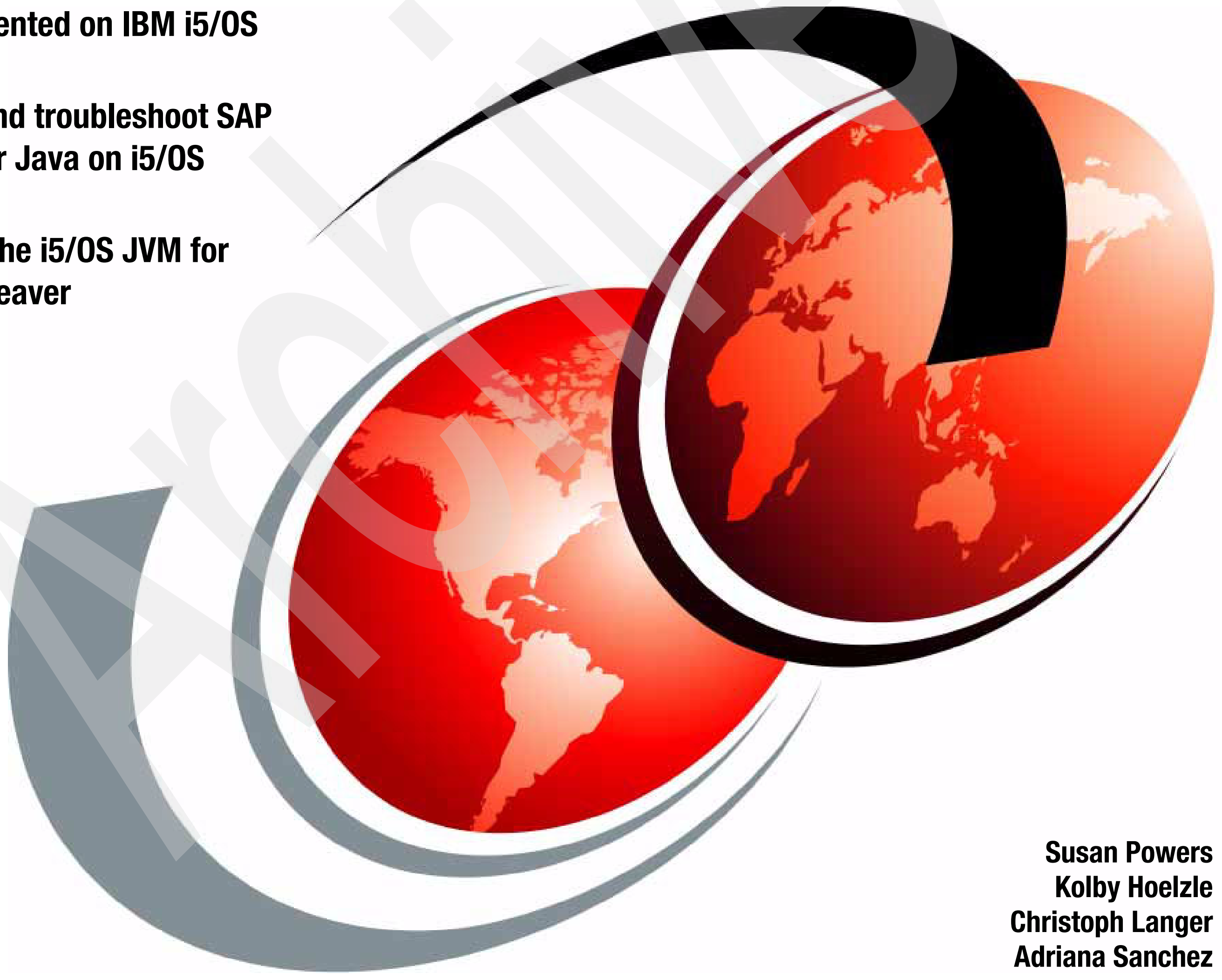
SAP NetWeaver Java on IBM i5/OS

Understand how SAP NetWeaver Java is implemented on IBM i5/OS

Analyze and troubleshoot SAP NetWeaver Java on i5/OS

Optimize the i5/OS JVM for SAP NetWeaver



Susan Powers
Kolby Hoelzle
Christoph Langer
Adriana Sanchez

Archived

Redpaper

ibm.com/redbooks

Archived

International Technical Support Organization

SAP NetWeaver Java on IBM i5/OS

November 2006

Archived

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

Archived

First Edition (November 2006)

This edition applies to Version 5 Release 4 of IBM i5/OS (product number 5722-SS1) and SAP WebAS 7.00 based on SAP NetWeaver04®.

© Copyright International Business Machines Corporation 2006. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Archived

Archived

Notices

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

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Archived

Trademarks

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

AS/400®	iSeries™	Redbooks (logo) ™
DB2®	i5/OS®	System i™
Eserver®	IBM®	WebSphere®
eServer™	OS/400®	Workplace™
ibm.com®	Redbooks™	

The following terms are trademarks of other companies:

ABAP, SAP NetWeaver, SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

SAP NetWeaver Developer Studio, ABAP, SAP NetWeaver, and SAP are the trademarks or registered trademarks of SAP AG in Germany and in several other countries.

SAP, SAP WebAS 7.00 based on SAP NetWeaver04®, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data contained in this document serve informational purposes only. National product specifications may vary.

EJB, Java, JavaSoft, JDBC, JDK, JSP, JVM, J2EE, Sun, Sun Microsystems, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

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

Archived

Preface

IBM® i5/OS® and System i™ technology are proven platforms for SAP® applications. With more than 10 years of success across more than 2500 SAP worldwide installations in small, medium, and large enterprises, System i models are an ideal platform for SAP customers who are looking for easy usability, high performance, reliability, and carefree operation of their SAP applications. The unique value of System i lies in its ability to reduce information technology (IT) complexity and to simplify SAP system landscapes.

This IBM Redpaper focuses on SAP Java™ technology. It can be used in conjunction with the IBM Redbook *Implementing SAP Applications on the IBM System i Platform with IBM i5/OS*, SG24-7166. This publication explores areas that are specific to the implementation and integration of the SAP Web Application Server for Java on i5/OS. Included in this document is an overview of the SAP Application Server architecture and how it is implemented on i5/OS. It also includes a discussion on Java and the i5/OS Classic JVM™, followed by configuration and tuning recommendations and finally how to analyze problems.

We wrote this Redpaper to assist SAP basis consultants and other information technology (I/T) professionals in implementing a successful SAP system installation based on SAP Java technology.

Archived

The team that wrote this Redpaper

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

Susan Powers is a Consulting IT Specialist at the ITSO, Rochester Center. Prior to joining the ITSO in 1997, she was an AS/400® Technical Advocate in the IBM Support Center with a variety of communications, performance, and work management assignments. Her IBM career began as a Program Support Representative and Systems Engineer in Des Moines, Iowa. She holds a degree in mathematics, with an emphasis in education, from St. Mary's College of Notre Dame. She is a project manager in the System i ITSO in Rochester, Minnesota.

Kolby Hoelzle is a Staff Software Engineer on the SAP on System i team, which is part of the i5/OS development lab in Rochester, Minnesota. He joined IBM in 1999 and has six years of experience with SAP on the System i platform, including two years working at SAP development in Walldorf, Germany as a member of the joint IBM SAP i5/OS porting team.

Christoph Langer is a member of the joint IBM/SAP porting team for SAP applications on the System i platform and Team Lead of the group working on porting the SAP Java solution stack. He has five years of experience in the field of SAP applications on System i models. He holds a bachelor's degree in Information Technology Management from the BA Stuttgart, Germany.

Adriana Sanchez is a developer for WebSphere® Serviceability tools in Rochester, Minnesota. She has two years of experience with SAP on the System i platform in the areas of performance, three-tier, and Java. Before joining the SAP team, Adriana worked on Linux® on Power Performance. She holds a degree in Computer Science from Florida International University.

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

Dan Gateno
IBM Rochester, Minnesota

Dorothea Rink
SAP, Germany

Ron Schmerbauch
IBM Rochester, Minnesota

Craig Stancl
IBM Rochester, Minnesota

Become a published author

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

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

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

`ibm.com/redbooks/residencies.html`

Comments welcome

Your comments are important to us!

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

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

  `ibm.com/redbooks`

- Send your comments in an e-mail to:

  `redbooks@us.ibm.com`

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099

Archived

2455 South Road
Poughkeepsie, NY 12601-5400

Archived

Archived

1

Architecture

SAP made it clear that Java is strategic to their development. The new SAP applications are based on the Java 2 Platform Enterprise Edition (J2EE™) model. Because of this shift from Advanced Business Application Programming (ABAP™) based development to Java and J2EE based development, it is important for you to understand the SAP Java Application Server and the underlying IBM i5/OS Java technology that runs it.

SAP applications run on a technology layer that is adapted to the various operating systems and platforms that SAP supports. Traditionally, SAP ABAP applications run on a technology layer based on C and C++. Many of the latest SAP applications, such as Enterprise Portals (EP) and Exchange Infrastructure (XI) are built on a technology layer based on Java. Even though some Java-based SAP applications have the similar look and feel of some of the traditional SAP applications, the underlying technology is vastly different. Because of this difference in technology, the Java-based SAP administrative tasks such as configuration, tuning, and troubleshooting differ from the C and C++-based administrative tasks. This Redpaper highlights these differences.

Archived

1.1 Java technology

Java technology consists of the following three basic components:

- Java language
- Java packages
- Java Virtual Machine (JVM)

These three components provide the foundation for all Java applications from the simple "Hello World" program to complex enterprise class applications.

The Java programming language is an object-oriented language with syntax largely based on C++. However, in Java as opposed to C++, nearly everything is an object. Every Java object is compiled into a Java executable called a Java class.

Java packages are Java classes that are packaged together for the convenience of developers. Java packages are similar in purpose to C or C++ libraries. Packages contain commonly-used classes and the methods or functions to access and utilize these classes. A developer imports a Java package in order to utilize these ready-to-use classes in his or her own program. Java packages increase efficiency by encouraging code reuse.

Perhaps the biggest difference between Java and traditional languages, such as C or C++, is the manner in which Java is executed. Traditional programming languages are compiled into executable programs that run directly on an operating system. Java code is compiled into a platform-independent Java executable called a class.

A Java class is made up of Java bytecodes that are executed on a Java Virtual Machine (JVM). The JVM runs directly on the operating system. This allows you to write and compile a Java program on one platform and run the same Java program on a completely different platform without making any changes. You can run Java classes on any platform as long as a compliant JVM exists. This makes Java extremely portable and provides "write-once, run anywhere" capability.

1.2 Java 2 Platform Enterprise Edition

Java 2 Platform Enterprise Edition (J2EE) is a standard for developing multi-tier enterprise applications based on Java technology. J2EE simplifies development of enterprise applications by using a set of standardized, modular components and by providing a complete set of services to these components. This allows J2EE to automatically handle many details of application behavior without the need for complex programming.

Following are some of the standardized components of J2EE:

- Data access using Java Database Connectivity (JDBC™)
- Security model
- Server-based applications with Java servlets
- Encapsulated business logic with Enterprise Java Beans (EJB™)
- Document generation with Java Server Pages (JSP™)

The J2EE standard includes complete specifications and compliance tests to ensure portability across all operating systems that support the J2EE platform.

SAP developed a J2EE application server that implements the J2EE standard. SAP refers to their J2EE application server as the Java Application Server. It is available as part of the SAP

Archived

Web Application Server (WebAS). SAP WebAS can run both traditional SAP ABAP applications and applications based on the J2EE standard, which are written in Java.

The SAP Java Application Server includes the services and tools required to deploy J2EE applications and it includes the J2EE engine itself which is required to run J2EE applications.

1.3 SAP Java cluster

The SAP Java cluster installation consists of the following items:

- One central service instance
- One or more instances of the SAP Java Application Server
- One database

You can have a Java-only system that only supports Java and J2EE applications. Alternately, you can have an SAP WebAS Add-In (ABAP Application Server and Java Application Server) installation that allows you to run both ABAP and J2EE applications.

The simplest SAP Java cluster installation consists of a Java central instance, a central services instance, and a database. A Java instance has a dispatcher that is responsible for dispatching client requests and a server node that receives the request from the dispatcher and executes the request. The Software Deployment Manager (SDM) is also part of a Java instance. SDM allows you to deploy and manage software packages distributed by SAP. It can run in integrated mode as part of the Java instance or in stand-alone mode. A Java instance has a system ID (SID) and an instance number. Multiple Java instances can exist in an SAP Java cluster installation. In an SAP WebAS Add-in system, the Java instance and the ABAP instance are combined into one instance.

The central services instance is a special Java instance that contains the message service and the enqueue service. The central services instance is responsible for providing communication and synchronization in the Java cluster installation. The message server is used for communication between all the elements of a Java cluster. The dispatcher, for example, obtains the information about which processes are active from the message service. The enqueue server is used for synchronization within the Java cluster by managing logical locks. The central service instance is identified by the system ID (SID) of the Java cluster installation, but like any instance, it has its own instance number. One central services instance is required for each SAP Java cluster installation.

The Java cluster installation, like ABAP, uses the integrated DB2® UDB for i5/OS database. The DB2 UDB for i5/OS is a full relational database system integrated in the operating system. i5/OS performs storage management. Because it is integrated in the operating system, the database does not compete with other applications like on other platforms.

Archived

Figure 1-1 shows a simple installation of the SAP Java cluster.

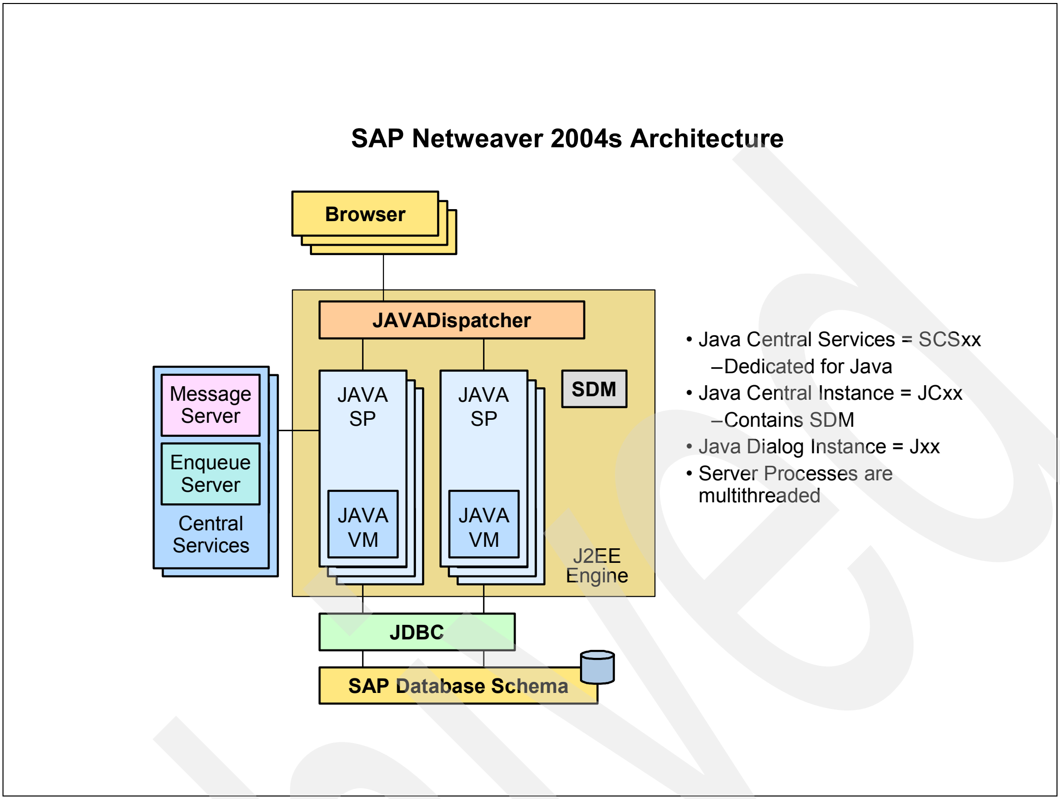


Figure 1-1   Simple SAP Java cluster installation

Archived

2

The Java Virtual Machine

Unlike the SAP ABAP Application Server, the SAP Java Application Server does not run directly on an operating system. Since the SAP Java Application Server is programmed in Java itself, it runs on a Java Virtual Machine (JVM). JVM in turn runs directly on the operating system.

Because a JVM is required for the SAP Java Application Server to function, it is important to understand some of the characteristics of the JVM and its impact to the SAP Java Application Server. Currently only the IBM i5/OS Classic JVM is supported by SAP for the SAP Java Application Server on i5/OS.

Archived

2.1 IBM i5/OS Classic JVM

On i5/OS, all applications and the base operating system are based on an architecture known as technology independent machine interface (TIMI), or more simply referred to as the machine interface (MI). When a program runs on i5/OS, it presents the instructions to the MI for execution. The instructions presented to the MI pass in turn through a layer of microcode known as System Licensed Internal Code (SLIC), before they are executed by the hardware.

The i5/OS Classic JVM was first available on OS/400® V4R2. It is implemented within SLIC. The Classic JVM is available on all i5/OS installations. Because the Classic JVM does most of its work below the MI layer, the interpretation and execution of the Java code on i5/OS is very fast. The implementation of the classic JVM on i5/OS provides improved scalability compared to other Java platforms.

Because the i5/OS Classic JVM is integrated into the operating system, all Java classes must be verified. There is overhead associated with Java class verification; however, a class verification cache mechanism (discussed in 4.4.2, “Activating the class verification cache for all JVMs” on page 25) significantly improves the performance of this process.

The Java Development Kits (JDKs), which provide the base Java packages, compilers, and other commands, are located above the MI layer like any other program on i5/OS. You can get versions of the JDK™ by installing the licensed program 5722-JV1.

Figure 2-1 shows a representation of the IBM i5/OS Classic JVM and JDK implementation.

