamic cache is enabled and that the cache monitor application is installed on your application server.

See the Displaying cache information topic in the *Administering applications and their environment* PDF to configure the cache monitor application.

Use the cache monitor to watch cache hits versus misses. By comparing these two values, you can determine how much dynamic cache is helping your application, and if you can take any additional steps to further improve performance and decrease the cost of processing for your application server.

1. Start cache monitor and click on **Cache Statistics**. You can view the following cache statistics:

| Cache statistic | Description |
|---|---|
| **Cache Size** | The maximum number of entries that the cache can hold. |
| **Used Entries** | The number of cache entries used. |
| **Cache Hits** | The number of request responses that are served from the cache. |
| **Cache Misses** | The number of request responses that are cacheable but cannot be served from the cache. |
| **LRU Evictions** | The number of cache entries removed to make room for new cache entries. |
| **Explicit Removals** | The number of cache entries removed or invalidated from the cache based on cache policies or were deleted from the cache through the cache monitor. |

2. You can also view the following cache configuration values:

| Cache configuration value | Description |
|---|---|
| **Default priority** | Specifies the default priority for all cache entries. Lower priority entries are moved from the cache before higher priority entries when the cache is full. You can specify the priority for individual cache entries in the cache policy. |

| Cache configuration value | Description |
|---|---|
| **Servlet Caching Enabled** | If servlet caching is enabled, results from servlets and JavaServer Pages (JSP) files are cached. See the *Administering applications and their environment* PDF for more information. |
| **Disk Offload Enabled** | Specifies if entries that are being removed from the cache are saved to disk. See the *Administering applications and their environment* PDF for more information. |

3. Wait for the application server to add data to the cache. You want the number of used cache entries in the cache monitor to be as high as it can go. When the number of used entries is at its highest, the cache can serve responses to as many requests as possible.
4. When the cache has a high number of used entries, reset the statistics. Watch the number of cache hits versus cache misses. If the number of hits is far greater than the number of misses, your cache configuration is optimal. You do not need to take any further actions. If you find a higher number of misses with a lower number of hits, the application server is working hard to generate responses instead of serving the request using a cached value. The application server might be making database queries, or running logic to respond to the requests.
5. If you have a large number of cache misses, increase the number of cache hits by improving the probability that a request can be served from the cache.

   To improve the number of cache hits, you can increase the cache size or configure additional cache policies. See the *Administering applications and their environment* PDF for more information to increase the cache size and to configure cache policies.

By using the cache monitor application, you optimized the performance of the dynamic cache service.

See the *Administering applications and their environment* PDF for more information about the dynamic cache.

***Dynamic cache MBean statistics:***

The dynamic cache service provides an MBean interface to access cache statistics.

**Access cache statistics with the MBean interface, using JACL**

- Obtain the MBean identifier with the **queryNames** command, for example:
  ```
  $AdminControl queryNames type=DynaCache,*  // Returns a list of the available dynamic cache MBeans
  ```
  Select your dynamic cache MBean and run the following command:
  ```
  set mbean <dynamic_cache_mbean>
  ```
- Retrieve the names of the available cache statistics:
  ```
  $AdminControl invoke $mbean getCacheStatisticNames
  ```
- Retrieve the names of the available cache instances:
  ```
  $AdminControl invoke $mbean getCacheInstanceNames
  ```
- Retrieve all of the available cache statistics for the base cache instance:
  ```
  $AdminControl invoke $mbean getAllCacheStatistics
  ```
- Retrieve all of the available cache statistics for the named cache instance:
  ```
  $AdminControl invoke $mbean getAllCacheStatistics "services/cache/servletInstance_4"
  ```
- Retrieve cache statistics that are specified by the names array for the base cache instance:
  ```
  $AdminControl invoke $mbean getCacheStatistics
   {"DiskCacheSizeInMB ObjectsReadFromDisk4000K RemoteObjectMisses"}
  ```

  **Note:** This command should all be entered on one line. It is broken here for printing purposes.
- Retrieve cache statistics that are specified by the names array for the named cache instance:

```
$AdminControl invoke $mbean getCacheStatistics
 {services/cache/servletInstance_4 "ExplicitInvalidationsLocal CacheHits"}
```

**Note:** This command should all be entered on one line. It is broken here for printing purposes.

***Accessing dynamic cache PMI counters:***

The dynamic cache statistics interface is defined as WSDynamicCacheStats under the `com\ibm\websphere\pmi\stat` package.

