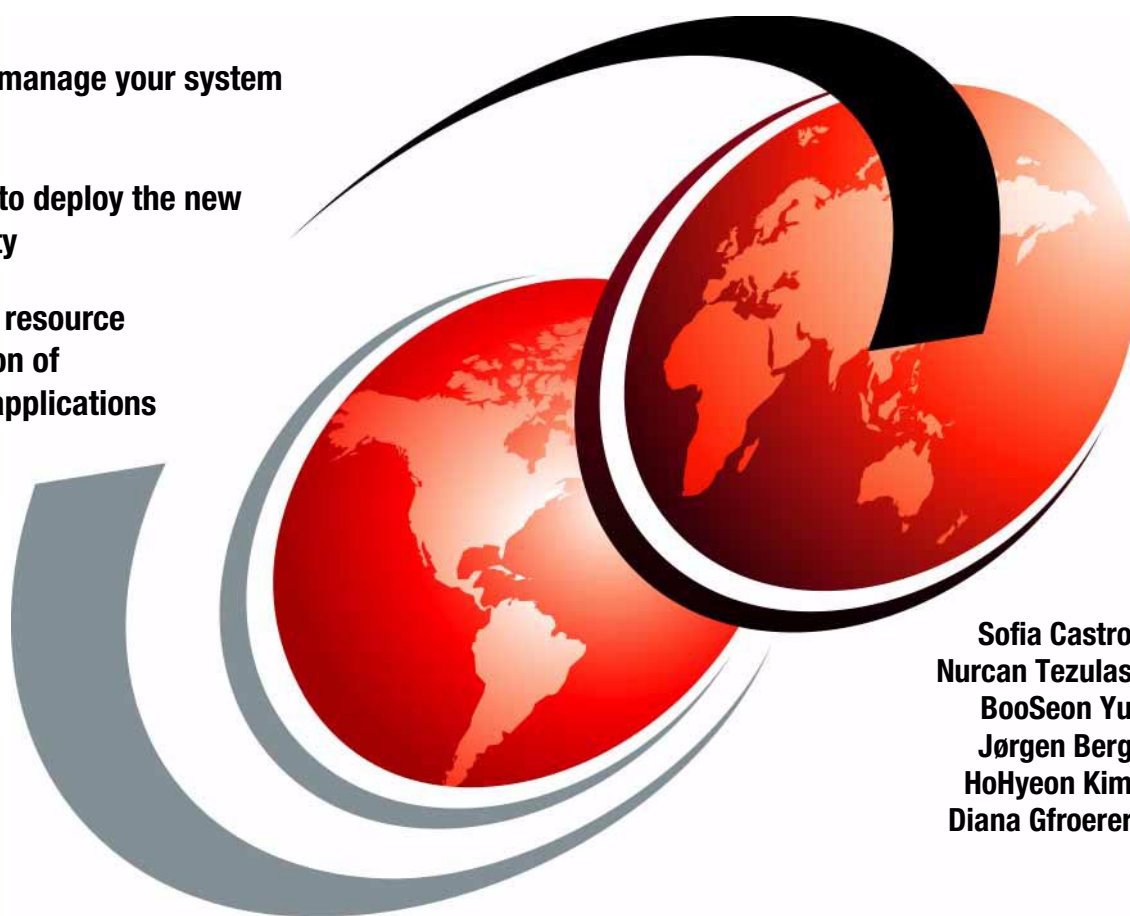


# AIX 5L Workload Manager (WLM)

Effectively manage your system resources

Learn how to deploy the new functionality

Control the resource consumption of individual applications



Sofia Castro  
Nurcan Tezulas  
BooSeon Yu  
Jørgen Berg  
HoHyeon Kim  
Diana Gfroerer

[ibm.com/redbooks](http://ibm.com/redbooks)

**Redbooks**





International Technical Support Organization

**AIX 5L Workload Manager (WLM)**

June 2001

**Take Note!**

Before using this information and the product it supports, be sure to read the general information in Appendix G, "Special notices" on page 315.

**Second Edition (June 2001)**

This edition applies to AIX Workload Manager for use with the AIX 5L for Power Version 5.1 Operating System.