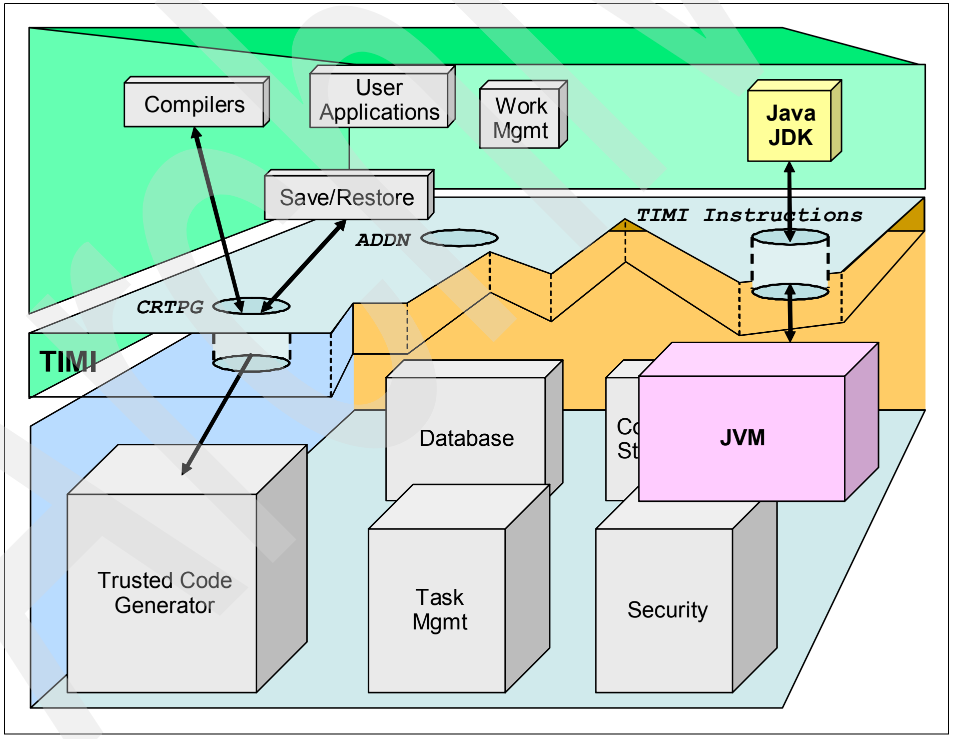


Figure 2-1 Implementation of the IBM i5/OS Classic JVM and JDK

With the i5/OS Classic JVM, Java programs are optimized using the Just-In-Time (JIT) compiler. A JIT compiler is a platform-specific compiler that generates machine instructions for Java code that is used very often. These machine instructions exist only as long as the JVM that created them runs.

The JIT mode is much faster compared to the interpreted mode that interprets each Java bytecode at runtime. In section 4.5, "IBM i5/OS Classic JVM system properties" on page 26 we discuss how a JVM property can be set to run in JIT mode and improve runtime performance.

Although Java is platform independent, the JVM where Java bytecode is executed is not. In addition, each JVM has a set of Java system properties you can set to modify the behavior of the JVM. These system properties determine the environment on which the Java programs run. They are similar to system values or environment variables in i5/OS. Starting a JVM sets the system properties for that instance of the JVM.

2.2 JVM Garbage collection

Java stores all objects created by a Java application in an area of memory called the heap. Java garbage collection is the mechanism of freeing unused objects so that the heap space is made available for new objects. The garbage collector (GC) is responsible for determining which objects are no longer referenced by the Java application and freeing the associated memory in the heap.

Garbage collection in Java makes Java different from other programming languages like C and C++ where memory allocation and deallocation is the responsibility of the developer. This places a larger burden on the developer and introduces the risk of memory leaks, which eventually lead to programs crashing. Even though Java garbage collection reduces the probability of memory management problems, it does comes with the cost of more CPU consumption.

The garbage collector is part of the JVM and runs automatically as needed. The GC runs in cycles. In the i5/OS Classic JVM, the frequency of the GC cycles and the cycle times are determined by a threshold specified by the initial heap size parameter (-Xms).

The i5/OS Classic JVM garbage collector has the following characteristics:

- Asynchronous: Runs in the background
- Concurrent: Runs concurrent to the other threads on the system
- Multithreaded: Has multiple threads working together to free the unused objects.

These characteristics have many advantages, including improved performance. The GC can run simultaneously with your application without stopping the application threads and causing long pauses. The disadvantage is that it could be more difficult to determine when the garbage collector is not properly tuned.

For specific details on how to tune the Classic JVM garbage collector, refer to section 4.6, "Fine tuning the IBM i5/OS Classic JVM" on page 28.

Archived

Archived

3

Implementation of the SAP Java Application Server in i5/OS

The implementation of the SAP Java Application Server varies from platform to platform due to differences within operating systems. This section describes what the SAP Java Application Server looks like when installed on i5/OS.

Archived

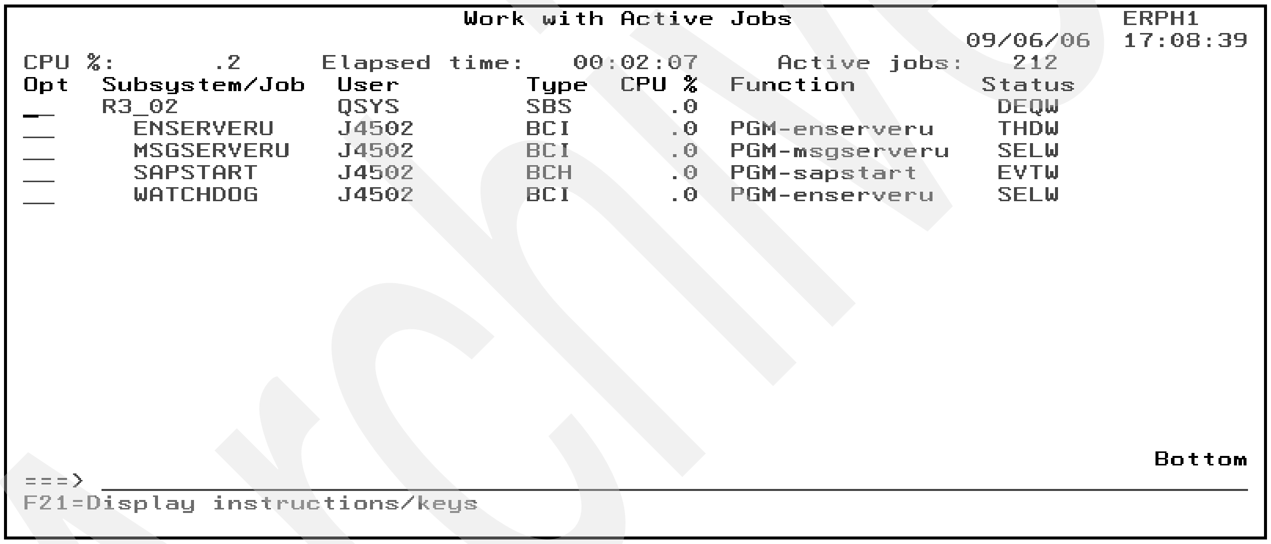
3.1 SAP jobs

On i5/OS, each SAP instance is implemented as a subsystem with a name R3_<nn>, where <nn> corresponds to the instance number. There are at least two instances associated with an SAP Java Application Server: one central services instance and one or more Java instances. The central services instance, which contains the message and enqueue services as described in section 1.3, “SAP Java cluster” on page 3, must be running before starting the Java instance. Therefore, the STARTSAP command must be executed first for the central services instance, and secondly for the Java instance.

When you run the following Work with Active Jobs command: **`WRKACTJOB SBS(R3_<nn>)`**, where <nn> is the instance number of a central services instance, the display shows the SAP jobs for the central services instance running in an i5/OS subsystem. For a central service instance, these jobs correspond to the following SAP processes:

- ENSERVER: enqueue server
- MSGSERVER: message server
- SAPSTART: startup job

Figure 3-1 shows the i5/OS subsystem running a central services instance.



```
                               Work with Active Jobs                         ERPH1
                                                                   09/06/06  17:08:39
CPU %:       .2     Elapsed time:   00:02:07      Active jobs:   212
Opt  Subsystem/Job  User         Type  CPU %  Function          Status
__   R3_02          QSYS         SBS      .0                      DEQW
__     ENSERVERU    J4502        BCI      .0  PGM-enserveru       THDW
__     MSGSERVERU   J4502        BCI      .0  PGM-msgserveru      SELW
__     SAPSTART     J4502        BCH      .0  PGM-sapstart        EVTW
__     WATCHDOG     J4502        BCI      .0  PGM-enserveru       SELW










                                                                            Bottom
===> ____________________________________________________________________________
F21=Display instructions/keys
```

Figure 3-1 SAP jobs in the subsystem for a central services instance

For an SAP Java instance, these jobs correspond to the following SAP processes:

- DISPATCHER: Dispatcher
- JCONTROL: Control framework
- SAPSTART: Startup job
- SDM: Software deployment manager
- SERVER<n>: Server node where <n> is the number of the server node

A Java instance can have more than one server node to execute client requests. Each server node is multi-threaded and can process requests simultaneously. The number of server nodes in an instance and the number of threads are configurable through the SAP J2EE Engine - Config Tool (commonly referred to as the Config Tool). Each server node is a Java process and runs in its own JVM. Similarly, the dispatcher and SDM processes are multithreaded and run in their own JVM.

Archived

The Java Startup and Control framework starts, stops, and monitors a Java instance. The JControl process is the job responsible for starting, stopping, and monitoring the processes of a Java instance. Following is the process:

- The STARTSAP command first starts the SAP subsystem (instance) and submits a background job to this subsystem that runs the SAP start program (SAPSTART).
- The SAPSTART program starts the JControl program that starts the bootstrap jobs to synchronize the binary data of the Java database with the local file system.
- The SAPSTART program starts the SAP services and processes associated with each instance. In an Add-In configuration, the dispatcher starts JControl.

The Internet Graphic Service (IGS) jobs are responsible for enabling the application developer to display graphics in an Internet browser with a minimum of effort. They provide a server architecture where data from an SAP system or another source can be used to generate graphical or non-graphical output.

Figure 3-2 shows the jobs for the SAP Java Application Server.

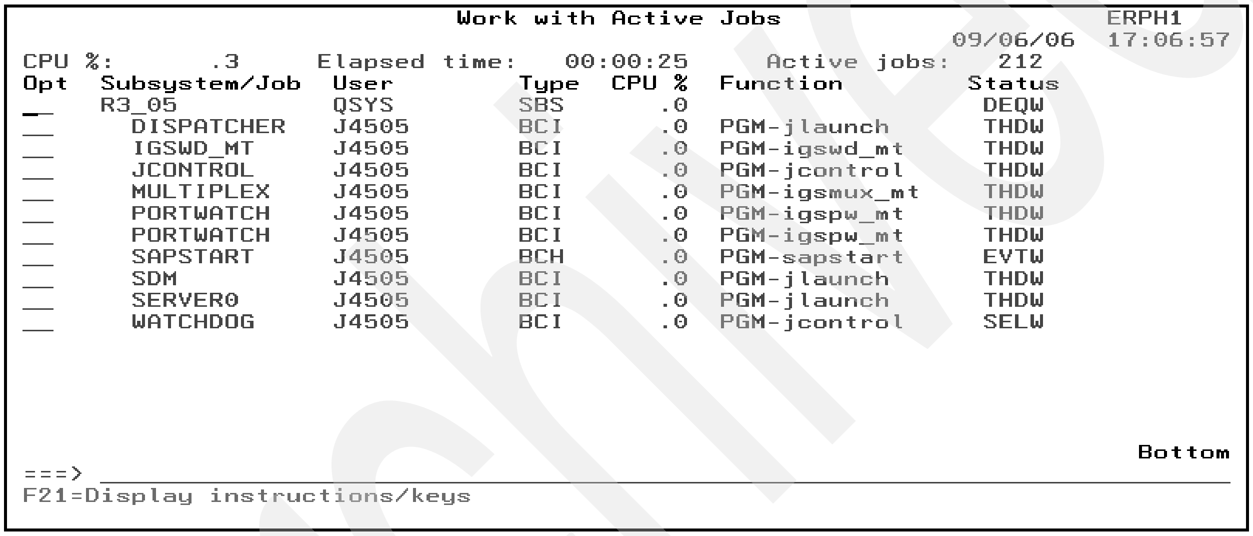

```
                            Work with Active Jobs                    ERPH1
                                                         09/06/06  17:06:57
CPU %:      .3     Elapsed time:   00:00:25     Active jobs:   212
Opt  Subsystem/Job  User         Type  CPU %  Function        Status
__   R3_05          QSYS         SBS      .0                   DEQW
__     DISPATCHER   J4505        BCI      .0  PGM-jlaunch      THDW
__     IGSWD_MT     J4505        BCI      .0  PGM-igswd_mt     THDW
__     JCONTROL     J4505        BCI      .0  PGM-jcontrol     THDW
__     MULTIPLEX    J4505        BCI      .0  PGM-igsmux_mt    THDW
__     PORTWATCH    J4505        BCI      .0  PGM-igspw_mt     THDW
__     PORTWATCH    J4505        BCI      .0  PGM-igspw_mt     THDW
__     SAPSTART     J4505        BCH      .0  PGM-sapstart     EVTW
__     SDM          J4505        BCI      .0  PGM-jlaunch      THDW
__     SERVER0      J4505        BCI      .0  PGM-jlaunch      THDW
__     WATCHDOG     J4505        BCI      .0  PGM-jcontrol     SELW




                                                                     Bottom
===> ____________________________________________________________________
F21=Display instructions/keys
```

Figure 3-2 SAP jobs in the Java instance subsystem

In an Add-In SAP Web AS (ABAP and Java Engine) installation, the processes associated with the J2EE engine run as part of the ABAP instance. In this case, all of the work processes and the dispatcher together with all the J2EE engine processes run under the same subsystem.

3.2 Integrated File System

The file system structure for the SAP Java Application Server is very similar to that of an SAP ABAP Application Server, with the two main SAP directory trees being /sapmnt and /usr/sap. Like an ABAP Application Server, the Integrated File System (IFS) directories for a Java Application Server installation contain many objects, including: executables, configuration data, logs, and standalone tools such as SAP J2EE Engine - Visual Administrator (commonly referred to as the Visual Administrator tool).

For a visual representation of the Integrated File System directory structure for an SAP Java Application Server, see Figure 3-3 on page 12.

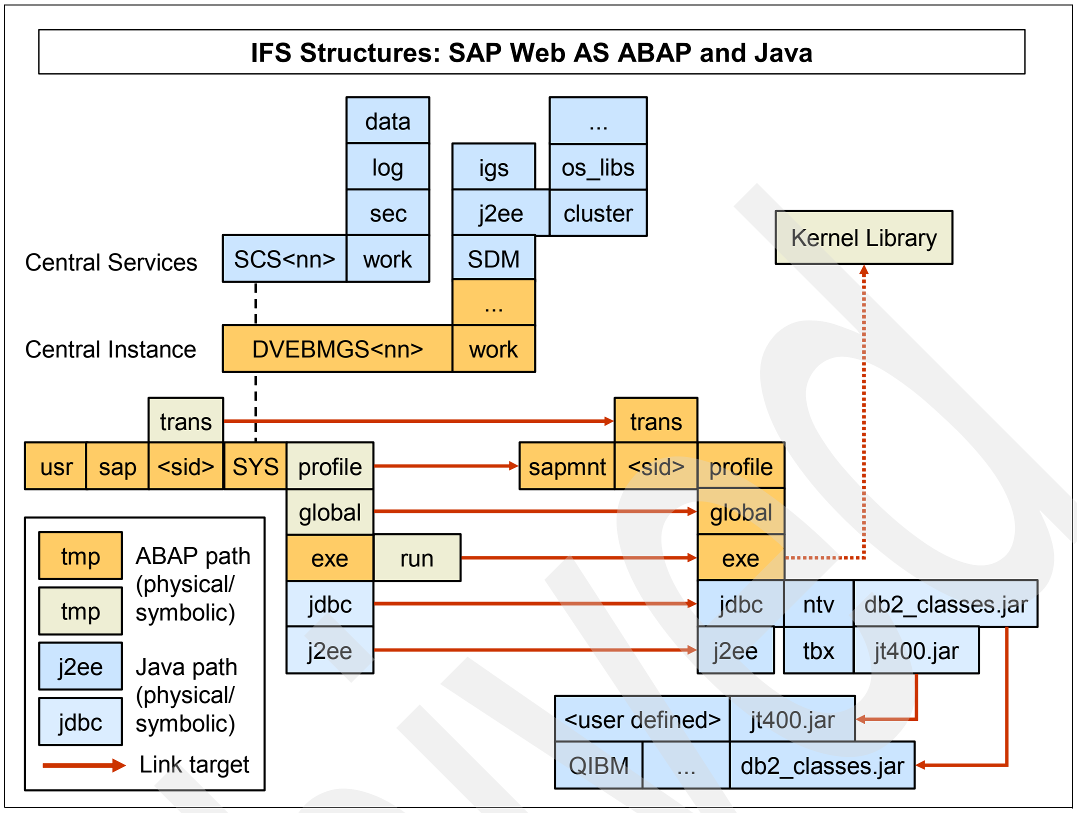


Figure 3-3 Integrated File System structures for the SAP Web Application Server (ABAP and Java)

Archived

The instance type and the instance number determine the instance name. In the case of a Java-only installation, the instance name consists of a prefix (*JC* or *J*) and the two-digit instance number afterwards. In an Add-In SAP Web AS (ABAP and Java Engine) installation the instance name consists of the prefix DVEBMGS and the two-digit instance number like any ABAP system.

The following list describes the most relevant Integrated File System paths for Java:

- /usr/sap/SID/SCSxx

  This is the Java central services instance directory (Java enqueue, and message server).
- /usr/sap/SID/<instance_name>/j2ee

  Contains all J2EE data for that instance including administration tools for Java (Visual Administrator, Config Tool) and log files.
- /usr/sap/SID/SYS/jdbc

  Contains links to the JDBC drivers that are used for database connectivity.
- /usr/sap/SID/SYS/global/security/data

  Contains the Secure Store.
- /usr/sap/sid/<instance_name>/SDM

  Contains data for the Software Deployment Manager (SDM).

The following list describes the relevant Integrated File System paths that are common to both Java and ABAP application servers:

- /usr/sap/SID/SYS/exe/run

Contains native executables from the SAP kernel and links to objects remaining in the SAP kernel library.

- /usr/sap/sid/<instance_name>/work

  Contains SAP trace files for all jobs under the instance.
- /usr/sap/SID/profile

  Contains SAP instance profiles consisting of characteristics of an instance.
- /usr/sap/sid/<instance_name>/igs

  Contains data for the Internet Graphic Server.

For a description of the i5/OS Integrated File System and the SAP directory structure on i5/OS, refer to section 8.2.4 “The Integrated File System” in the IBM Redbook *Implementing SAP Applications on the IBM System i Platform with IBM i5/OS*, SG24-7166.

3.3 Database

An SAP WebAS systems must contain at least one database schema. If ABAP and Java are installed then two databases are installed. The naming of the databases differ depending on the type of WebAS installation.

Depending on the installation type, the following libraries exist on your i5/OS system:

- For SAP WebAS Java systems:
  - SAP<SID>DB: Data library for Java.
  - SAP<SID>JRN: Journal receiver library that is associated with library SAP<SID>DB.
- For SAP WebAS ABAP+Java systems (Add-In):
  - R3<SID>DATA: Data library for ABAP.
  - R3<SID>JRN: Journal receiver library that is associated with R3<SID>DATA.
  - SAP<SID>DB: Data library for Java.
  - SAP<SID>JRN: Journal receiver library that is associated with library SAP<SID>DB.