Dynamic cache statistics are structured as follows in the Performance Monitoring Infrastructure (PMI) tree:

```
__Dynamic Caching+
  |
  |__<Servlet: instance_1>
    |__Templates+
      |__<template_1>
      |__<template_2>
    |__Disk+
      |__<Disk Offload Enabled>
  |
  |__<Object: instance_2>
    |__Object Cache+
      |__<Counters>
+ indicates logical group
```

StatDescriptor locates and accesses particular statistics in the PMI tree. For example:

1. StatDescriptor to represent statistics for cache servlet: `instance_1 templates group template_1: new StatDescriptor (new String[] {WSDynamicCacheStats.NAME, ″Servlet: instance1″, WSDynamicCacheStats.TEMPLATE_GROUP, ″template_1″});`
2. StatDescriptor to represent statistics for cache servlet: `instance_1 disk group Disk Offload Enabled: new StatDescriptor (new String[] {WSDynamicCacheStats.NAME, ″Servlet: instance_1″, WSDynamicCacheStats.DISK_GROUP, WSDynamicCacheStats.DISK_OFFLOAD_ENABLED});`
3. StatDescriptor to represent statistics for cache object: `instance2 object cache group Counters: new StatDescriptor (new String[] {WSDynamicCacheStats.NAME, ″Object: instance_2″, WSDynamicCacheStats.OBJECT_GROUP, WSDynamicCacheStats.OBJECT_COUNTERS});`

**Important:** Cache instance names are prepended with cache type (″Servlet: ″ or ″Object: ″).

### Counter definitions for Servlet Cache

| Name of PMI statistics | Path | Description | Version |
|---|---|---|---|
| WSDynamicCacheStats. MaxInMemoryCache EntryCount | `WSDynamicCacheStats.NAME - "Servlet: instance_1"` | The maximum number of in-memory cache entries. | 5.0 and later |
| WSDynamicCacheStats. InMemoryCache EntryCount | `WSDynamicCacheStats.NAME - "Servlet: instance_1"` | The current number of in-memory cache entries | 5.0 and later |
| WSDynamicCacheStats. HitsIn MemoryCount | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"` | The number of requests for cacheable objects that are served from memory. | 5.0 and later |

| Name of PMI statistics | Path | Description | Version |
|---|---|---|---|
| WSDynamicCacheStats. HitsOnDiskCount | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"` | The number of requests for cacheable objects that are served from disk. | 5.0 and later |
| WSDynamicCacheStats. ExplicitInvalidationCount | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"` | The number of explicit invalidations. | 5.0 and later |
| WSDynamicCacheStats. LruInvalidationCount | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1` | The number of cache entries that are removed from memory by a Least Recently Used (LRU) algorithm. instance. | 5.0 and later |
| WSDynamicCacheStats. TimeoutInvalidationCount | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"` | The number of cache entries that are removed from memory and disk because their timeout has expired. | 5.0 and later |
| WSDynamicCacheStats. InMemoryAndDisk CacheEntryCount | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"` | The current number of used cache entries in memory and disk. | 5.0 and later |
| WSDynamicCacheStats. RemoteHitCount | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"` | The number of requests for cacheable objects that are served from other Java virtual machines within the replication domain. | 5.0 and later |
| WSDynamicCacheStats. MissCount | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"` | The number of requests for cacheable objects that were not found in the cache. | 5.0 and later |
| WSDynamicCacheStats. ClientRequestCount | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"` | The number of requests for cacheable objects that are generated by applications running on this application server. | 5.0 and later |
| WSDynamicCacheStats. DistributedRequestCount | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"` | The number of requests for cacheable objects that are generated by cooperating caches in this replication domain. | 5.0 and later |

| Name of PMI statistics | Path | Description | Version |
|---|---|---|---|
| WSDynamicCacheStats. ExplicitMemory InvalidationCount | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"` | The number of explicit invalidations resulting in the removal of an entry from memory. | 5.0 and later |
| WSDynamicCacheStats. ExplicitDisk InvalidationCount | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"` | The number of explicit invalidations resulting in the removal of an entry from disk. | 5.0 and later |
| WSDynamicCacheStats. LocalExplicit InvalidationCount | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"` | The number of explicit invalidations generated locally, either programmatically or by a cache policy. | 5.0 and later |
| WSDynamicCacheStats. RemoteExplicit InvalidationCount | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"` | The number of explicit invalidations received from a cooperating Java virtual machine in this replication domain. | 5.0 and later |
| WSDynamicCacheStats. RemoteCreationCount | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. TEMPLATE_GROUP -"Template_1"` | The number of cache entries that are received from cooperating dynamic caches. | 5.0 and later |