Comments may be addressed to:  
IBM Corporation, International Technical Support Organization  
Dept. JN9B Building 003 Internal Zip 2834  
11400 Burnet Road  
Austin, Texas 78758-3493

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

© Copyright International Business Machines Corporation 2000, 2001. All rights reserved.  
Note to U.S Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Preface</b> .....	ix
The team that wrote this redbook .....	ix
Comments welcome .....	xi
<b>Chapter 1. The need for workload management</b> .....	1
1.1 Architectural differences .....	2
1.1.1 Mainframe partitioning .....	2
1.1.2 UNIX partitioning .....	2
1.1.3 Workload management .....	5
1.2 The purpose of AIX WLM .....	6
<b>Chapter 2. AIX Workload Manager functionality</b> .....	9
2.1 Overview .....	10
2.2 Classes .....	12
2.2.1 Hierarchy of classes .....	12
2.2.2 Superclasses .....	13
2.2.3 Subclasses .....	14
2.2.4 Backward compatibility considerations .....	15
2.3 Tiers .....	16
2.4 Class attributes .....	17
2.5 Classification process .....	22
2.5.1 Automatic assignment .....	22
2.5.2 Manual assignment .....	23
2.5.3 Class assignment rules .....	23
2.5.3.1 Process classification .....	25
2.6 WLM class accounting .....	27
2.6.1 Process accounting using WLM class .....	27
2.6.1.1 Displaying WLM class accounting information .....	28
2.7 Resources .....	29
2.7.1 Resources managed by WLM .....	30
2.7.2 Class resource shares .....	31
2.7.3 Class resource limits .....	33
2.7.4 Backward compatibility considerations .....	35
2.8 WLM interaction with the kernel .....	36
2.8.1 Resource usage statistics .....	36
2.8.1.1 CPU .....	36
2.8.1.2 Memory .....	37
2.8.1.3 Disk I/O .....	37
2.8.2 Uniform Resource Access Priority (URAP) .....	37
2.8.2.1 Tier regulation .....	38
2.8.3 Interaction with the scheduler .....	38

2.8.4	Interaction with VMM . . . . .	39
2.8.5	Interaction with disk device drivers . . . . .	39
2.9	WLM Application Programming Interface . . . . .	40
2.10	Additional characteristics . . . . .	40
<b>Chapter 3. AIX Workload Manager administration . . . . .</b>		<b>43</b>
3.1	Property files . . . . .	44
3.2	WLM configuration . . . . .	53
3.2.1	Steps for a WLM configuration . . . . .	53
3.2.2	Working with WLM configurations . . . . .	56
3.2.3	Working with classes . . . . .	62
3.2.3.1	Using the command line . . . . .	62
3.2.3.2	Using SMIT . . . . .	66
3.2.3.3	Using WSM . . . . .	73
3.2.4	AIX Version 4.3.3 maintenance level 8 wlmset command . . . . .	81
3.2.5	Working with rules . . . . .	82
3.2.5.1	Editing the rules files on the command line . . . . .	82
3.2.5.2	Using SMIT . . . . .	83
3.2.5.3	Using WSM . . . . .	87
3.2.6	Checking the configuration - wlmcheck . . . . .	90
3.2.7	Working with resource sets . . . . .	92
3.2.7.1	Rset registry . . . . .	92
3.2.7.2	Using the command line . . . . .	95
3.2.7.3	Using SMIT . . . . .	97
3.2.7.4	Using WSM . . . . .	97
3.3	WLM operation . . . . .	98
3.3.1	Modes of operation . . . . .	98
3.3.2	Start/Stop/Update WLM - wlmcntrl . . . . .	99
3.3.2.1	Using the command line . . . . .	100
3.3.2.2	Using SMIT . . . . .	102
3.3.2.3	Using WSM . . . . .	103
3.4	Hints and tips . . . . .	107
3.4.1	Things to do . . . . .	107
3.4.2	Things to be aware of . . . . .	111
3.4.3	LoadLeveler and WLM . . . . .	113
3.4.3.1	How does LoadLeveler work . . . . .	113
3.4.3.2	LoadLeveler functionality . . . . .	115
3.4.3.3	LoadLeveler and WLM interaction . . . . .	115
<b>Chapter 4. WLM performance tools . . . . .</b>		<b>117</b>
4.1	wlmstat . . . . .	117
4.2	ps . . . . .	121
4.3	topas . . . . .	124
4.4	svmon . . . . .	135