For more information about SAP databases refer to section 9.5 “Basic principles of SAP databases and SAP systems” in the IBM Redbook *Implementing SAP Applications on the IBM System i Platform with IBM i5/OS*, SG24-7166.

3.4 Database access

The SAP Java Application Server accesses DB2 UDB for i5/OS using Java Database Connectivity (JDBC) drivers. JDBC is an application programming interface (API) developed to provide a standardized method of accessing a database from a Java application. The JDBC API was developed by Sun™ Microsystems™ subsidiary JavaSoft™ and provides the specifications for executing structured query language (SQL) statements from Java programs on any database that supports SQL. Since implementations of SQL standards vary from database to database, the database vendors and other third parties usually provide optimized JDBC drivers specific to a particular database.

Archived

3.4.1 Types of JDBC drivers

The SAP Java Application Server supports two types of JDBC drivers for DB2 UDB for i5/OS data access: type 2 and type 4 JDBC drivers. See SAP note 654800 “iSeries™: FAQ: JDBC driver certified for SAP on iSeries” for more information about SAP-certified JDBC drivers.

Type 2 JDBC driver

A type 2 JDBC driver uses native operating system code in conjunction with Java to access the database. The i5/OS type 2 JDBC driver is commonly called the Native JDBC driver or simply the Native Driver. The Native Driver uses the i5/OS call level interface (CLI) to access the database. The native JDBC driver requests are handled by QSQSRVR jobs. These jobs are pre-start jobs and are managed by i5/OS.

The Native Driver is highly optimized for high performance Java data access. Since the Native Driver is not pure Java and uses i5/OS native code, it can only be used if both the Java application and the database are on the same i5/OS partition. The Native Driver is shipped with the Developer Kit for Java, product 5722-JV1.

Type 4 JDBC driver

Type 4 JDBC drivers are pure Java drivers and as such can run on any operating system platform running a compliant JVM. The i5/OS type 4 JDBC driver is commonly called the Toolbox Driver because it is shipped with the Toolbox for Java, product 5722-JC1. The Toolbox Driver connects to DB2 UDB for i5/OS using socket connections. The Toolbox Driver requests are handled by the i5/OS host server. The host server jobs are named QZDASOINIT or QZDASSINIT if secure socket layer (SSL) is used. These jobs are pre-start jobs and are managed by i5/OS.

IBM provides and fully supports an open source version of the Toolbox Driver called the JTOpen driver. It contains the latest enhancement to the Toolbox Driver and enables you to acquire the latest version of the Toolbox Driver without waiting for the next service release or operating system release of Toolbox for Java. Because the JTOpen and Toolbox Drivers both have the same codebase, they are both commonly referred to as the Toolbox Driver. We recommend that you use the latest version of JTOpen with the SAP Java Application Server.

3.4.2 Choosing a JDBC driver

Advantages and disadvantages exist for both the native driver and the Toolbox JDBC Driver. Choosing which driver to use depends on your landscape.

The Native Driver offers performance and integration advantages over the Toolbox Driver; however, the Native Driver is not as flexible since it can only run on i5/OS. The Native Driver does support remote database access from i5/OS to i5/OS, but this is not supported by SAP for SAP applications.

The Toolbox Driver is more flexible than the Native Driver since it is pure Java. It can be used both for local database access and remote database access, including access from non-i5/OS clients to DB2 UDB for i5/OS. This flexibility comes at the cost of performance. In general the Toolbox Driver does not perform as well as the Native Driver.

If your landscape is 3-tier, the Toolbox Driver is your only choice regardless of the type of application server. In a 2-tier landscape, you can use either the Toolbox or the Native Driver.

If the Native Driver is used for the SAP Java Application Server, the Toolbox Driver is still required. Some remote tools, such as the Config Tool, require the Toolbox Driver to access

Archived

DB2 UDB for i5/OS. For more information see SAP note 657117 “iSeries: JDBC configuration of config tool”.

In a 2-tier landscape it is possible to switch JDBC drivers after installation. For more information about changing JDBC drivers for the SAP Java Application Server see SAP note 826449 “iSeries: Changing the JDBC driver”.

3.4.3 The JDBC URL

When a Java application uses a JDBC driver to connect to a database, the application needs to know which JDBC driver to use. Also, the JDBC driver needs to know to which database to connect.

A Java application has the flexibility to use different JDBC drivers to connect to different databases. In some cases, different JDBC drivers can be used to connect to the same database. Because of this flexibility the application must know which JDBC driver to use. For each JDBC driver used by an application, the application must know the location of the JDBC driver class and then register the JDBC driver class with the application. This is done using the Config Tool.

The JDBC driver knows which database to access through a set of parameters called the JDBC URL. The JDBC URL consists of the following three distinct parts:

- JDBC driver protocol
- Datasource
- Connection properties

The JDBC driver protocol consists of two parts: the JDBC protocol and the JDBC driver subprotocol. The protocol is always jdbc. The subprotocol depends on the driver being used: db2iseries for the Native Driver or as400 for the Toolbox Driver.

The datasource consists of the name of the database host or the relational database name. Since the Native Driver is only used for local database access, the value *LOCAL can be used for the datasource.

Connection properties force the database and database components to behave a certain way. Multiple properties can be included in a JDBC URL. Each property is separated by a semi-colon (“;”).

Example 3-1 shows a URL for the Toolbox Driver, connecting to the datasource my_server and with a trace activated through a connection property.

Example 3-1

```
jdbc:as400://my_server;trace=true;
```

Example 3-2 shows a URL for the Native Driver, connecting to the local database, and with a trace activated through a connection property.

Example 3-2

```
jdbc:db2iseries:*LOCAL;trace=true
```

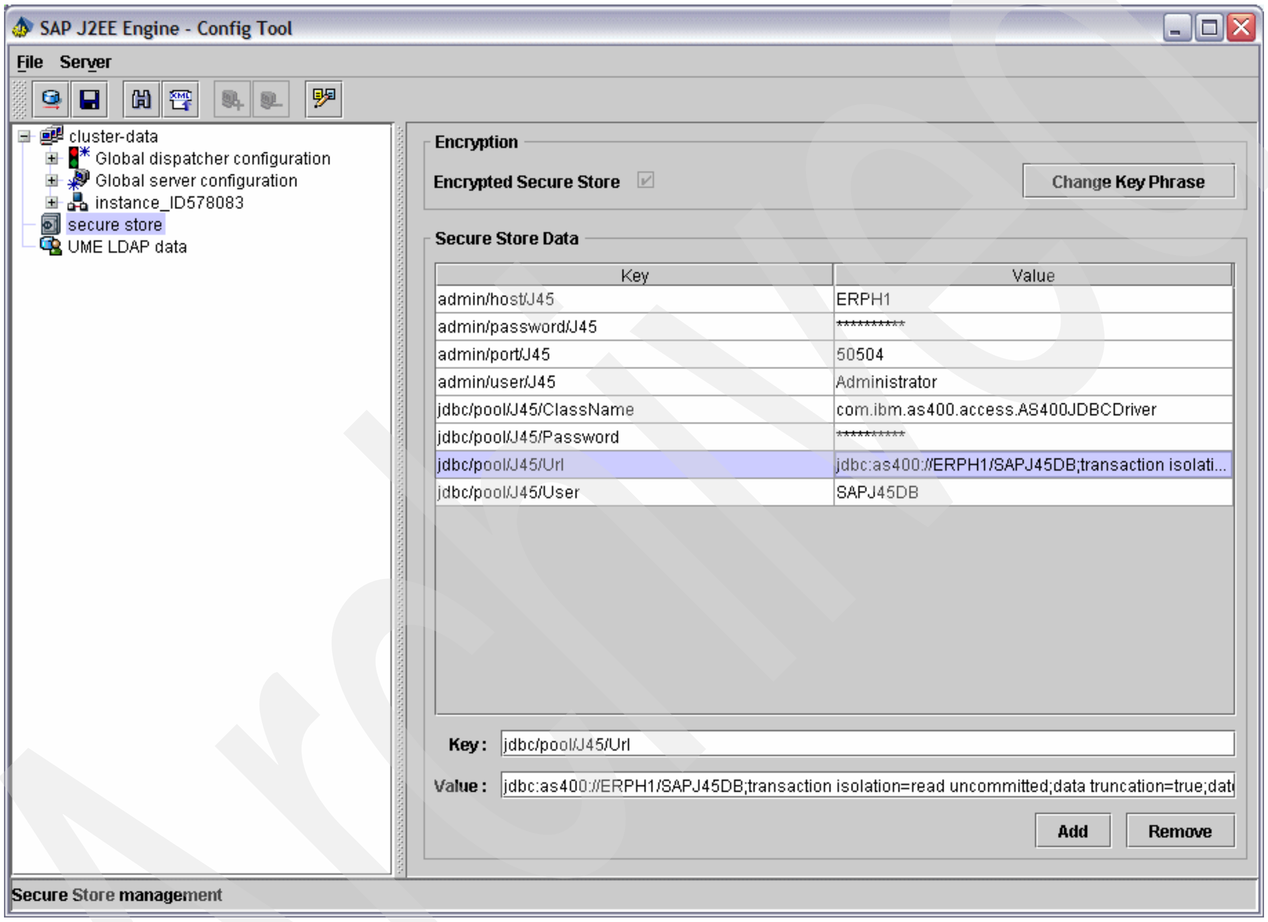
Archived

Changing JDBC URL

To change the JDBC URL, use the following steps:

1. Open the Config Tool.
2. Select secure store.
3. Select jdbc/pool/<sid>/Url.
4. Enter URL changes in the Value field.
5. Click **Add**.
6. Click **Save**.

The Config Tool is illustrated in Figure 3-4.



Key	Value
admin/host/J45	ERPH1
admin/password/J45	**********
admin/port/J45	50504
admin/user/J45	Administrator
jdbc/pool/J45/ClassName	com.ibm.as400.access.AS400JDBCDriver
jdbc/pool/J45/Password	**********
jdbc/pool/J45/Url	jdbc:as400://ERPH1/SAPJ45DB;transaction isolati...
jdbc/pool/J45/User	SAPJ45DB

Key: jdbc/pool/J45/Url

Value: jdbc:as400://ERPH1/SAPJ45DB;transaction isolation=read uncommitted;data truncation=true;dat

Figure 3-4 Changing the JDBC URL (©SAP AG 2006. All rights reserved.)

For more information about recommended JDBC connection properties for the SAP Java Application Server see SAP note 654800 “iSeries: FAQ: JDBC driver certified for SAP on iSeries”.

Archived

4

Configuration and tuning

To maximize hardware resources and to provide acceptable performance for end users, you must properly configure and tune the application stack. The application stack includes the application(s), Java Application Server, JVM, and the operating system. Configuring and tuning an application stack is dependent on many things, including the following:

- Type of applications
- Amount of throughput expected
- Currency of hardware
- Amount of available hardware resources
- Concurrent workloads

Because of these dependencies, configuration and tuning varies with the environment. What might work well in one situation might not be as optimal in another. The approach taken in this document is to first provide recommendations that provide a stable system with acceptable performance. Second, we provide guidelines and instructions on how to fine-tune the JVM for your particular environment. The focus of this IBM Redpaper is for platform dependent configuration and tuning only. For general SAP application configuration and tuning see the appropriate SAP documentation.

Most configuration and tuning options can be categorized two ways:

- Set and forget

  Set and forget options are generally set once during configuration and then never touched again. These options are not dependent on the workload of an application so changes to the workload have no impact. Most set and forget options are optimal for a particular application stack regardless of workload characteristics and hardware. Many of these options are set by default.

- Workload and hardware dependent

  Workload and hardware dependent options need to be adjusted occasionally if the characteristics of a workload change significantly or if hardware characteristics change. For example, adding 100 additional users and upgrading to a faster processor might necessitate adjusting workload and hardware dependent options, while set and forget options can remain untouched.

Archived

Due to the interdependencies between configuration and tuning and the impact that one step can have on another we recommend that you follow the instructions in this section in the order in which they appear.

4.1 SAP Java Application Server configuration

Most SAP Java Application Server configuration is platform independent. However, in certain areas consider the operating system, JVM, and the hardware when making configuration changes.

The following recommendations are i5/OS-specific. For other configurations consult SAP documentation.

- Number of server nodes
- Number of application threads per node

4.1.1 Server nodes

Server nodes are similar in function to work processes in an ABAP environment; however, they are very different in structure. Both server nodes and work processes are responsible for performing the real work done by an SAP system. Both receive that work from a dispatcher.

Whereas a work process is a single-threaded job that can only handle one task at a time, a server node is a Java Virtual Machine (JVM) that by its nature is multi-threaded and can handle multiple tasks at one time. Because of the multi-threaded nature of the server node, the number of server nodes in an instance is quite small compared to the number of work processes in an ABAP instance doing the same amount of work.

Since additional server nodes consume resources, do not add additional server nodes if they are not necessary. The number of server nodes required is dependent on the load or number of concurrent users.

Due to the scalability of the i5/OS Classic JVM, a high level of throughput can be achieved with a limited number of server nodes. Since each server node adds significant memory and CPU overhead, reducing the number of server nodes decreases the memory footprint and CPU utilization. A small number of server nodes also reduces the amount of configuration and maintenance required.

Number of server nodes

We recommend that you configure the number of server nodes based on the number of processors dedicated to or utilized by the SAP Java Application Server. For more information about determining the number of server nodes, see SAP note 717376 “iSeries: Recommended Settings for SAP WebAS Java”.

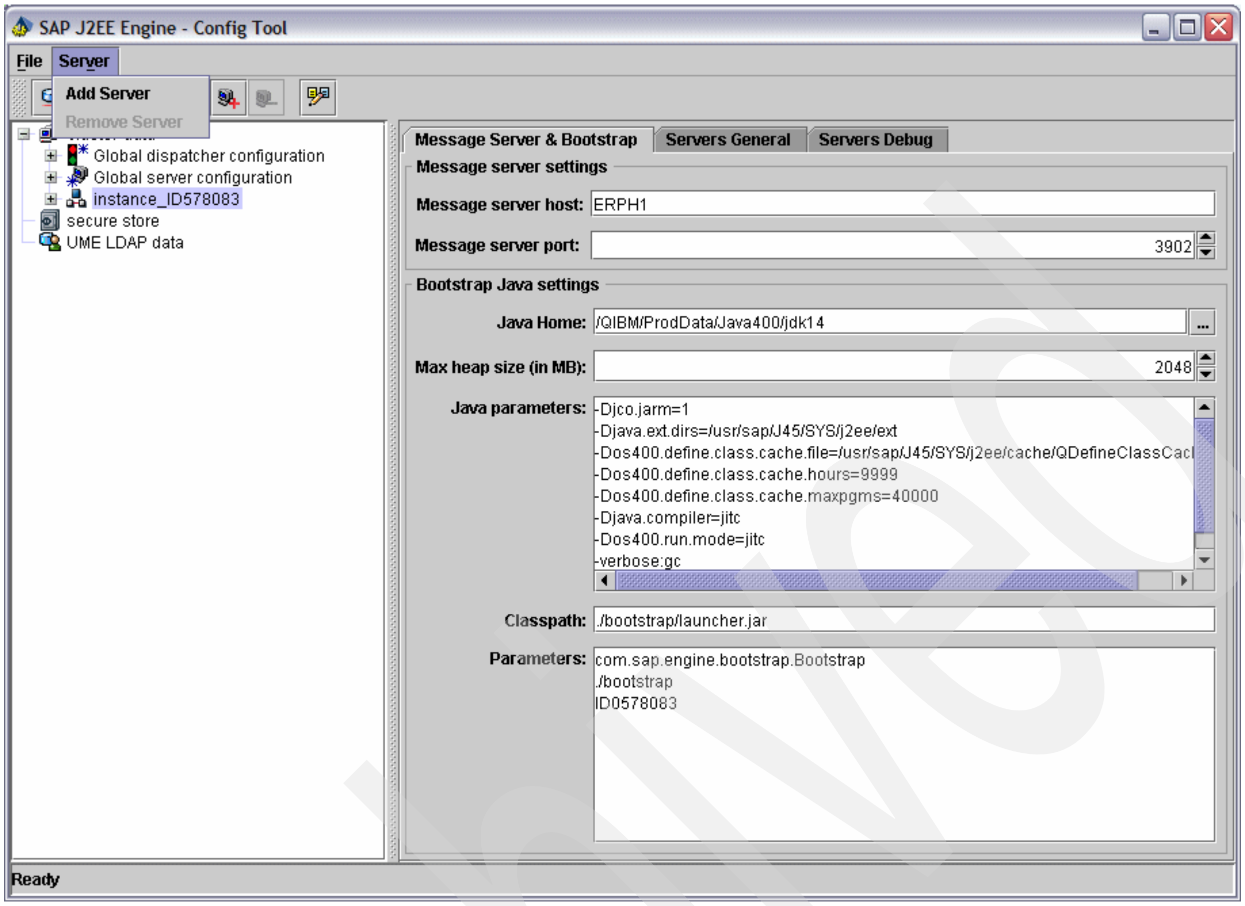
Adding Server Nodes

To add a server node, use the following steps:

1. Open the SAP J2EE Engine - Config Tool (commonly referred to as the Config Tool).
2. Select the instance where the server node is to be added.
3. From the menu select **Server** → **Add Server**.
4. Click **Yes** if prompted to add another server.
5. Click **Ok**.

Figure 4-1 on page 19 shows the Config Tool being used to add a server node.