| Name of PMI statistics | Path | Description | Version |
|---|---|---|---|
| WSDynamicCacheStats. ObjectsOnDisk | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The current number of cache entries on disk. | 6.1 |
| WSDynamicCacheStats. HitsOnDisk | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of requests for cacheable objects that are served from disk. | 6.1 |
| WSDynamicCacheStats. ExplicitInvalidations FromDisk | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of explicit invalidations resulting in the removal of entries from disk. | 6.1 |

| Name of PMI statistics | Path | Description | Version |
|---|---|---|---|
| WSDynamicCacheStats. TimeoutInvalidations FromDisk | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of disk timeouts. | 6.1 |
| WSDynamicCacheStats PendingRemoval FromDisk | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The current number of pending entries that are to be removed from disk. | 6.1 |
| WSDynamicCacheStats. DependencyIdsOnDisk | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The current number of dependency ID that are on disk. | 6.1 |
| WSDynamicCacheStats. DependencyIdsBuffered ForDisk | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The current number of dependency IDs that are buffered for the disk. | 6.1 |
| WSDynamicCacheStats. DependencyIds OffloadedToDisk | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of dependency IDs that are offloaded to disk. | 6.1 |
| WSDynamicCacheStats. DependencyIdBased InvalidationsFromDisk | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of dependency ID-based invalidations. | 6.1 |
| WSDynamicCacheStats. TemplatesOnDisk | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The current number of templates that are on disk. | 6.1 |
| WSDynamicCacheStats. TemplatesBuffered ForDisk | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The current number of templates that are buffered for the disk. | 6.1 |

| Name of PMI statistics | Path | Description | Version |
|---|---|---|---|
| WSDynamicCacheStats. TemplatesOffloaded ToDisk | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of templates that are offloaded to disk. | 6.1 |
| WSDynamicCacheStats. TemplateBased InvalidationsFromDisk | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of template-based invalidations. | 6.1 |
| WSDynamicCacheStats. GarbageCollector InvalidationsFromDisk | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of garbage collector invalidations resulting in the removal of entries from disk cache due to high threshold has been reached. | 6.1 |
| WSDynamicCacheStats. OverflowInvalidations FromDisk | `WSDynamicCacheStats.NAME - "Servlet: cache_instance_1 " - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of invalidations resulting in the removal of entries from disk due to exceeding the disk cache size or disk cache size in GB limit. | 6.1 |

## Counter definitions for Object Cache

| Name of PMI Statistics | Path | Description | Version |
|---|---|---|---|
| WSDynamicCacheStats. MaxInMemoryCache EntryCount | `WSDynamicCacheStats.NAME - "Object: instance_2"` | The maximum number of in-memory cache entries. | 5.0 and later |
| WSDynamicCacheStats. InMemoryCache EntryCount | `WSDynamicCacheStats.NAME - "Object: instance_2"` | The current number of in-memory cache entries | 5.0 and later |
| WSDynamicCacheStats. HitsInMemoryCount | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats.OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS` | The number of requests for cacheable objects that are served from memory. | 5.0 and later |
| WSDynamicCacheStats. HitsOnDiskCount | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS` | The number of requests for cacheable objects that are served from disk. | 5.0 and later |

| Name of PMI Statistics | Path | Description | Version |
|---|---|---|---|
| WSDynamicCacheStats. ExplicitInvalidationCount | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS` | The number of explicit invalidations. | 5.0 and later |
| WSDynamicCacheStats. LruInvalidationCount | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS` | The number of cache entries that are removed from memory by a Least Recently Used (LRU) algorithm. instance. | 5.0 and later |
| WSDynamicCacheStats. TimeoutInvalidation Count | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS` | The number of cache entries that are removed from memory and disk because their timeout has expired. | 5.0 and later |
| WSDynamicCacheStats. InMemoryAndDisk CacheEntryCount | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS` | The current number of used cache entries in memory and disk. | 5.0 and later |
| WSDynamicCacheStats. RemoteHitCount | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS` | The number of requests for cacheable objects that are served from other Java virtual machines within the replication domain. | 5.0 and later |
| WSDynamicCacheStats. MissCount | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS` | The number of requests for cacheable objects that were not found in the cache. | 5.0 and later |
| WSDynamicCacheStats. ClientRequestCount | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS` | The number of requests for cacheable objects that are generated by applications running on this application server. | 5.0 and later |
| WSDynamicCacheStats. DistributedRequest Count | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS` | The number of requests for cacheable objects that are generated by cooperating caches in this replication domain. | 5.0 and later |