4.4.1	Workload manager class report . . . . .	137
4.4.2	Workload manager tier report . . . . .	145
4.5	Web-based System Manager (WSM) . . . . .	150
4.6	Monitoring Workload Manager with PTX . . . . .	152
4.6.1	xmperf . . . . .	153
4.6.2	xmservd . . . . .	159
4.6.3	Jazizo . . . . .	160
4.6.4	wlmon / wlmpf. . . . .	160
<b>Chapter 5. Manual assignment . . . . .</b>		<b>177</b>
5.1	Description . . . . .	177
5.1.1	First assignment . . . . .	178
5.1.2	Reassignment and cancellation . . . . .	179
5.1.3	Interaction with inheritance . . . . .	179
5.2	Manual assignment methods . . . . .	182
5.2.1	AIX Version 4.3.3 maintenance level 8 manual assignment . . .	188
5.3	Oracle database example . . . . .	188
5.4	DB2 UDB . . . . .	190
5.4.1	DB2 process model . . . . .	191
5.4.2	Using AIX WLM with DB2 UDB . . . . .	192
5.5	Conclusion . . . . .	192
<b>Chapter 6. WLM Application Programming Interface (API) . . . . .</b>		<b>195</b>
6.1	Application tag . . . . .	195
6.1.1	Description . . . . .	196
6.1.2	An application tag situation . . . . .	196
6.1.3	Example of an application tag program . . . . .	197
6.2	Class management . . . . .	199
6.3	WLM management . . . . .	200
6.4	WLM statistics . . . . .	201
6.5	WLM classification . . . . .	201
6.6	WLM accounting . . . . .	201
6.7	Binary compatibility . . . . .	201
6.8	Integration with Tivoli products . . . . .	202
6.8.1	TAPM overview . . . . .	202
6.8.1.1	Application instrumentation . . . . .	202
6.8.1.2	Transaction simulation . . . . .	202
6.8.2	TAPM and WLM . . . . .	203
6.8.3	Monitoring an application in a WLM and Tivoli environment . . .	203
6.9	Summary . . . . .	204
<b>Chapter 7. Sizing recommendations for Workload Manager . . . . .</b>		<b>205</b>
7.1	Typical UNIX system capacity sizing . . . . .	205
7.2	Server consolidation considerations . . . . .	206

7.3	System capacity sizing for Workload Management . . . . .	208
7.3.1	System capacity sizing steps for server consolidation. . . . .	209
7.3.1.1	Step 1 - Monitor resource usage . . . . .	209
7.3.1.2	Step 2 - Estimate the requirements for each application . . . . .	210
7.3.1.3	Estimate the capacity for integrated applications. . . . .	214
7.3.2	Examples . . . . .	214
7.3.2.1	Base line - Applications running on separate systems . . . . .	214
7.3.2.2	Approach 1 - All applications are mission-critical. . . . .	216
7.3.2.3	Approach 2 - Only some applications are mission-critical . . . . .	217
7.3.2.4	Comparison of the cases . . . . .	220
7.3.3	Considerations for memory and disk I/O bandwidth . . . . .	221
7.4	Conclusion . . . . .	222
	<b>Chapter 8. Practical experience . . . . .</b>	<b>223</b>
8.1	ISV case studies . . . . .	223
8.1.1	PeopleSoft . . . . .	223
8.1.1.1	Case study description. . . . .	224
8.1.1.2	Case study method . . . . .	225
8.1.1.3	WLM configuration . . . . .	228
8.1.1.4	One batch - Two OLTP benchmarks: PAYROLL-FI-HR . . . . .	232
8.1.1.5	One batch - Two OLTP benchmarks: GL-FI-HR . . . . .	234
8.1.1.6	Two batch benchmarks: GL-PAYROLL . . . . .	235
8.1.1.7	Two batch - Two OLTP benchmarks: PAYROLL-GL-FI-HR . . . . .	235
8.1.1.8	Summary . . . . .	237
8.1.2	SAP R/3 Case Study . . . . .	238
8.1.2.1	SAP standard benchmark tool . . . . .	239
8.1.2.2	WLM classes versus OS processes . . . . .	239
8.1.2.3	Multiple SAP R/3 systems consolidation objectives. . . . .	242
8.1.2.4	Multiple systems of equal size and equal priority. . . . .	243
8.1.2.5	Multiple systems of unequal priority. . . . .	245
8.1.2.6	Systems of unequal size but equal priority . . . . .	247
8.1.2.7	One priority system with several additional systems . . . . .	248
8.1.2.8	Process distribution recommendation . . . . .	249
8.2	Customer experience - WLM and a compute server for research . . . . .	252
8.2.1	The installation . . . . .	252
8.2.2	Central AIX system. . . . .	253
8.2.3	Problems . . . . .	254
8.2.4	A pre-WLM solution . . . . .	254
8.2.5	The WLM solution with AIX Version 4.3.3-02 . . . . .	255
8.2.5.1	Major advantages of this solution . . . . .	257
8.2.5.2	Disadvantage of this solution. . . . .	257
8.2.6	The second WLM solution with AIX 5L . . . . .	257
8.2.7	Conclusion . . . . .	258