Archived


SAP J2EE Engine - Config Tool

Message server settings

Message server host: ERPH1

Message server port: 3902

Bootstrap Java settings

Java Home: /QIBM/ProdData/Java400/jdk14

Max heap size (in MB): 2048

Java parameters:
```
-Djco.jarm=1
-Djava.ext.dirs=/usr/sap/J45/SYS/j2ee/ext
-Dos400.define.class.cache.file=/usr/sap/J45/SYS/j2ee/cache/QDefineClassCac
-Dos400.define.class.cache.hours=9999
-Dos400.define.class.cache.maxpgms=40000
-Djava.compiler=jitc
-Dos400.run.mode=jitc
-verbose:gc
```

Classpath: ./bootstrap/launcher.jar

Parameters:
```
com.sap.engine.bootstrap.Bootstrap
./bootstrap
ID0578083
```

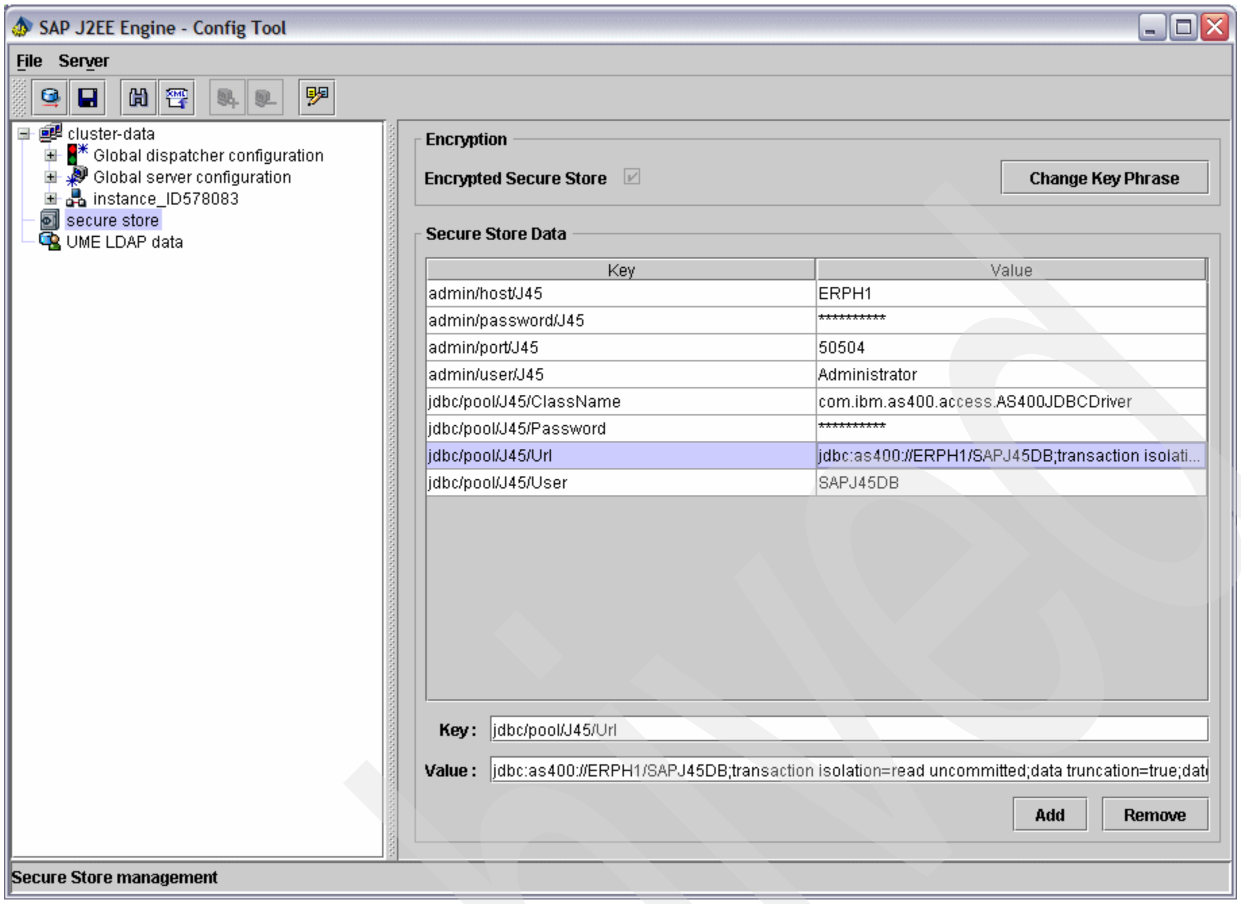
Figure 4-1 Adding a server node (©SAP AG 2006. All rights reserved.)

If your SAP Java Application Server is configured to use the Native JDBC driver, change the driver location of the newly created server node. Use the following steps:

1. Open the Config Tool.
2. Expand the new server node.
3. Expand managers.
4. Select ConfigurationManager.
5. Select rdbms.driverLocation.
6. Enter the location of the Native JDBC driver jar file.
7. Click **Set**.
8. Click **Save**.

Figure 4-2 on page 20 shows the Config Tool being used to change the location of the JDBC driver.

Archived



Encrypted Secure Store ✓

Secure Store Data

Key	Value
admin/host/J45	ERPH1
admin/password/J45	**********
admin/port/J45	50504
admin/user/J45	Administrator
jdbc/pool/J45/ClassName	com.ibm.as400.access.AS400JDBCDriver
jdbc/pool/J45/Password	**********
jdbc/pool/J45/Url	jdbc:as400://ERPH1/SAPJ45DB;transaction isolati...
jdbc/pool/J45/User	SAPJ45DB

Key : jdbc/pool/J45/Url

Value : jdbc:as400://ERPH1/SAPJ45DB;transaction isolation=read uncommitted;data truncation=true;dat

Figure 4-2 Changing the JDBC driver location of a new server node (©SAP AG 2006. All rights reserved.)

4.1.2 Application threads per server node

A server node in the SAP Java Application Server is a multi-threaded job that carries out the real work of the SAP Java Application Server. Each thread can be thought of as a light-weight job.

The server node application threads are responsible for completing the work submitted by the end user. The number of application threads in a server node is configurable and determines the amount of work an individual server node can process at one time. Since increasing the number of threads increases resource usage and might cause resource contention (locking), it is important to set the highest number of threads possible without a negative impact on the system.

Number of application threads per server node

The number of application threads per server node is determined by the load on the SAP Java Application Server. The more load or users supported by the application server, the higher the number of application threads.

For more information about configuring the number of application threads per node see SAP note 717376 “iSeries: Recommended Settings for SAP WebAS Java”.

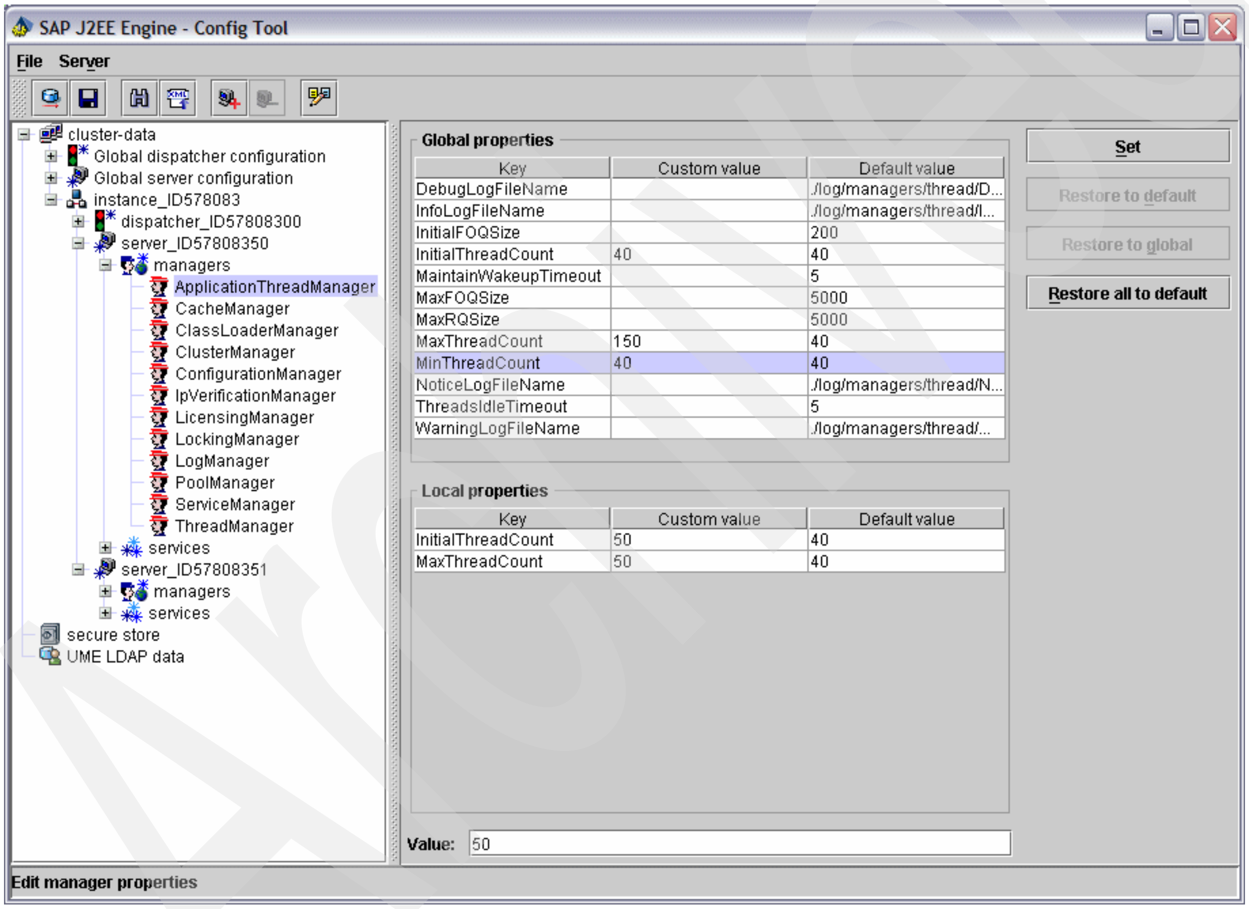
Archived

Changing the number of threads per node

Use the following steps to change the number of threads per node:

1. Open the Config Tool.
2. Expand the instance.
3. Expand the server node.
4. Expand managers.
5. Select ApplicationThreadManager.
6. Select InitialThreadCount.
7. Change value.
8. Click **Set**.
9. Click **Save**.
10. Repeat these steps for MaxThreadCount and for MinThreadCount.

Figure 4-3 shows the Config Tool being used to adjust the number of application threads.



- cluster-data
  - Global dispatcher configuration
  - Global server configuration
  - instance_ID578083
    - dispatcher_ID57808300
    - server_ID57808350
      - managers
        - ApplicationThreadManager
        - CacheManager
        - ClassLoaderManager
        - ClusterManager
        - ConfigurationManager
        - IpVerificationManager
        - LicensingManager
        - LockingManager
        - LogManager
        - PoolManager
        - ServiceManager
        - ThreadManager
      - services
    - server_ID57808351
      - managers
      - services
  - secure store
  - UME LDAP data

Global properties

Key	Custom value	Default value
DebugLogFileName		./log/managers/thread/D...
InfoLogFileName		./log/managers/thread/I...
InitialFOQSize		200
InitialThreadCount	40	40
MaintainWakeupTimeout		5
MaxFOQSize		5000
MaxRQSize		5000
MaxThreadCount	150	40
MinThreadCount	40	40
NoticeLogFileName		./log/managers/thread/N...
ThreadsIdleTimeout		5
WarningLogFileName		./log/managers/thread/...

Local properties

Key	Custom value	Default value
InitialThreadCount	50	40
MaxThreadCount	50	40

Value: 50

Edit manager properties

Figure 4-3 Adjusting the number of application threads (©SAP AG 2006. All rights reserved.)

4.2 Minimum hardware requirements

Application configuration, such as the number of server nodes and the number of application threads, have an impact on both memory and processor usage. Since the application configuration is needed to determine the memory needs, the section discussing minimum hardware requirements follows the section describing SAP Java Application Server configuration.

Archived

The minimum hardware requirements are the minimum amount of hardware resources required to meet the demand of a particular workload. This includes processors, memory, disk storage, communication devices, and so on.

Only physical memory and processors are discussed in this document. No amount of configuration or tuning can fully compensate for not meeting minimum hardware requirements. All applications require a certain amount of resources in order to operate.

Consider the following items when determining the resources available to an application:

- Running an application in an environment that does not meet minimum hardware requirements leads to an increased risk of functional and performance problems if the application runs at all.
- Resource requirements vary from application to application and vary within the same application depending on configuration, workload, and so on.

Physical memory

Memory requirements are dependent on many factors. Consider the following items when determining memory needs for the SAP Java Application Server:

- Memory requirements include the memory necessary for Java (GC, JIT, JVM), the operating system, and the application to start and function properly.
- Memory usage in the SAP J2EE environment is coupled tightly with the number of users (throughput).
- Memory usage increases as the number of users increase.
- Additional memory beyond minimum requirements is recommended to act as a buffer.

For more information about physical memory requirements see SAP note 717376 “iSeries: Recommended Settings for SAP WebAS Java”.

Processors

IBM OS/400 V5R2 is the minimum operating system release supported by SAP for their Java Application Server. All processors supported by IBM for V5R2 and later releases are sufficient to run the SAP Java Application Server. The number of processors required varies by workload.

For more information about processor requirements see SAP note 717376 “iSeries: Recommended Settings for SAP WebAS Java”.

4.3 Operating system configuration recommendations

There are many options for configuring i5/OS for running SAP applications. Describing all of the configuration options for i5/OS as it pertains to SAP applications is beyond the scope of this document, but the following options exist that are helpful and specific for SAP J2EE workloads:

- Shared Memory Pools
- Performance Adjuster
- Max Active

Archived

4.3.1 Shared memory pools

An i5/OS memory pool is a division of main or auxiliary storage. On i5/OS, all main storage can be divided into logical allocations called memory pools. There are two types of memory pools on i5/OS: shared memory pools and private memory pools. Multiple subsystems can share the shared memory pools. Private memory pools only allow one subsystem. Most SAP applications run in the *BASE memory pool by default.

The use of a garbage collector in the Java language causes Java applications to be more sensitive to inadequate memory than other languages such as C and C++. When the garbage collector runs, it needs to access all of the objects in the Java heap, even if the objects were not recently used. If the Java heap is not entirely in memory, the operating system must fault the pages back in memory during a garbage collection cycle. When there is an excessive amount of page faulting due to a large portion of the heap not being in memory, the garbage collector takes longer to run and the heap size continues to grow even as the application threads continue to do their work. This has the potential to result in bad performance and out of memory errors. Other jobs running in the same pool as the Java environment might also cause the Java heap to be paged out of memory, which could result in poor performance. In addition, Java applications can take memory resources away from other jobs, affecting their performance.

Because Java applications are very memory sensitive and variations in workloads exist, we recommend that you initially run each SAP Java Application Server in its own shared memory pool (not *BASE), especially for a production environment. This allows you to monitor and determine the memory usage characteristics of the workload, especially the JVM garbage collector. See section 4.6.1, “Analyzing the JVM garbage collector” on page 28 for more information. The amount of memory in the memory pool should initially meet the minimum memory recommendations, but can be reduced if memory is not being utilized.

Regardless of whether the SAP Java Application Server is running the base pool or in its own shared memory pool, the amount of memory available to the application server should always meet the minimum requirements as recommended or determined by monitoring the workload characteristics. This is especially important when many applications are running in the same memory pool.

For more details on memory pools and Java, refer to the “IBM eServer™ iSeries IBM Developer Kit for Java” publication, specifically the section “Tune Java program performance with the IBM Developer Kit for Java” in the iSeries Information Center.

Configuring shared memory pools in an SAP environment

Use the following steps to configure shared memory pool for an SAP environment:

1. From the i5/OS command line, type the WRKSHRPOOL command to view the shared pool settings on the system.
2. Select a pool, and set the initial size and activity level according to the SAP J2EE minimum requirements. Refer to SAP note 717376 “iSeries: Recommended Settings for SAP WebAS Java” for the minimum requirements.
3. Type the following command to assign the new memory pool to your SAP subsystem:

   ```
   CHGSBSD SBSD(R3<sid>400/R3_<nn>) POOLS((<pool id> *SHRPOOL<n>)
   ```

4. Type the following command to direct the SAP jobs to the new memory pool:

   ```
   CHGRTGE SBSD(R3<sid>400/R3_<nn>)
   SEQNBR(1)
   PGM(QSYS/QCMD)
   CLS(R3<sid>400/R3_<nn>)
   ```

Archived

```
POOLID(<pool id>)
THDRSCAFN(*SYSVAL)
RSCAFNGRP(*NO)
```

5. Restart the changed subsystem.

4.3.2 Performance adjuster

The performance adjuster allows i5/OS to automatically manage the shared memory pools without any user interaction. By default the performance adjuster is turned on and set to adjust memory pools automatically at Initial Program Load (IPL).

We recommend that you turn off the performance adjuster by setting the i5/OS system value QPFRADJ to 0.

It is important that the SAP Java Application Server always has the minimum required memory available. Running with less than the recommended minimum can lead to functional and performance problems.

The Performance Adjuster attempts to balance memory between memory pools based on historical data. In certain situations the Performance Adjuster can allocate less memory than is required for the SAP Java Application Server.

If the performance adjuster is set (the QPFRADJ system value is not set to 0), then the shared pool containing SAP Java Application Server must have a minimum size set according to the SAP system's minimum requirements.

The performance adjustment system value is QPFRADJ. When this system value is set to ‘2’ or ‘3,’ the system periodically checks the performance of all the active shared pools and adjusts or rearranges the storage and activity levels as needed.

The settings on the WRKSHRPOOL display affect the performance adjuster algorithm.

4.3.3 Max active

The activity level of a storage pool refers to the number of active threads within that storage pool. If the activity level is too low, a thread's context is paged out of main storage and marked as ineligible for a short time. This affects the performance of your application.

We recommend initially setting the activity level of a storage pool for an SAP J2EE installation to at least 500. Increase this value as necessary when threads are marked as ineligible. You can monitor the number of threads becoming ineligible by using the **WRKSYSSTS** command from the i5/OS command line, as follows:

1. From the i5/OS command line, type the **WRKSYSSTS** command.
2. Press F11 once to view the Wait → Inel and Active → Inel columns.

The resulting columns should have a value of 0. If the value is greater than 0, then increase the maximum active value (max active). Any adjustments to max active take place immediately and do not require restarting the application.

For more details see the memory pool activity level topic in the iSeries Information Center located at the following Web address:

http://publib.boulder.ibm.com/infocenter/iseries/v5rv/index.jsp

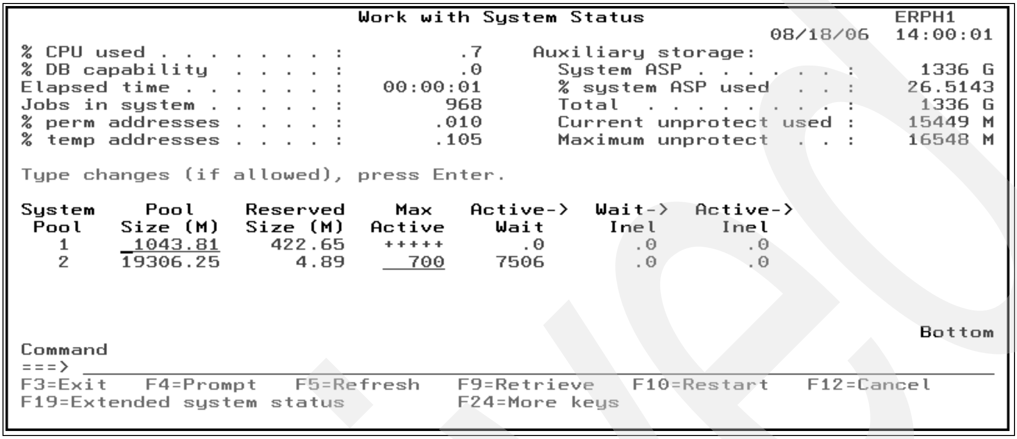
Archived

Changing Max Active

Use the following steps to change the maximum active value:

1. From the i5/OS command line, type the **WRKSYSSTS** command.
2. Change the activity level for a specific memory pool under the Max Active column.