| Name of PMI Statistics | Path | Description | Version |
|---|---|---|---|
| WSDynamicCacheStats. ExplicitMemory InvalidationCount | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS` | The number of explicit invalidations resulting in the removal of an entry from memory. | 5.0 and later |
| WSDynamicCacheStats. ExplicitDisk InvalidationCount | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS` | The number of explicit invalidations resulting in the removal of an entry from disk. | 5.0 and later |
| WSDynamicCacheStats. LocalExplicit InvalidationCount | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS` | The number of explicit invalidations generated locally, either programmatically or by a cache policy. | 5.0 and later |
| WSDynamicCacheStats. RemoteExplicit InvalidationCount | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS` | The number of explicit invalidations received from a cooperating Java virtual machine in this replication domain. | 5.0 and later |
| WSDynamicCacheStats. RemoteCreationCount | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. OBJECT_GROUP - WSDynamicCacheStats OBJECT_COUNTERS` | The number of cache entries that are received from cooperating dynamic caches. | 5.0 and later |

| Name of PMI statistics | Path | Description | Version |
|---|---|---|---|
| WSDynamicCacheStats. ObjectsOnDisk | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The current number of cache entries on disk. | 6.1 |
| WSDynamicCacheStats. HitsOnDisk | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of requests for cacheable objects that are served from disk. | 6.1 |

| Name of PMI statistics | Path | Description | Version |
|---|---|---|---|
| WSDynamicCacheStats. ExplicitInvalidations FromDisk | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of explicit invalidations resulting in the removal of entries from disk. | 6.1 |
| WSDynamicCacheStats. TimeoutInvalidations FromDisk | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of disk timeouts. | 6.1 |
| WSDynamicCacheStats PendingRemoval FromDisk | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The current number of pending entries that are to be removed from disk. | 6.1 |
| WSDynamicCacheStats. DependencyIdsOnDisk | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The current number of dependency ID that are on disk. | 6.1 |
| WSDynamicCacheStats. DependencyIds BufferedForDisk | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The current number of dependency IDs that are buffered for the disk. | 6.1 |
| WSDynamicCacheStats. DependencyIds OffloadedToDisk | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of dependency IDs that are offloaded to disk. | 6.1 |
| WSDynamicCacheStats. DependencyIdBased InvalidationsFromDisk | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats.DISK_ OFFLOAD_ENABLED` | The number of dependency ID-based invalidations. | 6.1 |
| WSDynamicCacheStats. TemplatesOnDisk | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The current number of templates that are on disk. | 6.1 |

| Name of PMI statistics | Path | Description | Version |
|---|---|---|---|
| WSDynamicCacheStats. TemplatesBuffered ForDisk | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP / -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The current number of templates that are buffered for the disk. | 6.1 |
| WSDynamicCacheStats. TemplatesOffloaded ToDisk | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of templates that are offloaded to disk. | 6.1 |
| WSDynamicCacheStats. TemplateBasedInvalidations FromDisk | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of template-based invalidations. | 6.1 |
| WSDynamicCacheStats. GarbageCollector InvalidationsFromDisk | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of garbage collector invalidations resulting in the removal of entries from disk cache due to high threshold has been reached. | 6.1 |
| WSDynamicCacheStats. OverflowInvalidations FromDisk | `WSDynamicCacheStats.NAME - "Object: cache_instance_2" - WSDynamicCacheStats. DISK_GROUP -" WSDynamicCacheStats. DISK_OFFLOAD_ENABLED` | The number of invalidations resulting in the removal of entries from disk due to exceeding the disk cache size or disk cache size in GB limit. | 6.1 |

## Work area

***Work area service performance considerations:*** The work area service is designed to address complex data passing patterns that can quickly grow beyond convenient maintenance. A *work area* is a note pad that is accessible to any client that is capable of looking up Java Naming Directory Interface (JNDI). After a work area is established, data can be placed there for future use in any subsequent method calls to both remote and local resources.