<b>Appendix A. AIX Workload Manager API routines</b> . . . . .	261
A.1 The Include file - sys/wlm.h. . . . .	261
A.1.1 wlm_args . . . . .	261
A.1.2 wlm_assign. . . . .	264
A.1.3 wlm_info . . . . .	264
A.1.4 wlm_bio_class_info_t . . . . .	266
A.1.5 wlm_bio_div_info_t. . . . .	266
A.2 WLM API functions error codes . . . . .	268
A.3 Initialization routines . . . . .	271
A.3.1 wlm_init_class_definition . . . . .	271
A.3.2 wlm_initialize. . . . .	272
A.4 Application tag . . . . .	273
A.4.1 wlm_set_tag . . . . .	273
A.5 Class management. . . . .	274
A.5.1 wlm_read_classes . . . . .	274
A.5.2 wlm_create_class . . . . .	276
A.5.3 wlm_change_class . . . . .	277
A.5.4 wlm_delete_class . . . . .	279
A.6 WLM management . . . . .	280
A.6.1 wlm_set. . . . .	280
A.6.2 wlm_load. . . . .	281
A.6.3 wlm_assign. . . . .	283
A.7 WLM statistics. . . . .	285
A.7.1 wlm_get_info. . . . .	285
A.7.2 wlm_get_bio_stats . . . . .	288
A.8 WLM classification . . . . .	290
A.8.1 wlm_check . . . . .	290
A.8.2 wlm_classify . . . . .	291
A.9 WLM accounting . . . . .	293
A.9.1 wlm_initkey . . . . .	293
A.9.2 wlm_class2key . . . . .	293
A.9.3 wlm_key2class . . . . .	294
A.9.4 wlm_endkey . . . . .	296
<b>Appendix B. Sample workload program</b> . . . . .	297
<b>Appendix C. Sample Korn shell scripts for manual assignment</b> . . . .	307
C.1 Oracle example script. . . . .	307
<b>Appendix D. Sample program for application tag</b> . . . . .	309
D.1 settag.c. . . . .	309

<b>Appendix E. Sample for CPU resource usage calculation</b> . . . . .	311
<b>Appendix F. Using the additional material</b> . . . . .	313
F.1 Using the diskette . . . . .	313
F.1.1 System requirements for using the diskette . . . . .	313
F.1.2 How to use the diskette. . . . .	313
F.2 Locating the additional material on the Internet . . . . .	313
<b>Appendix G. Special notices</b> . . . . .	315
<b>Appendix H. Related publications</b> . . . . .	319
H.1 IBM Redbooks . . . . .	319
H.2 IBM Redbooks collections . . . . .	319
H.3 Other resources . . . . .	319
H.4 Referenced Web site . . . . .	320
<b>How to get IBM Redbooks</b> . . . . .	321
IBM Redbooks fax order form . . . . .	322
<b>Abbreviations and acronyms</b> . . . . .	323
<b>Index</b> . . . . .	325
<b>IBM Redbooks review</b> . . . . .	335

## **Preface**

This redbook will help you work with AIX Workload Manager (WLM) and exploit the whole spectrum of functionality provided by WLM. It covers the WLM features, including the WLM performance tools, introduced in the Fall of 2000, and is intended to be a workbook and reference to help system administrators and technical support and service professionals gain a deeper understanding of AIX WLM.