Figure 4-4 shows the maximum active threads and the number of threads in each thread state using the Work with System Status (WRKSYSSTS) command.



```
                              Work with System Status                    ERPH1
                                                             08/18/06  14:00:01
% CPU used . . . . . . . :           .7    Auxiliary storage:
% DB capability  . . . . :           .0      System ASP . . . . . . :     1336 G
Elapsed time . . . . . . :     00:00:01      % system ASP used  . . :    26.5143
Jobs in system . . . . . :          968      Total  . . . . . . . . :     1336 G
% perm addresses . . . . :         .010      Current unprotect used :    15449 M
% temp addresses . . . . :         .105      Maximum unprotect  . . :    16548 M

Type changes (if allowed), press Enter.

System     Pool    Reserved     Max    Active->  Wait->  Active->
 Pool    Size (M)  Size (M)   Active     Wait     Inel     Inel
   1     _1043.81    422.65    +++++       .0       .0       .0
   2     19306.25      4.89   ___700     7506       .0       .0




                                                                         Bottom
Command
===> ______________________________________________________________________________
F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F10=Restart   F12=Cancel
F19=Extended system status         F24=More keys
```

Figure 4-4 Using WRKSYSSTS to monitor ineligible threads and to adjust max active

4.4 IBM i5/OS Classic JVM global properties

Global JVM system properties minimize configuration and maintenance by setting common JVM system properties in one place for all JVMs. Global JVM properties are set using the system default properties file. Any JVM system properties specified in the system default properties file are applied to all JVMs running in a partition. However, any or all JVM properties can be overridden by a JVM specific properties file or by command line parameters.

4.4.1 System default properties file

The system default properties file applies a value that needs to or should be applied to all JVMs in a partition. The system default properties file is named SystemDefault.properties and is located at /QIBM/UserData/Java400. You can update the file with Edit File (EDTF) command or any text editor with access to the Integrated File System.

4.4.2 Activating the class verification cache for all JVMs

Since the i5/OS Classic JVM is implemented in System Licensed Internal Code (SLIC), all executables must be verified to see if they are safe to run. For Java this means that every Java class must be verified. For a large Java application thousands of classes might need to be verified. All of this verification impacts startup time. To improve startup time activate a class verification cache.

The SAP Java Application Server is now installed with the class verification cache activated for individual JVMs. See SAP note 717376 “iSeries: Recommended Settings for SAP WebAS Java”. We still recommend that you activate the verification cache globally to improve the startup time of all JVMs running in a partition.

Archived

The following three properties must be used together in order to activate the class verification cache.

- **os400.define.class.cache.file=/QIBM/UserData/Java400/QDefineClassCache.jar**

  The name of a valid Java Archive (JAR) file that is used as a cache.

- **os400.define.class.cache.hours=9999**

  The amount of time (in hours) that a Java program object persists in the cache.

- **os400.define.class.cache.maxpgms=40000**

  The number of Java program objects that the cache can hold.

For more information about using caches for class loading, see the Java performance considerations on the iSeries Information Center.

4.4.3 Other global properties

Set the following global properties:

- **java.awt.headless=true**

  Specifies whether the Abstract Windowing Toolkit (AWT) API operates in headless mode or not. Headless mode is specified for servers that do not have a graphic subsystem, for example graphic card, monitor, and so on.

- **user.timezone=<local_timezone>**

  Specifies the time zone in which the JVM is running. Visit the following Web address for valid time zone values:

  http://publib.boulder.ibm.com/infocenter/wsdoc400/v6r0/index.jsp?topic=/com.ibm.websphere.iseries.doc/info/ae/ae/adrtzval.htm

4.5 IBM i5/OS Classic JVM system properties

JVM system properties change the behavior of a JVM. Most JVM system properties are *set and forget* and never need changing regardless of workload. Other properties *tune* the JVM and may occasionally need adjusting. Properties used for tuning can be impacted by the behavior of the application, workload, and other factors.

Describing all JVM system properties is beyond the scope of this Redpaper. The properties described here are important when running and tuning the SAP Java Application Server on i5/OS. Set each of the following properties for the individual SAP Java Application Server JVMs. For more information about other JVM system properties refer to the IBM i5/OS Information Center located at the following Web address:

http://publib.boulder.ibm.com/infocenter/iseries/v5rv/index.jsp

For more information about setting individual JVM properties for the SAP Java Application Server, refer to SAP note 717376 “iSeries: Recommended Settings for SAP WebAS Java”.

4.5.1 Set and forget properties

Set the following properties only once. They never need adjusting:

- **java.compiler=jitc and os400.run.mode=jitc**

Archived

Specifies whether code is compiled with the Just-In-Time (JIT) compiler. The property java.compiler overrides os400.run.mode.

- **os400.disable.explicity.gc**

  A Java program can explicitly invoke the JVM garbage collector. Invoking the garbage collector through the Java program directly instead of allowing the JVM to manage the garbage collector can have a negative impact on JVM performance. Disabling explicit garbage collection allows the JVM to run more efficiently.

- **class verification cache**

  Due to the large number of classes loaded by the server and dispatcher JVM, individually set the class verification cache for these JVMs. For a description of the class verification cache see section 4.4.2, “Activating the class verification cache for all JVMs” on page 25. For more details on setting the class verification cache for individual JVMs refer to SAP note 717376 “iSeries: Recommended Settings for SAP WebAS Java”.

4.5.2 Tuning properties

The following properties need to be adjusted occasionally. They are dependent on factors such as the application workload, hardware resources, and configuration:

- **Xmx (maximum heap size)**

  This parameter is the maximum size the heap is allowed to grow. The heap does not necessarily grow to the size set with this parameter. Ideally the heap should not grow to the maximum heap size value. Set the maximum heap size as a safety precaution in the unlikely event of abnormal heap growth. Under normal circumstances the heap should never grow to the size specified with maximum heap size. Setting this property depends on hardware, workload, and other variables. For more information about setting maximum heap size for a specific environment see SAP note 717376 “iSeries: Recommended Settings for SAP WebAS Java”.

- **Xms (initial heap size)**

  This parameter is the initial heap size value, which is the threshold that the garbage collector uses to know when to initiate a garbage collection cycle. The initial heap size does not refer to the size of the heap at any point during the life of the JVM. The initial heap size value is used to tune the JVM. Setting this property depends on hardware, workload, and other variables. For more information about setting initial heap size for a specific environment see SAP note 717376 “iSeries: Recommended Settings for SAP WebAS Java”.

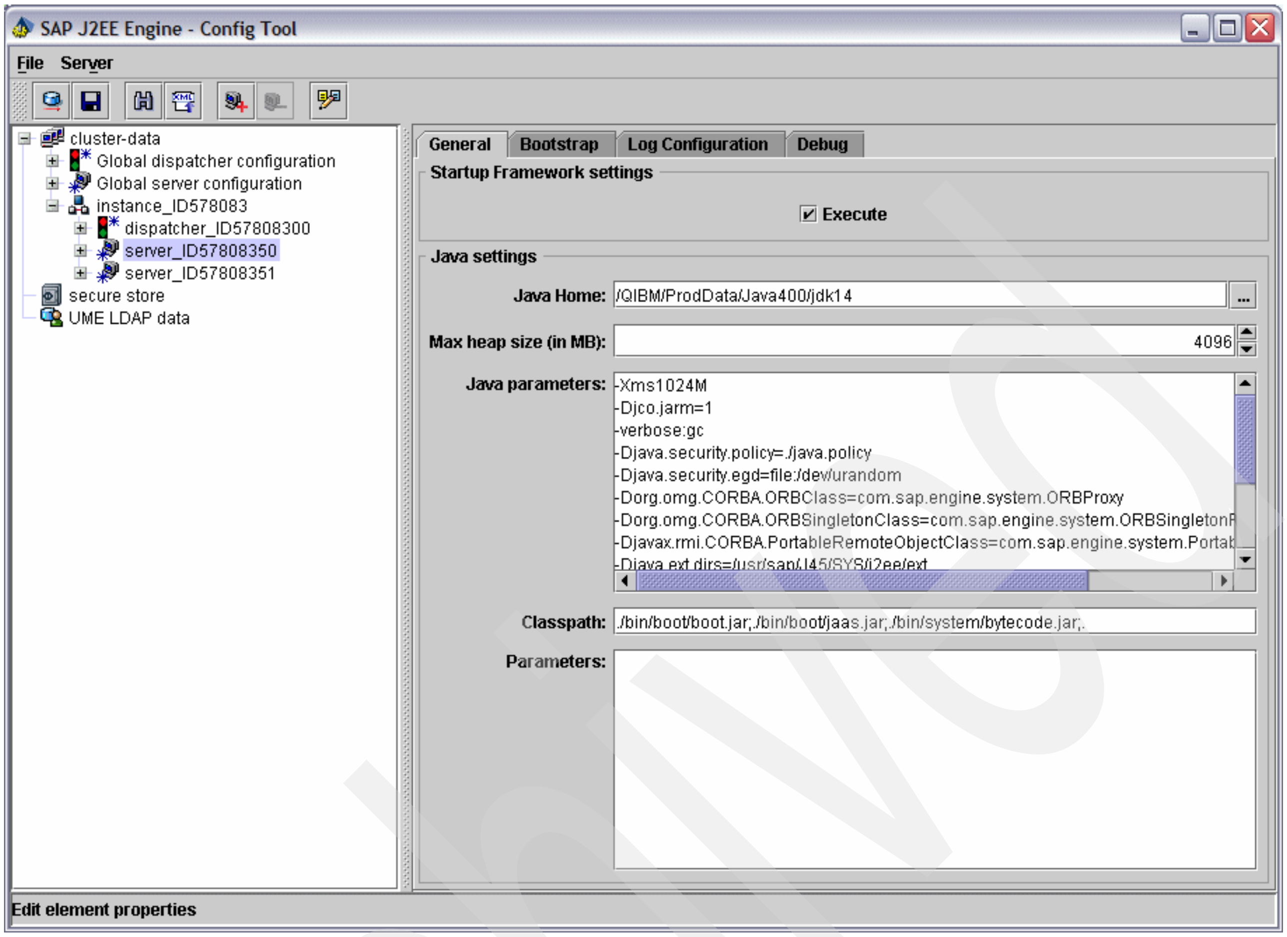
4.5.3 Modifying JVM system properties

Use the following steps to modify the JVM system properties:

1. Open the SAP J2EE Engine - Config Tool.
2. Expand cluster-data.
3. Expand the instance.
4. Click on the component to be changed.
5. Modify or add the property in the pane on the right.
6. To set the bootstrap JVM properties, click the Bootstrap tab.

Figure 4-5 on page 28 shows the Config Tool being used to change the JVM properties for a server node.

Archived


SAP J2EE Engine - Config Tool

File Server

- cluster-data
  - Global dispatcher configuration
  - Global server configuration
  - instance_ID578083
    - dispatcher_ID57808300
    - server_ID57808350
    - server_ID57808351
  - secure store
  - UME LDAP data

General | Bootstrap | Log Configuration | Debug

Startup Framework settings

☑ Execute

Java settings

Java Home: /QIBM/ProdData/Java400/jdk14

Max heap size (in MB): 4096

Java parameters:

```
-Xms1024M
-Djco.jarm=1
-verbose:gc
-Djava.security.policy=./java.policy
-Djava.security.egd=file:/dev/urandom
-Dorg.omg.CORBA.ORBClass=com.sap.engine.system.ORBProxy
-Dorg.omg.CORBA.ORBSingletonClass=com.sap.engine.system.ORBSingletonP
-Djavax.rmi.CORBA.PortableRemoteObjectClass=com.sap.engine.system.Portab
-Djava.ext.dirs=/usr/sap/J45/SYS/j2ee/ext
```

Classpath: ./bin/boot/boot.jar;./bin/boot/jaas.jar;./bin/system/bytecode.jar;.

Parameters:

Edit element properties

Figure 4-5 Changing JVM system properties for a server node (©SAP AG 2006. All rights reserved.)

4.6 Fine tuning the IBM i5/OS Classic JVM

By meeting the minimum requirements and applying the recommended settings, your SAP Java Application Server should have acceptable performance. Due to the variables involved and the differences between landscapes it is impossible to provide one set of recommendations that is ideal for every situation. You might find that it is necessary to tune the JVM in order to get the desired performance. As your business changes and your workload characteristics change significantly you can find that your JVM needs occasional tuning as well.

No one thing impacts the JVM performance more than the garbage collector. By properly adjusting the garbage collector settings your JVM runs more efficiently with an optimal balance between memory consumption and CPU utilization.

4.6.1 Analyzing the JVM garbage collector

There are many options available when it comes to analyzing the behavior of the JVM garbage collector. Some methods such as using Performance Explorer require additional products and tend to provide low level information.

Look at the output of the garbage collector to get a quick analysis of the garbage collector. The JVM prints garbage collector information when the verbose:gc system property is set. By default the SAP Java Application Server has this parameter set for each server JVM. The garbage collector output is printed to standard out, but is redirected by SAP to the work directory and is found in files named std_<component>, where <component> is the server, dispatcher, and so on. See section 3.2, “Integrated File System” on page 11 for the location of the work directory.

You can view the garbage collector output using any text viewer or editor. A pattern matching tool, such as grep, is helpful but not necessary. Grep is available on i5/OS through the PASE terminal, which is started by running CALL QP2TERM from the command line.

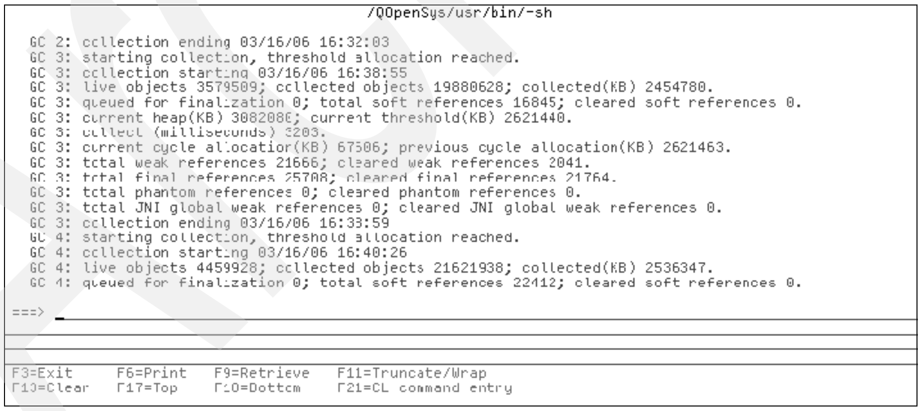
For information about other garbage collector analysis options see the IBM Redbook *Java and WebSphere Performance on IBM @server iSeries Servers*, SG24-6256.

Understanding garbage collector output

Each time the garbage collector runs (this is called a garbage collection cycle) a multi-line entry is made in the log. Entries for each garbage collection cycle include the following data:

- Cycle number
- Collection start time
- Reason for collection
- Number of objects on the heap
- Number of objects collected
- Size of objects collected
- Number and types of references
- Cycle time
- Size of the heap
- Amount heap has grown since garbage collection cycle started
- Cycle ending time

Figure 4-6 shows a portion of the garbage collector output for a server JVM running the SAP Java Application Server.



```
                              /QOpenSys/usr/bin/-sh

  GC 2: collection ending 03/16/06 16:32:03
  GC 3: starting collection, threshold allocation reached.
  GC 3: collection starting 03/16/06 16:38:55
  GC 3: live objects 3579509; collected objects 19880628; collected(KB) 2454780.
  GC 3: queued for finalization 0; total soft references 16845; cleared soft references 0.
  GC 3: current heap(KB) 3082080; current threshold(KB) 2621440.
  GC 3: collect (milliseconds) 3203.
  GC 3: current cycle allocation(KB) 67506; previous cycle allocation(KB) 2621463.
  GC 3: total weak references 21666; cleared weak references 2041.
  GC 3: total final references 25708; cleared final references 21764.
  GC 3: total phantom references 0; cleared phantom references 0.
  GC 3: total JNI global weak references 0; cleared JNI global weak references 0.
  GC 3: collection ending 03/16/06 16:33:59
  GC 4: starting collection, threshold allocation reached.
  GC 4: collection starting 03/16/06 16:40:26
  GC 4: live objects 4459928; collected objects 21621938; collected(KB) 2536347.
  GC 4: queued for finalization 0; total soft references 22412; cleared soft references 0.

 ===> _

 F3=Exit      F6=Print    F9=Retrieve   F11=Truncate/Wrap
 F13=Clear    F17=Top     F20=Bottom    F21=CL command entry
```

Figure 4-6 JVM garbage collector output

Looking at the garbage collector output gives you a good idea of how the garbage collector and the JVM are behaving. Start by looking for obvious problems, that include the following:

- Abnormal heap growth
- Extremely long cycle times
- Garbage collector not keeping up
- Unexpected reasons for garbage collection

Archived

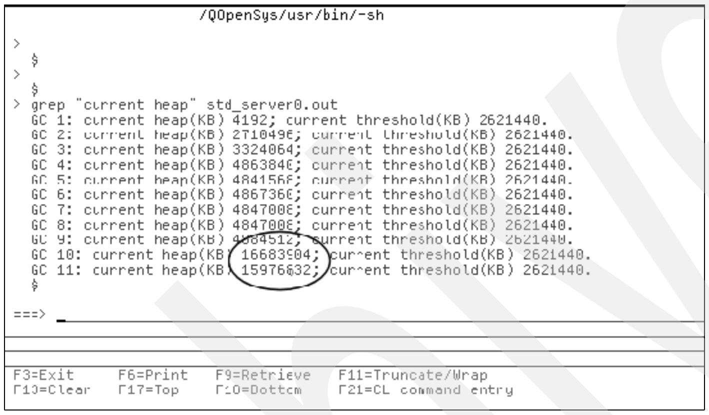
After you check for the obvious tuning problems, and correct if necessary, then analyze the frequency and duration of garbage collection cycles.

Heap growth

After a Java system comes up and is in a steady state, the size of the Java heap should remain within a fairly consistent range. Large deviations that are not the result of a change in workload characteristics are an indication that the garbage collector is improperly tuned. You can identify abnormal heap growth by looking at the *current heap (KB)* field. Compare multiple cycles to identify an anomaly.