You can utilize a work area when a large number of methods require common information or if information is only needed by a method that is significantly further down the call graph. The former avoids the need for complex parameter passing models where the number of arguments passed becomes excessive and hard to maintain. You can improve application function by placing the information in a work area and subsequently accessing it independently in each method, eliminating the need to pass these parameters from method to method. The latter case also avoids unnecessary parameter passing and helps to improve performance by reducing the cost of marshalling and de-marshalling these parameters over the Object Request Broker (ORB) when they are only needed occasionally throughout the call graph.

When attempting to maximize performance by using a work area, cache the UserWorkArea partition that is retrieved from JNDI wherever it is accessed. You can reduce the time spent looking up information in JNDI by retrieving it once and keeping a reference for the future. JNDI lookup takes time and can be costly.

Additional caching mechanisms available to a user-defined partition are defined by the configuration property, ″Deferred Attribute Serialization″. This mechanism attempts to minimize the number of serialization and deserialization calls. See Work area partition service for further explanation of this configuration attribute.

The maxSendSize and maxReceiveSize configuration parameters can affect the performance of the work area. Setting these two values to 0 (zero) effectively turns off the policing of the size of context that can be sent in a work area. This action can enhance performance, depending on the number of nested work areas an application uses. In applications that use only one work area, the performance enhancement might be negligible. In applications that have a large number of nested work areas, there might be a performance enhancement. However, a user must note that by turning off this policing it is possible that an extremely large amount of data might be sent to a server.

Performance is degraded if you use a work area as a direct replacement to passing a single parameter over a single method call. The reason is that you incur more overhead than just passing that parameter between method calls. Although the degradation is usually within acceptable tolerances and scales similarly to passing parameters with regard to object size, consider degradation a potential problem before utilizing the service. As with most functional services, intelligent use of the work areas yields the best results.

The work area service is a tool to simplify the job of passing information from resource to resource, and in some cases can improve performance by reducing the overhead that is associated with a parameter passing when the information is only sparsely accessed within the call graph. Caching the instance retrieved from JNDI is important to effectively maximize performance during runtime.

# Chapter 7. Troubleshooting performance

This topic illustrates that solving a performance problem is an iterative process and shows how to troubleshoot performance problems.

Solving a performance problem is frequently an iterative process of:
- Measuring system performance and collecting performance data
- Locating a bottleneck
- Eliminating a bottleneck

This process is often iterative because when one bottleneck is removed the performance is now constrained by some other part of the system. For example, replacing slow hard disks with faster ones might shift the bottleneck to the CPU of a system.

**Measuring system performance and collecting performance data**

Begin by choosing a *benchmark*, a standard set of operations to run. This benchmark exercises those application functions experiencing performance problems. Complex systems frequently need a warm-up period to cache objects, optimize code paths, and so on. System performance during the warm-up period is usually much slower than after the warm-up period. The benchmark must be able to generate work that warms up the system prior to recording the measurements that are used for performance analysis. Depending on the system complexity, a warm-up period can range from a few thousand transactions to longer than 30 minutes.

If the performance problem under investigation only occurs when a large number of clients use the system, then the benchmark must also simulate multiple users. Another key requirement is that the benchmark must be able to produce repeatable results. If the results vary more than a few percent from one run to another, consider the possibility that the initial state of the system might not be the same for each run, or the measurements are made during the warm-up period, or that the system is running additional workloads.

Several tools facilitate benchmark development. The tools range from tools that simply invoke a URL to script-based products that can interact with dynamic data generated by the application. IBM Rational has tools that can generate complex interactions with the system under test and simulate thousands of users. Producing a useful benchmark requires effort and needs to be part of the development process. Do not wait until an application goes into production to determine how to measure performance.

The benchmark records throughput and response time results in a form to allow graphing and other analysis techniques. The performance data that is provided by WebSphere Application Server Performance Monitoring Infrastructure (PMI) helps to monitor and tune the application server performance. Request metrics is another sources of performance data that is provided by WebSphere Application Server. Request metrics allows a request to be timed at WebSphere Application Server component boundaries, enabling a determination of the time that is spent in each major component. For more information about PMI and request metrics, see the *Administering applications and their environment* PDF.

**Locating a bottleneck**

Consult the following scenarios and suggested solutions:
- **Scenario:** Poor performance occurs with only a single user.

  **Suggested solution:** Utilize request metrics to determine how much each component is contributing to the overall response time. Focus on the component accounting for the most time. Use Tivoli Performance Viewer to check for resource consumption, including frequency of garbage collections. You might need code profiling tools to isolate the problem to a specific method. See the *Administering applications and their environment* PDF for more information.