This redbook contains a detailed description of how to configure WLM; explains the use of new features such as manual assignment, the WLM API, and the WLM performance tools; and provides hints and tips gained from practical experience. Guidance on system sizing with WLM, primarily in Server Consolidation environments, has been included.

The appendices describe the test programs that were used during the creation of this redbook, and contain sample scripts for manual assignment that can help you use the new features in your environment. They also contain an exhaustive explanation of the WLM API routines as well as a sample program for application tagging to be used with the WLM API.

The shell scripts and sample program are included on a floppy disk at the back of this book.

---

### **The team that wrote this redbook**

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

**Gfroerer Diana** is an International Technical Support Specialist for RS/6000 and AIX Performance at the International Technical Support Organization, Austin Center. She writes extensively and teaches IBM classes worldwide on all areas of AIX, with a focus on performance and tuning. Before joining the ITSO in 1999, Diana Gfroerer worked in AIX pre-sales Technical Support in Munich, Germany, and led the Region Central, EMEA, and World Wide Technical Skill Communities for AIX and PC Interoperability.

**Sofia Castro** is an IT Specialist who has worked for IBM Global Services in Portugal since December 1995. She has four and half years of experience in AIX and communication applications in the area of post-sales support and services. She holds a degree in Computer Science from the Universidade Nova de Lisboa, Portugal, and the University of Leeds, England.

**Nurcan Tezulas** is an IT Specialist who has worked for IBM Germany since March 1996. She began working at IBM Global Services and moved to the Web Server Sales - Enterprise Systems Groups Central Region division in August 1998. Her areas of expertise include HACMP, SAP R/3, and RS/6000 high-end and midrange servers. Currently Nurcan Tezulas leads the High End Technology Focus Group. She holds a degree in Mathematics from the Fachhochschule, Stuttgart, Germany.

**BooSeon Yu** is an IT Specialist who has worked for IBM Korea since April 1996. He spent six months on the S/390 marketing team and has worked for the pre-sales technical support team for RS/6000 since then. His mission includes various benchmark tests, performance tuning, troubleshooting, and solution implementation. BooSeon Yu holds a degree in Materials Engineering.

**Jørgen Berg** is a Senior IT Specialist working in a technical pre-sale support function for ESG Nordic technical support based in Denmark. His areas of expertise include AIX, HACMP, and SP-systems. His mission includes customer presentations, benchmarks, troubleshooting, solution implementation, and giving selected HACMP and SP-system workshops on behalf of IBM learning services.

**HoHyeon Kim** is a System Service Representative who has worked for IBM Global Services in Korea since December 1995. He began at S/390 System during his first year, and has four years of experience in RS/6000 in the area of post-sales and services. He holds a degree in Avionics.

Special thanks to the following people for their invaluable contributions to this project:

**IBM Atlanta**

Tommy Todd

**IBM Austin**

George Accapadi, André Albot, Jack Alford, Jim Beesley, Lee Cheng, Mark Greenberg, Mike Harrell, Ernest A. Keenan, Stephen Nasypany, Anthony Ramirez, Ken Rozendal

**IBM Belgium**

Anke Hollanders

**IBM Dallas**

Tim Leo

**IBM Germany**  
Angel González

**IBM Germany, ISICC Walldorf**  
Carol Davis, Bardin Nelson

**IBM Netherlands**  
Michael A.M. Felt

**Tivoli Systems**  
Fergus Stewart

**Zentralinstitut für Angewandte Mathematik, Forschungszentrum Jülich,  
Germany**  
Klaus Wolkersdorfer

---

## Comments welcome

### Your comments are important to us!

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

- Fax the evaluation form found in “IBM Redbooks review” on page 335 to the fax number shown on the form.
- Use the online evaluation form found at [ibm.com/redbooks](http://ibm.com/redbooks)
- Send your comments in an Internet note to [redbook@us.ibm.com](mailto:redbook@us.ibm.com)



---

## Chapter 1. The need for workload management

This chapter describes the differences between physical partitioning, logical partitioning, and workload management based on the AIX Workload Manager.