Figure 4-7 shows an example of runaway heap growth. The data in the figure is the garbage collector output from server0 and is formatted using the following grep command from the PASE terminal:

```
grep “current heap” std_server0.out
```



```
                    /QOpenSys/usr/bin/-sh

>
  $
>
  $
> grep "current heap" std_server0.out
  GC 1: current heap(KB) 4192; current threshold(KB) 2621440.
  GC 2: current heap(KB) 2710496; current threshold(KB) 2621440.
  GC 3: current heap(KB) 3324064; current threshold(KB) 2621440.
  GC 4: current heap(KB) 4863840; current threshold(KB) 2621440.
  GC 5: current heap(KB) 4841568; current threshold(KB) 2621440.
  GC 6: current heap(KB) 4867360; current threshold(KB) 2621440.
  GC 7: current heap(KB) 4847008; current threshold(KB) 2621440.
  GC 8: current heap(KB) 4847008; current threshold(KB) 2621440.
  GC 9: current heap(KB) 4884512; current threshold(KB) 2621440.
  GC 10: current heap(KB) 16683904; current threshold(KB) 2621440.
  GC 11: current heap(KB) 15976632; current threshold(KB) 2621440.
  $

===> _

F3=Exit     F6=Print    F9=Retrieve   F11=Truncate/Wrap
F13=Clear   F17=Top     F18=Bottom    F21=CL command entry
```

Figure 4-7   Abnormal heap growth

Possible reasons for abnormal heap growth include:

- Not enough memory available to the JVM (an increase in non-database paging can also occur).
- Initial heap size too small, leading to the situation where the garbage collector cannot keep up.

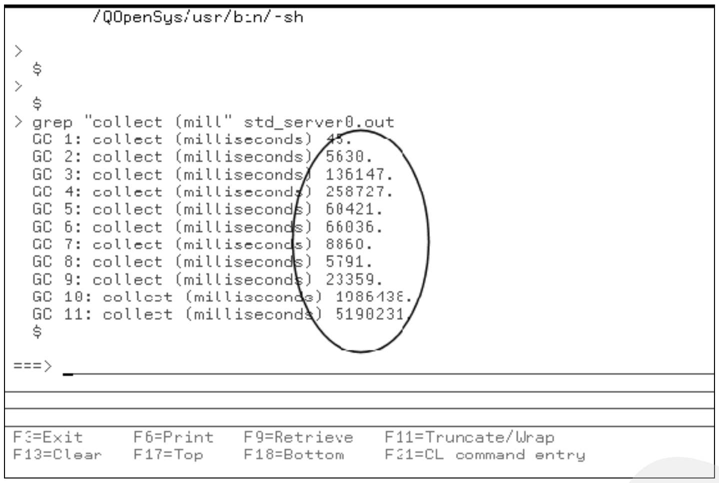
Garbage collection cycle time

Even though the garbage collector itself is multi-threaded, only one garbage collection cycle can run at one time. During the time that the garbage collector is running, the Java heap continues to grow. Extremely long cycle times allow the Java heap to grow more than it should, which can lead to a situation where the garbage collector cannot keep up or the heap grows too large. Large fluctuations in the length of cycle time might also indicate that the garbage collector is not well tuned.

Figure 4-8 on page 31 shows an example of extremely long cycle times and large fluctuations in cycle time. The data in the figure is the garbage collector output from server0. It is formatted using the following grep command from the PASE terminal:

```
grep “collect (mill” std_server0.out
```

Archived



```
       /QOpenSys/usr/bin/-sh

>
  $
>
  $
> grep "collect (mill" std_server0.out
  GC 1: collect (milliseconds) 45.
  GC 2: collect (milliseconds) 5630.
  GC 3: collect (milliseconds) 136147.
  GC 4: collect (milliseconds) 258727.
  GC 5: collect (milliseconds) 60421.
  GC 6: collect (milliseconds) 66036.
  GC 7: collect (milliseconds) 8860.
  GC 8: collect (milliseconds) 5791.
  GC 9: collect (milliseconds) 23359.
  GC 10: collect (milliseconds) 1986438.
  GC 11: collect (milliseconds) 5190231.
  $

===> _

F3=Exit     F6=Print   F9=Retrieve   F11=Truncate/Wrap
F13=Clear   F17=Top    F18=Bottom    F21=CL command entry
```

Figure 4-8 Large fluctuations and extremely long cycle times

Possible reasons for extremely large cycle times or large fluctuations in cycle time include the following:

- Not enough memory available to the JVM, which causes a significant portion of the Java heap to be paged to disk. Extremely long cycle times are usually accompanied by a large increase in heap size.
- Initial heap size too small, leading to the situation where the garbage collector cannot keep up.

Garbage collector efficiency

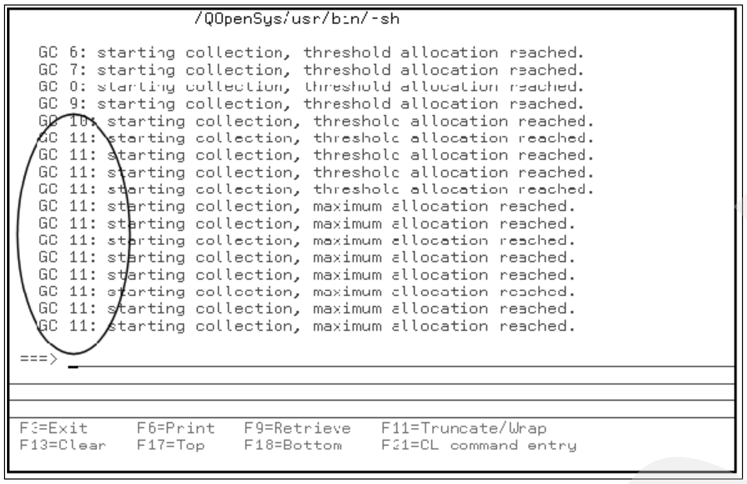
Due to the asynchronous nature of the garbage collector, see section 2.2, “JVM Garbage collection” on page 7 for a discussion on the garbage collector, the heap continues to grow during a garbage collection cycle. This offers the advantage that a user is not interrupted when the garbage collector is running. However, it can lead to a situation where the heap grows faster than the garbage collector can collect.

The garbage collector needs to have enough time to complete before the next cycle begins. Symptoms of a garbage collector that cannot keep up include garbage collection cycles that are stacked up and reasons other than “threshold allocation reached” for the starting collection reason.

Figure 4-9 on page 32 shows an example of stacked garbage collection cycles. The data in the figure is the garbage collector output from server0 and is formatted using the following grep command from the PASE terminal:

```
grep “starting collection” std_server0.out
```

Archived



```
                  /QOpenSys/usr/bin/-sh

  GC 6: starting collection, threshold allocation reached.
  GC 7: starting collection, threshold allocation reached.
  GC 8: starting collection, threshold allocation reached.
  GC 9: starting collection, threshold allocation reached.
  GC 10: starting collection, threshold allocation reached.
  GC 11: starting collection, threshold allocation reached.
  GC 11: starting collection, threshold allocation reached.
  GC 11: starting collection, threshold allocation reached.
  GC 11: starting collection, threshold allocation reached.
  GC 11: starting collection, maximum allocation reached.
  GC 11: starting collection, maximum allocation reached.
  GC 11: starting collection, maximum allocation reached.
  GC 11: starting collection, maximum allocation reached.
  GC 11: starting collection, maximum allocation reached.
  GC 11: starting collection, maximum allocation reached.
  GC 11: starting collection, maximum allocation reached.
  GC 11: starting collection, maximum allocation reached.

===> _

F3=Exit     F6=Print    F9=Retrieve    F11=Truncate/Wrap
F13=Clear   F17=Top     F18=Bottom     F21=CL command entry
```

Figure 4-9 Stacked garbage collection cycles

The main reason that the garbage collector cannot keep up is that the initial heap size is too small. Increase the heap size until this condition no longer occurs.

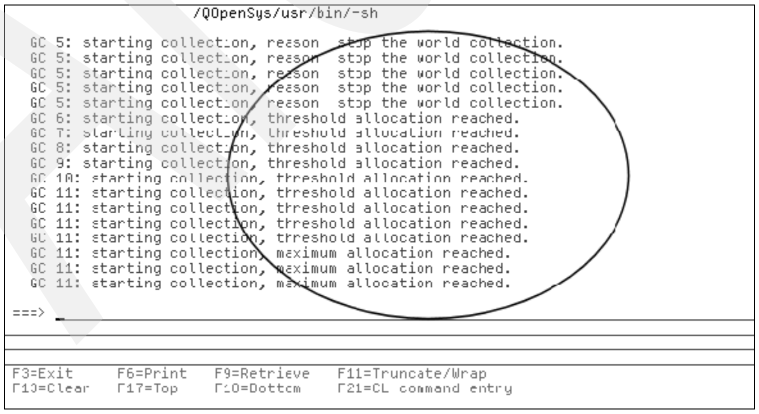
Reasons for garbage collection invocation

Under normal circumstances the garbage collector is only invoked when the threshold, indicated by the initial heap size, is reached. See section 2.2, "JVM Garbage collection" on page 7 for more discussion on the garbage collector design. The only exception to this is garbage cycle 1 (GC 1). The garbage collection reason for GC1 is always "external thread attached". For all other garbage collection cycles the reason should be "threshold allocation reached". Following are other reasons for garbage collection:

- Maximum allocation reached
- Stop the world collection

Figure 4-10 shows an example of what the garbage collector output looks like when the garbage collector is invoked for different reasons. The data in the figure is the garbage collector output from server0 and is formatted using the following grep command from the PASE terminal:

```
grep "starting collection" std_server0.out
```



```
                  /QOpenSys/usr/bin/-sh

  GC 5: starting collection, reason  stop the world collection.
  GC 5: starting collection, reason  stop the world collection.
  GC 5: starting collection, reason  stop the world collection.
  GC 5: starting collection, reason  stop the world collection.
  GC 5: starting collection, reason  stop the world collection.
  GC 6: starting collection, threshold allocation reached.
  GC 7: starting collection, threshold allocation reached.
  GC 8: starting collection, threshold allocation reached.
  GC 9: starting collection, threshold allocation reached.
  GC 10: starting collection, threshold allocation reached.
  GC 11: starting collection, threshold allocation reached.
  GC 11: starting collection, threshold allocation reached.
  GC 11: starting collection, threshold allocation reached.
  GC 11: starting collection, threshold allocation reached.
  GC 11: starting collection, maximum allocation reached.
  GC 11: starting collection, maximum allocation reached.
  GC 11: starting collection, maximum allocation reached.

===> _

F3=Exit     F6=Print    F9=Retrieve    F11=Truncate/Wrap
F13=Clear   F17=Top     F18=Bottom     F21=CL command entry
```

Figure 4-10 Garbage collector output showing different reasons for invocation

A reason of *Maximum allocation reached* indicates that the maximum heap size is set too low. The maximum heap size should not be used to limit the size of the heap and should be

Archived

set large enough that the size of Java heap does not approach this value. If the size of the heap repeatedly reaches the maximum heap size value, then there is an increased chance that excessive “out of memory” errors can occur. The reason for “maximum allocation reached” is almost exclusively due to the maximum heap size being set too low. The maximum heap size system property is discussed in more detail in section 4.5.2, “Tuning properties” on page 27.

A reason of *stop the world* indicates that the garbage collection cycle is taking too long or that the heap is reaching critical mass. A “stop the world” collection forces all Java threads to suspend while a collection is performed. This causes the user to experience long response times and is generally unacceptable. Reasons for “stop the world” collections include the following:

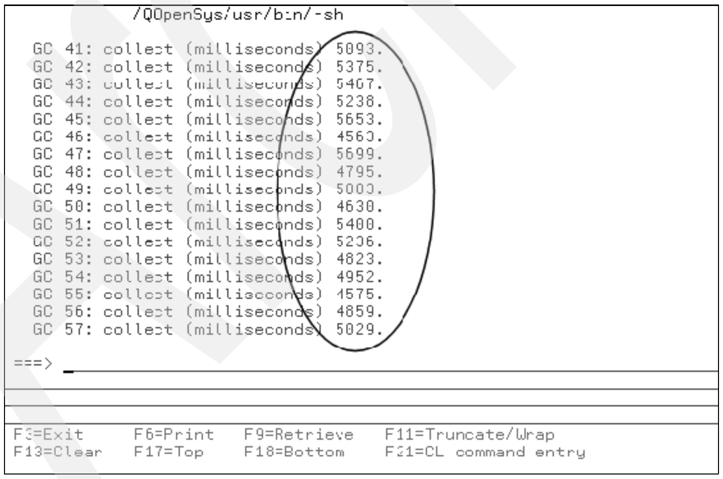
- Garbage collector unable to keep up
- Insufficient memory available to the JVM
- Maximum heap size set too low

Frequency and duration of garbage collection cycles

The garbage collector should be tuned so that there is a balance between the duration of garbage collection cycle and the frequency of those cycles. A garbage collection cycle has significant overhead in the form of CPU consumption. Running the garbage collector too frequently because of the initial heap size settings can consume CPU unnecessarily. A rule of thumb is that the garbage collector should be idle more than half of the time a JVM is running under a load. A quick way to determine this is to add up the lengths of all of the garbage collection cycles that occurred during an arbitrary minute and then subtract that from 60 seconds. Doing this for a few samples gives you a pretty good idea whether your application is spending too much time collecting garbage.

In Figure 4-11 you can see that the garbage collection cycle time ranges from between four and five seconds. The data in the figure is the garbage collector output from server0 and is formatted using the following grep command from the PASE terminal:

```
grep “collect (mill” std_server0.out
```



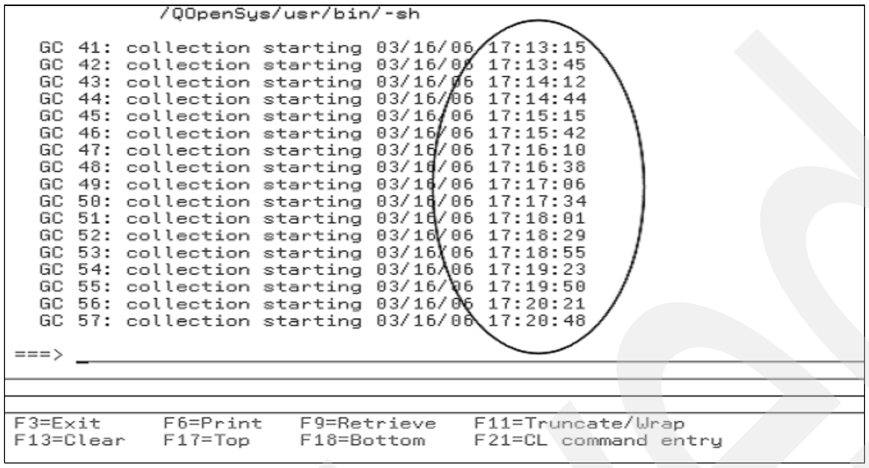
```
            /QOpenSys/usr/bin/-sh

  GC 41: collect (milliseconds) 5093.
  GC 42: collect (milliseconds) 5375.
  GC 43: collect (milliseconds) 5407.
  GC 44: collect (milliseconds) 5238.
  GC 45: collect (milliseconds) 5653.
  GC 46: collect (milliseconds) 4563.
  GC 47: collect (milliseconds) 5699.
  GC 48: collect (milliseconds) 4795.
  GC 49: collect (milliseconds) 5003.
  GC 50: collect (milliseconds) 4630.
  GC 51: collect (milliseconds) 5400.
  GC 52: collect (milliseconds) 5236.
  GC 53: collect (milliseconds) 4823.
  GC 54: collect (milliseconds) 4952.
  GC 55: collect (milliseconds) 4575.
  GC 56: collect (milliseconds) 4859.
  GC 57: collect (milliseconds) 5029.

 ===> _

 F3=Exit      F6=Print    F9=Retrieve    F11=Truncate/Wrap
 F13=Clear    F17=Top     F18=Bottom     F21=CL command entry
```

Figure 4-11 Determining garbage collection cycle time

Archived

In Figure 4-12 you can see that the number of garbage collection cycles per minute is roughly two per minute. The data in the figure is the garbage collector output from server0 and is formatted using the following grep command from the PASE terminal:

```
grep "collect (mill" std_server0.out
```



```
             /QOpenSys/usr/bin/-sh

  GC 41: collection starting 03/16/06 17:13:15
  GC 42: collection starting 03/16/06 17:13:45
  GC 43: collection starting 03/16/06 17:14:12
  GC 44: collection starting 03/16/06 17:14:44
  GC 45: collection starting 03/16/06 17:15:15
  GC 46: collection starting 03/16/06 17:15:42
  GC 47: collection starting 03/16/06 17:16:10
  GC 48: collection starting 03/16/06 17:16:38
  GC 49: collection starting 03/16/06 17:17:06
  GC 50: collection starting 03/16/06 17:17:34
  GC 51: collection starting 03/16/06 17:18:01
  GC 52: collection starting 03/16/06 17:18:29
  GC 53: collection starting 03/16/06 17:18:55
  GC 54: collection starting 03/16/06 17:19:23
  GC 55: collection starting 03/16/06 17:19:50
  GC 56: collection starting 03/16/06 17:20:21
  GC 57: collection starting 03/16/06 17:20:48

===> _

F3=Exit      F6=Print    F9=Retrieve    F11=Truncate/Wrap
F13=Clear    F17=Top     F18=Bottom     F21=CL command entry
```

Figure 4-12 Number of garbage collection cycles per minute

Using these estimates, you can calculate that the garbage collector is running roughly between 8 and 15 seconds out of each minute. Because this is less than 50% of the time, this JVM is adequately tuned for this workload.

Though not as common, it is important that the garbage collector does not run too infrequently. Under a load the garbage collector should run at least once a minute. If the garbage collector does not run frequently enough, there is an increased chance that the heap becomes too large. However, under a light load or no load at all the Java heap grows at a reduced rate, and the garbage collector might not run for many minutes. This is normal behavior.

For more advanced and detailed analysis of the garbage collector, use the IBM Performance Tools for iSeries (5722-PT1) to measure the amount of CPU consumed by the garbage collector relative to the other jobs on the system. If you are interested in this level of analysis refer to the "IBM eServer iSeries IBM Developer Kit for Java", section "Tune Java program performance with the IBM Developer Kit for Java" in the IBM iSeries Information Center.

4.6.2 Tuning the JVM garbage collector

Before you make any attempts at tuning the JVM, it is very important that your system meets the minimum hardware requirements. No amount of JVM tuning can fully compensate for inadequate hardware, especially memory. We also recommend that you set all parameter values directed and complete all other configuration. Tuning the JVM is the last step that takes place.

Tuning the JVM is actually tuning the JVM garbage collector. For a discussion on the JVM garbage collector, see section 2.2, "JVM Garbage collection" on page 7. Keeping in line with the i5/OS core value of simplicity, tune the garbage collector by adjusting only one parameter, the initial heap size (Xms). There might be occasions to adjust the maximum heap size (Xmx); however, if it is set properly to begin with, maximum heap size does not come into play when tuning the garbage collector.

Archived

To tune the garbage collector adjust the initial heap size until you find the right balance between the duration of a garbage collection cycle and the frequency of a garbage collection cycle. The initial heap size determines how often the garbage collector runs. With a small initial heap size the garbage collector runs more frequently on a smaller heap. A larger initial heap size causes the garbage collector to run less frequently but on a larger heap, which causes the garbage collector to run longer during a cycle. Finding a balance between frequency and duration is the key to an optimally tuned GC.