- **Scenario:** Poor performance only occurs with multiple users.

  **Suggested solution:** Check to determine if any systems have high CPU, network or disk utilization and address those. For clustered configurations, check for uneven loading across cluster members.
- **Scenario:** None of the systems seems to have a CPU, memory, network, or disk constraint but performance problems occur with multiple users.

  **Suggested solutions:**
  - Check that work is reaching the system under test. Ensure that some external device does not limit the amount of work reaching the system. Tivoli Performance Viewer helps determine the number of requests in the system.
  - A thread dump might reveal a bottleneck at a synchronized method or a large number of threads waiting for a resource.
  - Make sure that enough threads are available to process the work both in IBM HTTP Server, database, and the application servers. Conversely, too many threads can increase resource contention and reduce throughput.
  - Monitor garbage collections with Tivoli Performance Viewer or the verbosegc option of your Java virtual machine. Excessive garbage collection can limit throughput.

**Eliminating a bottleneck**

Consider the following methods to eliminate a bottleneck:
- Reduce the demand
- Increase resources
- Improve workload distribution
- Reduce synchronization

Reducing the demand for resources can be accomplished in several ways. Caching can greatly reduce the use of system resources by returning a previously cached response, thereby avoiding the work needed to construct the original response. Caching is supported at several points in the following systems:
- IBM HTTP Server
- Command
- Enterprise bean
- Operating system

Application code profiling can lead to a reduction in the CPU demand by pointing out hot spots you can optimize. IBM Rational and other companies have tools to perform code profiling. An analysis of the application might reveal areas where some work might be reduced for some types of transactions.

Change tuning parameters to increase some resources, for example, the number of file handles, while other resources might need a hardware change, for example, more or faster CPUs, or additional application servers. Key tuning parameters are described for each major WebSphere Application Server component to facilitate solving performance problems. Also, the performance advisors can provide advice on tuning a production system under a real or simulated load.

Workload distribution can affect performance when some resources are underutilized and others are overloaded. WebSphere Application Server workload management functions provide several ways to determine how the work is distributed. Workload distribution applies to both a single server and configurations with multiple servers and nodes.

See the *Administering applications and their environment* PDF for more information.

Some critical sections of the application and server code require synchronization to prevent multiple threads from running this code simultaneously and leading to incorrect results. Synchronization preserves correctness, but it can also reduce throughput when several threads must wait for one thread to exit the critical section. When several threads are waiting to enter a critical section, a thread dump shows these threads waiting in the same procedure. Synchronization can often be reduced by: changing the code to only use synchronization when necessary; reducing the path length of the synchronized code; or reducing the frequency of invoking the synchronized code.

**Additional references**

WebSphere Application Server V6 Scalability and Performance Handbook

WebSphere Application Server Performance Web site

All SPEC jAppServer2004 Results Published by SPEC.

# Appendix. Directory conventions

References in the product information to *app_server_root*, *profile_root*, and other directories imply specific directory locations. This topic describes the conventions in use for WebSphere Application Server for z/OS.

**smpe_root**
Refers to the root directory for product code installed with SMP/E.

The corresponding product variable is smpe.install.root.

The default is /usr/lpp/zWebSphere/V6R1.

**configuration_root**
Refers to the mount point for the configuration file system (formerly, the configuration HFS) in WebSphere Application Server for z/OS.

The configuration_root contains the various app_server_root directories and certain symbolic links associated with them. Each different configuration_root normally requires its own cataloged procedures under z/OS.

The default is /WebSphere/V6R1.

**app_server_root**
Refers to the top directory for a WebSphere Application Server node.

The node may be of any type—application server, deployment manager, or unmanaged for example. Each node has its own app_server_root. Before Version 6.0 of the product information, this was referred to as the ″WAS_HOME″ directory. Corresponding product variables are was.install.root and WAS_HOME.

The default varies based on node type. Common defaults are *configuration_root*/Appserver and *configuration_root*/DeploymentManager.

**profile_root**
Refers to the home directory for a particular instantiated WebSphere Application Server profile.

Corresponding product variables are server.root and user.install.root.

In general, this is the same as *app_server_root*/profiles/*profile_name*. On z/OS, this will be always be *app_server_root*/profiles/default because only the profile name ″default″ is used in WebSphere Application Server for z/OS.

# Notices

## Trademarks and service marks

For trademark attribution, visit the IBM Terms of Use Web site (http://www.ibm.com/legal/us/).