Unfortunately there are no magic numbers when it comes to frequency and duration of the garbage collector, which is due to the endless differences between applications and their behavior. This makes tuning the garbage collector more of an art than a science. Following are some guidelines that can make tuning the GC easier.

- Make sure that your system meets the minimum hardware requirements.
- Set all parameter values as directed, and make sure all other configuration is complete.
- Try to analyze data that is collected when the system is under a high load. A constant workload is best but normally not achievable in a real world environment.
- Start with a reasonable initial heap size, and work your way up. Do not start high and then work your way down.
- Increase initial heap size until the current garbage collector cycle can finish before the next one needs to begin.
- Try to balance the frequency and duration of garbage collection cycles so that the garbage collector runs less than 50% of the time.

The IBM Performance Tools for iSeries (5722-PT1) contains useful tools for analyzing and tuning a JVM. However, in a real world environment this level of tuning is normally not required. If you are interested in other methods of analyzing the JVM, refer to the “IBM eServer iSeries IBM Developer Kit for Java” section “Tune Java program performance with the IBM Developer Kit for Java” in the iSeries Information Center.

Archived

Archived

5

Debugging Java applications

The SAP Java Application Server offers the possibility to remotely debug deployed J2EE applications. The preferred tool for debugging (and developing SAP J2EE applications) is the SAP NetWeaver® Developer Studio, an Eclipse-based integrated development environment (IDE). However, any Java IDE that supports the Java Debug Wire Protocol (JDWP) should be able to connect to the JVM and enable remote debugging.

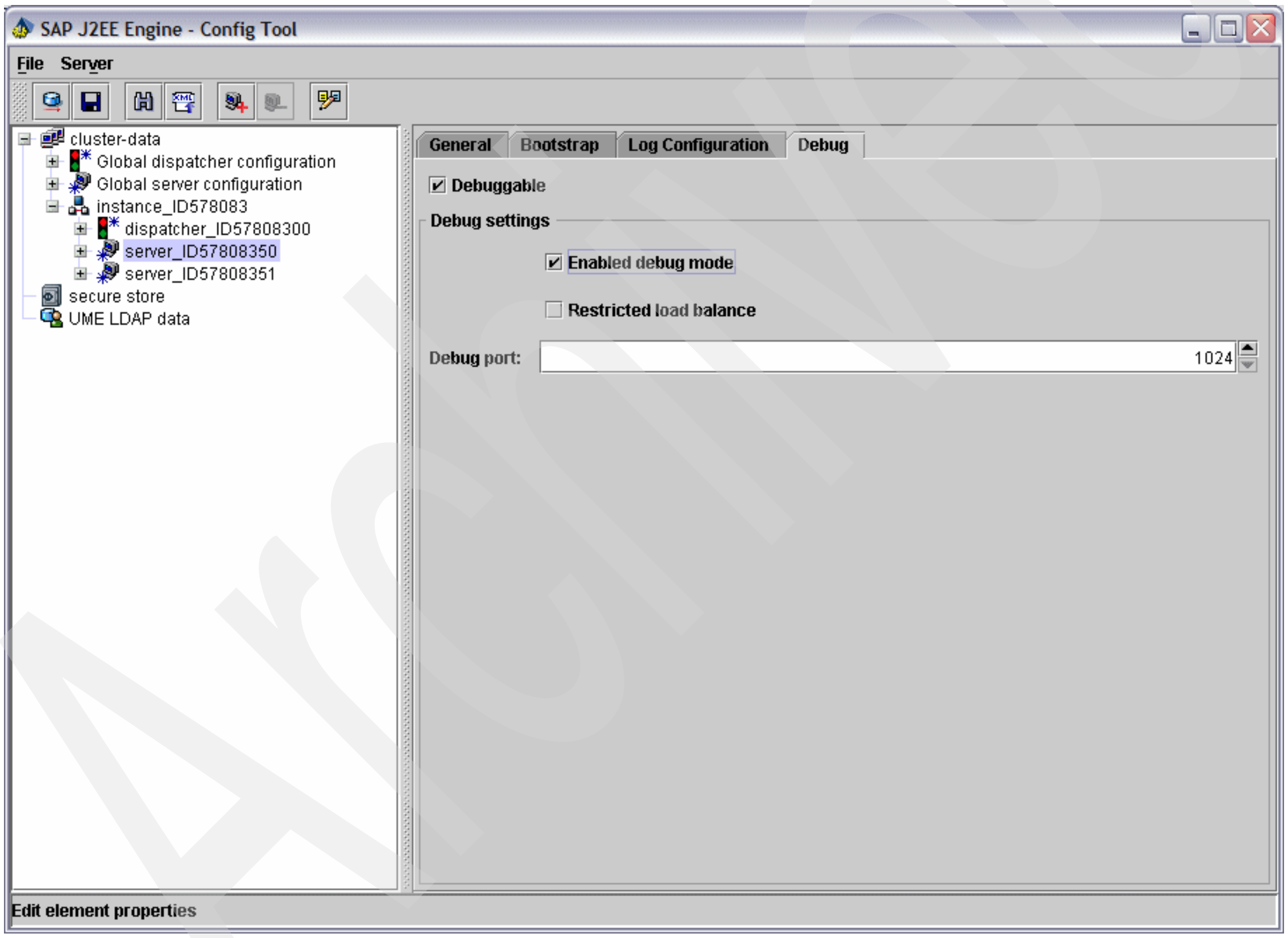
The installation of SAP NetWeaver Developer Studio (Windows® only) is on the SAP NetWeaver 04 SR Java DVD (for SAP NetWeaver 04 based systems) or on the SAP NETWEAVER 2004S SR 1 Developer Workplace™ DVD (for SAP NetWeaver 04S based systems). Make sure that the version of Developer Studio that you use matches the version of the SAP NetWeaver system and matches its support package level. The patches for the SAP Developer Studio are on the SAP software marketplace.

Archived

5.1 Enabling debug operations

Use the following instructions to enable debug operations:

1. To enable debug operations, start the SAP Java Application Server in debug mode, using the following steps:
   a. Open the Config Tool.
   b. Expand the instance.
   c. Expand the server node.
   d. In the right pane, select the debug tab.
2. Select cluster-data → instance_ID<instance ID number> → server_ID<instance ID number>50. The instance ID number is a seemingly random number. A dialog opens, similar to Figure 5-1.



☑ Debuggable

Debug settings

☑ Enabled debug mode

☐ Restricted load balance

Debug port: 1024

Figure 5-1 Debug settings (©SAP AG 2006. All rights reserved.)

3. Click the Debug tab.
4. Select the check box located next to **Debuggable**.
5. Select the check box located next to **Enabled debug mode**.
6. Use the pull-down list to locate and choose a debug TCP/IP port. The default should be acceptable in most cases. Remember the value because it connects to the Eclipse-based SAP Netweaver Developer Studio.

Archived

7. Save and restart the SAP Java Application Server.

The JVM of the server process is started with the following additional options to enable debug:

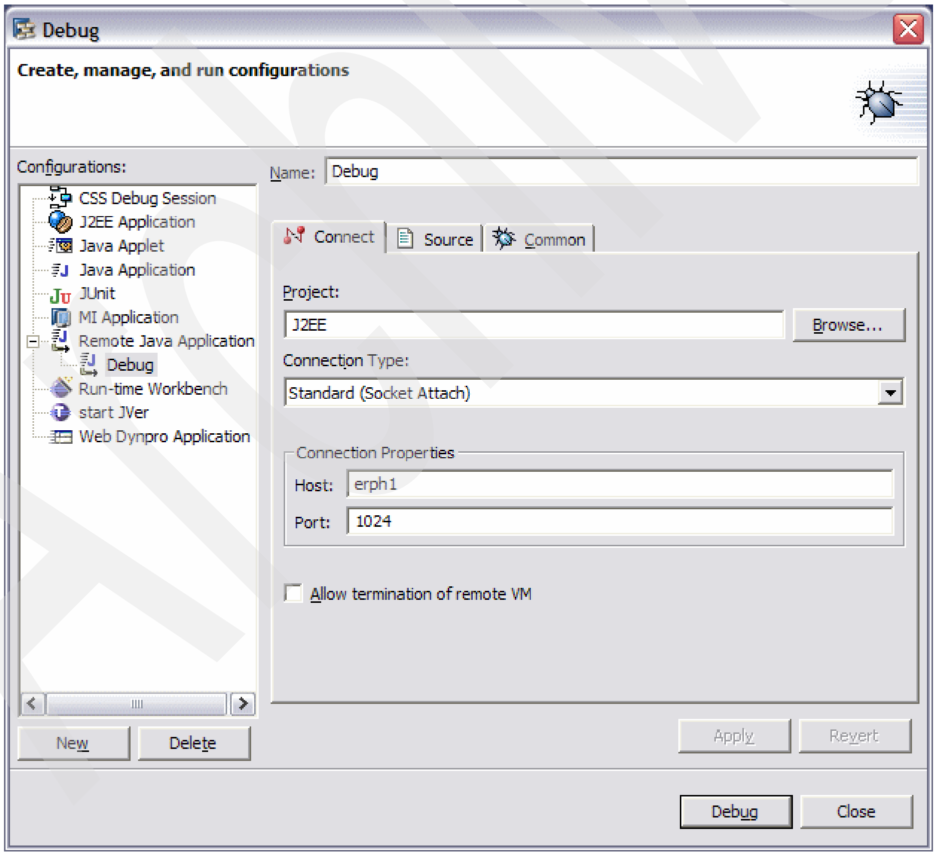
- -Xdebug
- -Xnoagent
- -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=1024

These options are automatically set for the server JVM when the debug settings are set. You do not need to set them manually. However, it is necessary to understand them, especially -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=1024, which is the important option. It causes the JVM start in debug mode and to open a remote debug connection running the Java Debug Wire Protocol (JDWP). Some options are set, including the port for incoming debug connections, in Figure 5-2 it is 1024.

5.1.1 Debug using the SAP NetWeaver Developer Studio

Use the following steps to use the SAP NetWeaver Developer Studio to remotely debug a Java application:

1. In SAP NetWeaver Developer Studio, start the debug session by selecting **Run** → **Debug**.
2. As shown in Figure 5-2, select “Remote Java Application”.
3. Specify your system host and debug port (as configured before).



Name: Debug

Project: J2EE

Connection Type: Standard (Socket Attach)

Connection Properties

Host: erph1

Port: 1024

☐ Allow termination of remote VM

Figure 5-2 Debug configuration (©SAP AG 2006. All rights reserved.)

4. Press Debug to start debugging. The SAP Developer Studio connects to the server VM and displays all threads.

Archived

5. Perform typical debug operations, such as setting breakpoints, evaluating variables, and so on.

Archived

6

Problem analysis

Occasionally problems can occur that prevent the SAP Java Application Server from starting or running properly. You can usually determine the cause of these problems by analyzing log and trace files. This section describes how to use logs and traces to determine the cause of a problem.

Archived

6.1 Locations for logs and traces

There are two important places to find logs and traces for the SAP Java Application Server: the work directory and the log directory. Traces of the SAP system process control layer and kernel are in the work directory. See section 3.2, “Integrated File System” on page 11 for the location of the work directory.

6.1.1 Work directory

Following are important files in the work directory:

- dev_jcontrol - trace file for control process JCONTROL
- dev_<processname> - trace file for control layer of a Java process
- jvm_<processname> - trace file for JVM output of a Java process
- std_<processname> - trace file for stdout output of Java processes
- <processname> can be:
  - bootstrap
  - bootstrap_ID<instance ID number>
  - dispatcher
  - server<n>, where <n> is the number of the server node
  - sdm

The named files contain technical traces and error messages and are readable with a standard text editor or by using SAP transaction AL11, if you are running a Java Add-In System.

6.1.2 Log directory

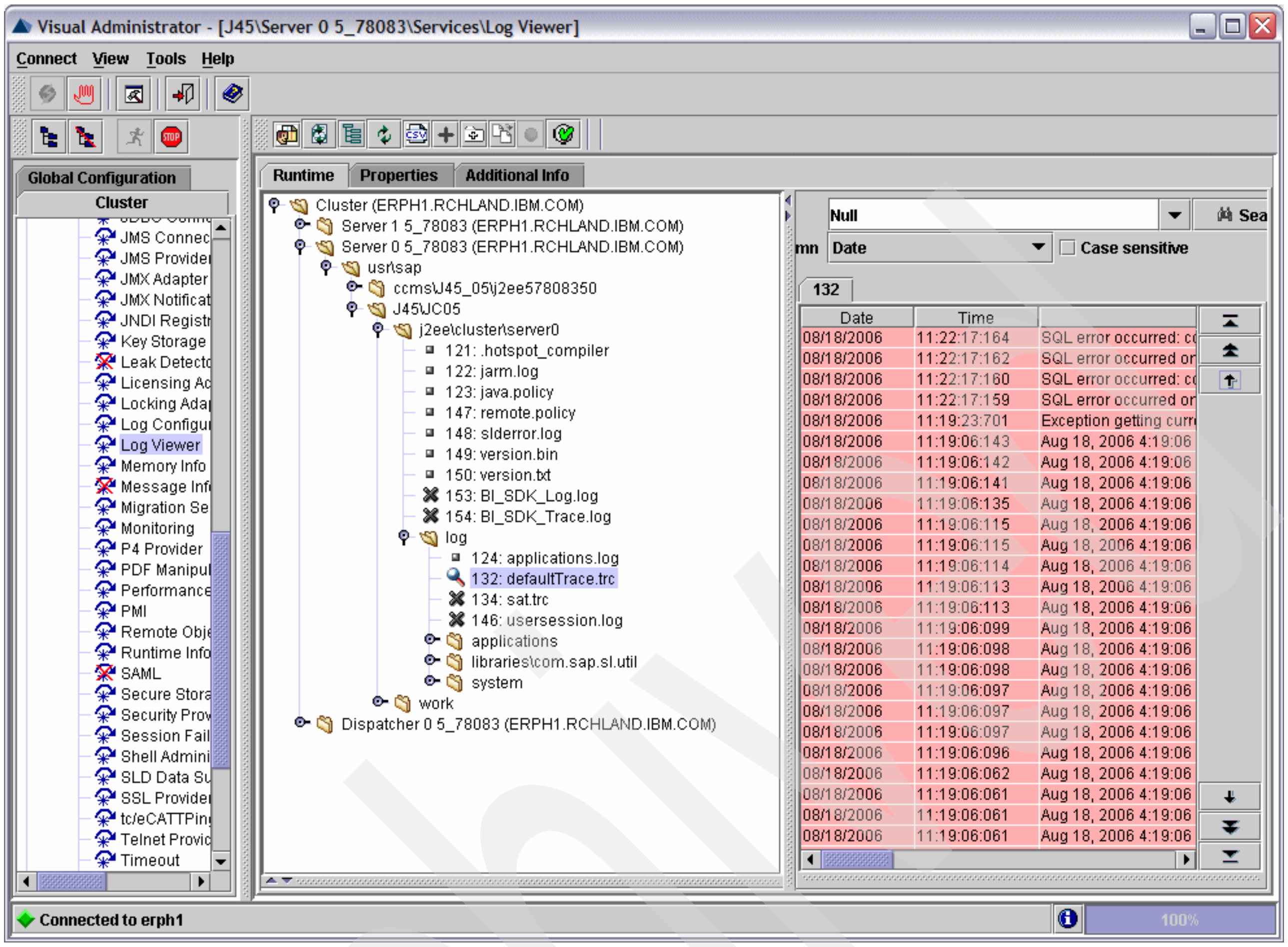
For the Java processes, that is dispatcher and server nodes, you can find Java log files in the directory <instance_directory>/j2ee/cluster/<process>/log, for example, /usr/sap/J45/JC05/j2ee/cluster/dispatcher/log. A helpful log file in this directory is the defaultTrace.<number>.trc, where <number> is a number from 0 to 10 (depending on the log configuration). This trace file is split into smaller files, but it eventually wraps.

These logs contain text entries in a special SAP log viewer format, and are hard to read with a simple text viewer/editor. To get a comfortable view on these log files, use the log viewer of the visual administrator. Another way to look at the Java log entries is using the NetWeaver Administrator in the Web Browser. The following instructions tell you how to start the log viewer:

Use the following steps to start the log viewer:

1. Log on with visual administrator.
2. Open an arbitrarily chosen server node.
3. Click the Log Viewer Service. The log viewer opens, as shown in Figure 6-1 on page 43.

Archived



```
Cluster (ERPH1.RCHLAND.IBM.COM)
  Server 1 5_78083 (ERPH1.RCHLAND.IBM.COM)
  Server 0 5_78083 (ERPH1.RCHLAND.IBM.COM)
    usr\sap
      ccms\J45_05\j2ee57808350
      J45\JC05
        j2ee\cluster\server0
          121: .hotspot_compiler
          122: jarm.log
          123: java.policy
          147: remote.policy
          148: slderror.log
          149: version.bin
          150: version.txt
          153: BI_SDK_Log.log
          154: BI_SDK_Trace.log
          log
            124: applications.log
            132: defaultTrace.trc
            134: sat.trc
            146: usersession.log
            applications
            libraries\com.sap.sl.util
            system
          work
  Dispatcher 0 5_78083 (ERPH1.RCHLAND.IBM.COM)
```

Date	Time	
08/18/2006	11:22:17:164	SQL error occurred: co
08/18/2006	11:22:17:162	SQL error occurred or
08/18/2006	11:22:17:160	SQL error occurred: co
08/18/2006	11:22:17:159	SQL error occurred or
08/18/2006	11:19:23:701	Exception getting curre
08/18/2006	11:19:06:143	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:142	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:141	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:135	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:115	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:115	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:114	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:113	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:113	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:099	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:098	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:098	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:097	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:097	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:097	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:096	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:062	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:061	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:061	Aug 18, 2006 4:19:06
08/18/2006	11:19:06:061	Aug 18, 2006 4:19:06

Figure 6-1 Visual admin logviewer view (©SAP AG 2006. All rights reserved.)

Both the log viewer and the NetWeaver Administrator tools are only available when the SAP Java Application Server is up and running.

To have a look at the logfiles when the Java Application Server is down, you can use the standalone logviewer tool. It can be found in the directory /usr/sap/<SID>/<instance ID>/j2ee/admin/logviewer-standalone. There you find a documentation about its usage, “Logviewer_Userguide.pdf”. It is possible to configure a standalone logviewer server or to just use the tool lv.bat for reading logs in the command line.

A user might look in the work directory when JVMs does not start or when errors occur early in startup. The bootstrap logs are a good place to start. Also, for analyzing garbage collection, the relevant log files are in the work directory. Sometimes both places contain helpful information, for example, a Java log and some stdout output of the JVM.

6.2 Analyzing startup problems

Occasionally the SAP Java Application Server does not start. This can happen, for example, after you apply patches or make configuration changes that cause the Java Application Server to not start. In this section, we provide a common methodology of how to isolate such issues. Performing the following steps to resolve such issues in most cases.

1. Stop the Java instance.

Archived

2. To prevent distractions by old log entries that might detract from the real cause of the problem, we recommend that you delete or move the old log and trace files. All log and trace files are automatically created on startup if they do not exist.
3. Try to start the SAP Java Application server. If no Java job (JCONTROL, JLAUNCH, BOOTSTRAP, DISPATCH, SDM or SERVER<n>) comes up in the instance's subsystems and no dev_jcontrol file is created in the work directory, check for spoolfiles of <SID>ADM, <SID>OFR or <SID><instance number> to get an indication about the cause.

If a dev_jcontrol can be found, this is a good point for the start of the error analysis. In the dev_jcontrol, you should be able to retrace the execution of the bootstrap processes and the subsequent start of the Java work processes (dispatcher, servers and SDM).

If the problem occurs already during bootstrap, that is in dev_jcontrol there is no indication of starting the Java worker processes, analyze the trace files for the bootstrap processes (as described in 6.1, “Locations for logs and traces” on page 42).

If the Java processes start but go down during the initial phase (for example, before reaching the state “running”, which can be checked in dev_<process>), the first step is to analyze their log files in the work directory as described in section 6.1, “Locations for logs and traces” on page 42.

For both kinds of problems (during bootstrap or during startup of Java server processes) the most important thing to check, in the dev_<process> logfile, is if the JVM is created successfully.

If the JVMs are created successfully and no issue occurs during the bootstrap processes, check the Java log files, in particular the file defaultTrace.<n>.trc, for Java error messages that indicate the root cause. These files are difficult to read in a standard text editor, but section 6.1, “Locations for logs and traces” on page 42 mentions some tools to use to convert the files into human-readable text.

6.3 Troubleshooting data access problems

Occasionally you need to collect data to analyze data access problems. Accessing the database from a Java application involves multiple layers of software components. All of these components must perform their respective jobs without error in order for the database to successfully execute the request that the application makes. Problems with data access can occur anywhere in this stack and in some cases have nothing to do with the database itself. See SAP note 934468 “JDBC Error Analysis”.

SAP provides various tools at the application level to help in diagnosing data access problems, including Open SQL Monitors and SQL Trace. For more information about some of these options visit the following Web site:

https://www.sdn.sap.com/irj/sdn/developerareas/java

6.3.1 JDBC traces

A JDBC driver trace can determine what database request the application makes. Knowing what the application is asking the database to do assists with isolating and determining the cause of a problem. Activating a JDBC trace impacts performance and can create large amounts of data. Activate a JDBC trace only when you want to resolve a data access problem.

Archived

Depending on the JDBC driver and the type of trace, you can activate a trace in one of three ways:

- Adding a connection property to the JDBC URL.
- Setting a JVM system property.
- Adding an environment variable.

By default, the output of most JDBC traces is printed to the standard output stream, commonly referred to as System.out. Typically System.out corresponds to the display. However, the host environment, application, or the user can specify another output destination.

The SAP Java Application Server usually redirects this output to a file in the Integrated File System. The location of this file depends on the job.

- For a server node the output of the trace is redirected to the defaultTrace.trc files, which are located at /usr/sap/J45/JC05/j2ee/cluster/server<n>/log, where <n> is the server node number.
- For the bootstrap jobs, the output is redirected to jvm_bootstrap<id>.out, where <id> is the identification number of the bootstrap process. This outfile is found in the work directory. See section 3.2, "Integrated File System" on page 11 for the location of the work directory.

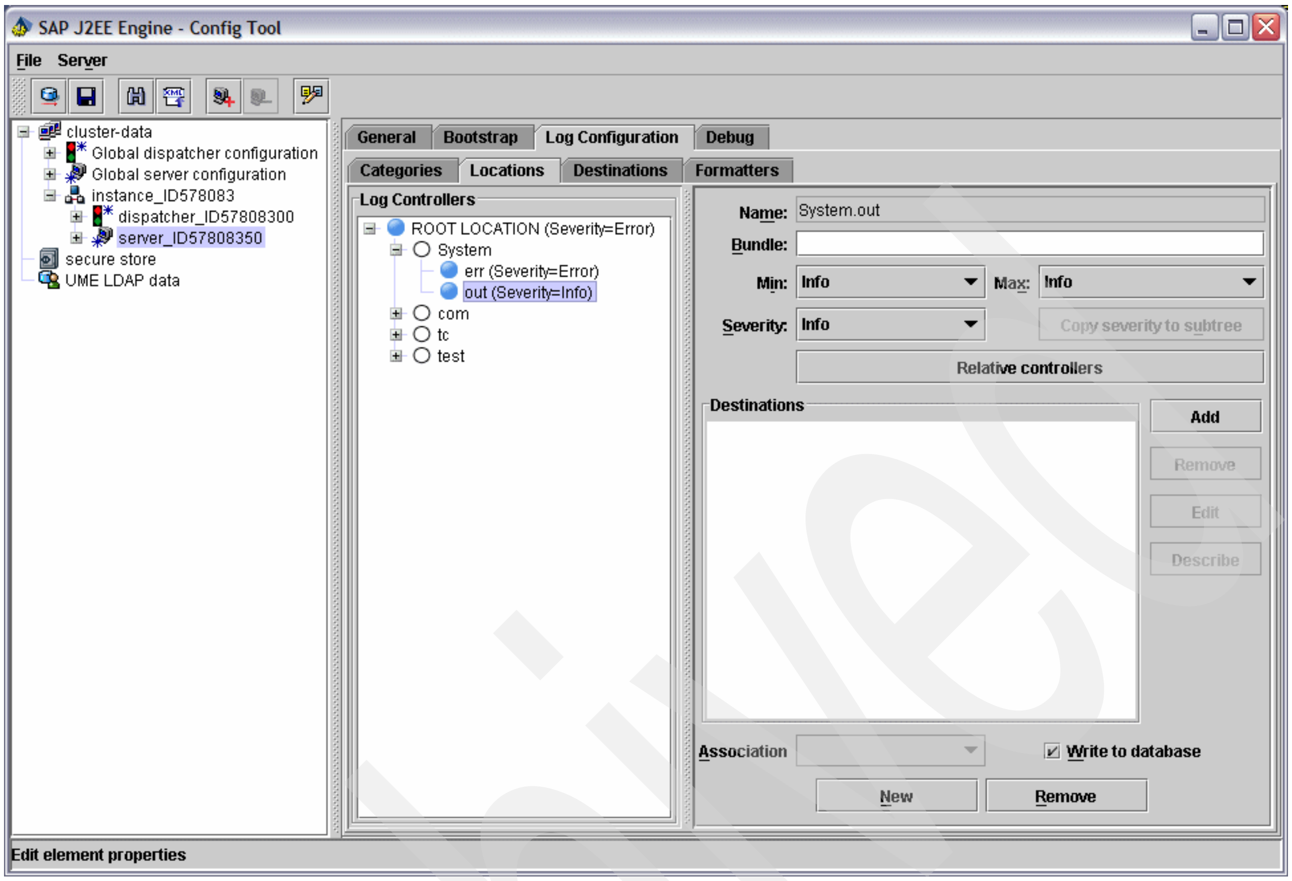
Whenever trace output is printed to System.out, change the SAP Java Application Server logging to allow this data to be logged.

To change the logging option for an instance perform the following steps:

1. Open the SAP J2EE Engine - Config Tool (commonly referred to as the Config Tool).
2. Expand cluster-data.
3. Expand the instance.
4. Click on the component to be changed.
5. In the right pane, select the Log Configuration tab.
6. Select the Locations tab.
7. Expand ROOT LOCATION.
8. Expand System.
9. Select out.
10. In the far right pane, click the pull-down menu for "Severity", and select "Info".
11. Click **Save**.

Figure 6-2 on page 46 shows the Config Tool being used to change the logging options for a server node.

Archived



- cluster-data
  - Global dispatcher configuration
  - Global server configuration
  - instance_ID578083
    - dispatcher_ID57808300
    - server_ID57808350
  - secure store
  - UME LDAP data

- ROOT LOCATION (Severity=Error)
  - System
    - err (Severity=Error)
    - out (Severity=Info)
  - com
  - tc
  - test

Name: System.out
Bundle:
Min: Info
Max: Info
Severity: Info
[✓] Write to database

Figure 6-2 Changing the logging options for System.out (©SAP AG 2006. All rights reserved.)

For most traces, you also have the option of redirecting the output to an Integrated File System file of your choice.

6.3.2 Native JDBC driver traces

The Native Driver supports many different levels of tracing from basic user level traces to more detailed developer level traces. Only user level traces are covered here.

Basic user level trace

A basic user level trace, which includes SQL statement text, can be turned on by adding the connection property "trace=true" to the JDBC URL. The output for this trace by default is printed to System.out. Tracing begins once a new database connection is made. Along with activating the trace, the connection property errors=full can be added to the JDBC URL. This property allows more level of detail to be written to the trace if an error does occur.

Detailed user level traces

Trace containing more detail and different levels of detail can be activated by either setting a JVM system property or by adding an environment variable. Tracing begins once a new database connection is made.

To activate a detailed user level trace using a JVM system property add the following property:

- jdbc.db2.trace=<trace level>

Archived

See section 4.5.3, “Modifying JVM system properties” on page 27 for more information about how to add JVM system properties.

Using the WRKENVVAR command from the i5/OS command line, you can add an environment variable to activate the same trace. Tracing begins once a new database connection is made. To activate the detailed user level trace add the following environment variable:

- QIBM_JDBC_TRACE_LEVEL <trace level>

You can set environment variables at the job or system level using the WRKENVVAR command. We recommend that you set the environment variable at the system level to ensure that tracing is activated.

Table 6-1 shows the trace levels that are available.

Table 6-1 Trace levels for the Native Driver.

Trace level	Description
0	Indicates tracing turned off and no data logged (default)
1	Logs errors that cause an exception
3	Logs JDBC data and flow of control through the code
4	Extended level trace logs all data

Specifying the location of trace data

Specify the location of the trace output by either setting a JVM system property or by adding an environment variable. To specify the location of the trace output with JVM system properties, add the following property:

- jdbc.db2.trace.config=<location>

To specify the location with an environment variable, add the following environment variable:

- QIBM_JDBC_TRACE_CONFIG <location>

Table 6-2 shows the values that you can specify for the location.

Table 6-2 Locations for Native Driver trace output

Value	Description
stdout	Trace data directed to standard out (System.out)
file://<name>	Trace data directed to a specified Integrated File System file, where <name> is a fully qualified file name
usrtrc	Trace data directed to CPA trace output file (default)

We recommend setting the location for the trace to an Integrated File System file for easy retrieval.

6.3.3 Toolbox JDBC driver traces

The Toolbox Driver consists of a Java client that connects to a server job running on i5/OS (the i5/OS Hostserver). Traces for the Toolbox Driver can consist of a client only trace, a server trace, or both. In general, analysis starts with the client, not the server.

Archived

Basic client trace

Turn on a basic client trace, which includes SQL statement text, by adding the connection property “trace=true” to the JDBC URL. By default the output for this trace is sent to System.out. Tracing begins once a new database connection is made. Along with activating the trace, the connection property **`errors=full`** can be added to the JDBC URL. This property allows more level of detail to be written to the trace if an error does occur.

Detailed client traces

Activate client side traces containing more details and different levels of detail by either adding a connection property to the JDBC URL or by setting a JVM system property. By default the output for these traces are sent to System.out. Tracing begins once a new database connection is made.

To activate a client side trace by adding a JDBC connection property, add the following to the JDBC URL:

- toolbox trace=<trace_level>

where <trace_level> is a value specifying the trace detail. See Table 6-3 for information about trace levels. See section 3.4.3, “The JDBC URL” on page 15 for more information about adding connection properties.

To activate the detailed user level trace using a JVM system property add the following property:

- com.ibm.as400.access.Trace.category=<trace_level>”

where <trace_level> is a value specifying the trace detail. See Table 6-3 for information about useful trace levels. See section 4.5.3, “Modifying JVM system properties” on page 27 for more information about setting JVM system properties.

Table 6-3 Trace levels for the Toolbox Driver client side traces

Trace level	Description
none	Tracing turned off and no data logged (default)
datastream	Logs the data flow between the client and the server
diagnostic	Logs object state information
error	Logs errors that cause an exception
information	Logs the flow of control through the code
warning	Logs errors that are recoverable
jdbc	Logs JDBC data
all	Logs all categories

Specifying the location of client side trace data

Specify the location of client side trace output by setting a JVM system property. To specify the location of the trace output with JVM system properties, add the following property:

- com.ibm.as400.access.Trace.file=<location>

where <location> is a fully qualified Integrated File System file name. We recommend setting the location for the trace to an Integrated File System file for easy retrieval.

Archived

Server side traces

Beginning with Toolbox V5R2 or JTOpen 2.02, activate a server trace on the host system. After the server trace is enabled, tracing starts when the client connects to the server and ends when the connection is disconnected. You must start tracing before connecting to the server, because the client enables server tracing only at connect time. A server side trace is used in conjunction with client side traces. The client side trace provides essential information, such as the job ID, for locating the server side trace on the host system. Activate the server trace by either adding a JDBC connection property or by setting a JVM system property. Results for the various traces are located depending on the type of trace that is activated.

To activate a server side trace by adding a JDBC connection property, add the following to the JDBC URL:

- server trace=<trace_level>

where <trace_level> is a value specifying the trace detail. See Table 6-4 for information about trace levels. See section 3.4.3, “The JDBC URL” on page 15 for more information about adding connection properties.

To activate a server side trace using a JVM system property add the following property:

- com.ibm.as400.access.ServerTrace.JDBC=<trace_level>

where <trace_level> is a value specifying the trace detail. See Table 6-4 for information about useful trace levels.

Table 6-4 Trace levels for Toolbox Driver server side traces.

Trace level	Description
0	Trace not activated (default)
1	Basic client side trace This activates the same trace as setting the JDBC connection property trace=true. By default trace data is written to System.out.

2	Starts the database monitor on the server job
	
	The user SAP<sid>DB must have *CHANGE authority to library QUSRSYS in order to activate the database monitor. To grant the necessary authority, run the following command from the i5/OS command line:
	`GRTOBJAUT OBJ(QUSRSYS) OBJTYPE(*LIB) USER(SAP<sid>DB) AUT(*CHANGE)`
	After the Grant Object Authority command completes, the trace authority can be revoked by running the following command:
	`RVKOBJAUT OBJ(QUSRSYS) OBJTYPE(*LIB) USER(SAPJ45DB) AUT(*CHANGE)`
	The database monitor data is then inserted into database files in library QUSRSYS. The names of the files are QJT<job_number>, where <job_number> is the number of the Hostserver job serving the JDBC requests.
	
	To identify a specific Hostserver job that is serving the JDBC requests for the Java Application Server, activate the client side basic trace. Do this by adding the JDBC connection property “trace=true” or by setting the system property "com.ibm.as400.access.Trace.category=jdbc”. Retrieve the job identifier of the server job by searching for the phrase “Server job identifier” in the client trace output.

Trace level	Description

4	Starts debug on the server job
	
	Activating this trace causes more detailed messages to be inserted in the joblog and allows the user to use the debugger on the server job.
	The user SAP<sid>DB must have *CHANGE authority to the command STRDBG in order to activate debug. To grant the necessary authority, run the following command from the i5/OS command line:
	`GRTOBJAUT OBJ(QSYS/STRDBG) OBJTYPE(*CMD) USER(SAP<sid>DB) AUT(*CHANGE)`
	After Grant Object Authority command completes, the trace authority can be revoked by running the following command:
	`RVKOBJAUT OBJ(QSYS/STRDBG) OBJTYPE(*CMD) USER(SAPJ45DB) AUT(*CHANGE)`
	To identify a specify Hostserver job that is serving the JDBC requests for the Java Application Server, activate the client side basic trace. Do this by adding the JDBC connection property “trace=true” or by setting the system property "com.ibm.as400.access.Trace.category=jdbc”. Retrieve the job identifier of the server job by searching for the phrase “Server job identifier” in the client trace output.

8	Saves the job log(s) of the server job when it ends The job logs are printed to the spool files for user QUSER when the Hostserver jobs end.
32	Saves SQL information The SQL information is printed to the spool files for user SAP<sid>DB. The spool files containing SQL information have PRTSQLINF for the user data. Use the following command to view the spool files: `WRKSPLF SELECT(SAPJ45DB *ALL *ALL PRTSQLINF)`

See section 4.5.3, “Modifying JVM system properties” on page 27 for more information about setting JVM system properties.

Multiple types of traces can be started concurrently by adding the trace level values together. For example, “6” starts the database monitor and starts debug.

Archived

Index

Archived

Archived

Archived

Archived

Related publications

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

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks”. Note that some of the documents referenced here may be available in softcopy only.

- *Java and WebSphere Performance on IBM @server iSeries Servers*, SG24-6256
- *Implementing SAP Applications on the IBM System i Platform with IBM i5/OS*, SG24-7166

Online resources

The following Web sites are also relevant as further information sources:

- Time zone values for JVM

  `http://publib.boulder.ibm.com/infocenter/wsdoc400/v6r0/index.jsp?topic=/com.ibm.websphere.iseries.doc/info/ae/ae/adrtzval.htm`
- SAP application level tools

  `https://www.sdn.sap.com/irj/sdn/developerareas/java`

How to get IBM Redbooks

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

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Archived

Archived

Archived

SAP NetWeaver Java on IBM i5/OS

Redpaper

Understand how SAP NetWeaver Java is implemented on IBM i5/OS

Analyze and troubleshoot SAP NetWeaver Java on i5/OS

Optimize the i5/OS JVM for SAP NetWeaver

IBM® i5/OS® and System i™ technology are proven platforms for SAP applications. With more than 10 years of success across more than 2500 SAP worldwide installations in small, medium, and large enterprises, System i models are an ideal platform for SAP customers who are looking for easy usability, high performance, reliability, and carefree operation of their SAP applications. The unique value of System i lies in its ability to reduce information technology (IT) complexity and to simplify SAP system landscapes.

This IBM Redpaper focuses on SAP Java™ technology. It can be used in conjunction with the IBM Redbook *Implementing SAP Applications on the IBM System i Platform with IBM i5/OS*, SG24-7166.

This publication explores areas that are specific to the implementation and integration of the SAP Web Application Server for Java on i5/OS. Included in this document is an overview of the SAP Application Server architecture and how it is implemented on i5/OS. It also includes a discussion on Java and the i5/OS Classic JVM™, followed by configuration and tuning recommendations and finally how to analyze problems.

We wrote this Redpaper to assist SAP basis consultants and other information technology (I/T) professionals in implementing a successful SAP system installation based on SAP Java technology.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

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

For more information:
ibm.com/redbooks

Archived