Workload management is vital because the conflicting pressures of costs, a lack of skilled support people, fast-growing server farms, and the need for competitive advantage are forcing customers to look for proactive solution designs. Solutions that are not scalable or flexible enough to handle or that cannot avoid increased architectural complexity lead directly to administrator overhead and solution downtime. The consequences are much larger and longer-term problems:

- Increased Total Cost of Ownership (TCO), such as increased hardware, software, and maintenance costs, and costs of excess administrators
- Increasing fragmentation of data and applications across the enterprise
- Reduced ability to exercise financial oversight
- Increased business costs due to outages

Server consolidation is one solution. It helps customers deliver higher IT service levels in a more cost-effective fashion by optimizing both the quantity and distribution of servers supporting mission-critical IT functions.

However, server consolidation does not only mean physical consolidation of many small servers into fewer, more powerful servers. Administrators must go beyond simply moving department applications onto a single system. They must:

- Understand how applications behave under loads and be able to realize what expected loads will be
- Guarantee service levels, such as faster response times, continuous availability, and increased access to data
- Gather detailed information on usage and capacity
- Maximize their ability to make system changes flexibly
- React to changes in workload

Workloads from many different server systems are combined into a single, large system. The most frequent different server systems to be combined are OLTP, batch, print, and general user processing systems. These workloads often interfere with each other and have different goals and service agreements.

The ability to change resource allocation very rapidly with minimal operator intervention but maximum precision utilizing scripts, traditional system management tools, and other components of their IT infrastructure becomes very necessary.

---

## **1.1 Architectural differences**

The demand for advanced management functionality has caused some confusion about the differences between partitioning and workload management.

These two functions are successfully integrated in mainframe environments. Current UNIX offerings for partitioning and workload management have clear architectural differences. Partitioning creates isolation between multiple applications running on a single server, hosting multiple instances of the operating system. Workload management supplies effective management of multiple, diverse workloads to efficiently share a single copy of the operating system and a common pool of resources.

### **1.1.1 Mainframe partitioning**

Mainframes first addressed the need to isolate application environments from each other through physical partitioning. A certain degree of operator intervention was involved when resizing the physical partitions, and applications had to be quiesced before boundaries could be shifted. This explicit management burden limited the use of physical partitions as a tool to respond to fluctuating workload needs.

About 15 to 20 years ago, mainframe developers replaced physical partitions with logical partitions (LPARs). They also created an additional layer of resource management across partitions by specifying time-slicing parameters. With these functions, logical partitions provided a much finer degree of granularity than physical partitions.

At the same time, mainframe developers produced workload management tools. Systems were now able to respond dynamically to fluctuating loads. These tools were implemented as a kernel function within each of the mainframe's SMP partitions.

### **1.1.2 UNIX partitioning**

In current UNIX environments, partitioning means splitting a hardware system into specific hardware boundaries (partitions), and then running a separate copy of the operating system on each partition. The copy of the operating



system may execute on a different level on each partition. This can be done on SMP or NUMA systems.

Generally each partition must include a basic set of resources to boot and execute a copy of the operating system, including:

- At least one processor
- A minimum amount of memory (hardware/operating system dependent)
- I/O devices for boot and application dependent functions
- HW interrupt controller functions

Different hardware vendors have implemented partitioning in various ways, for example:

- Real LPAR (individual processors, I/O boards, memory, and so forth can be independently assigned to a partition)
- Hardware based partitioning solutions based on hardware building blocks (for example four processors, associated memory, and dedicated PCI buses), which is referred to as physical partitioning (PPAR)
- Virtual LPAR (shared processors, non-shared PCI buses, and non-shared memory)

A diagram of partitioning is shown in Figure 1 on page 4.

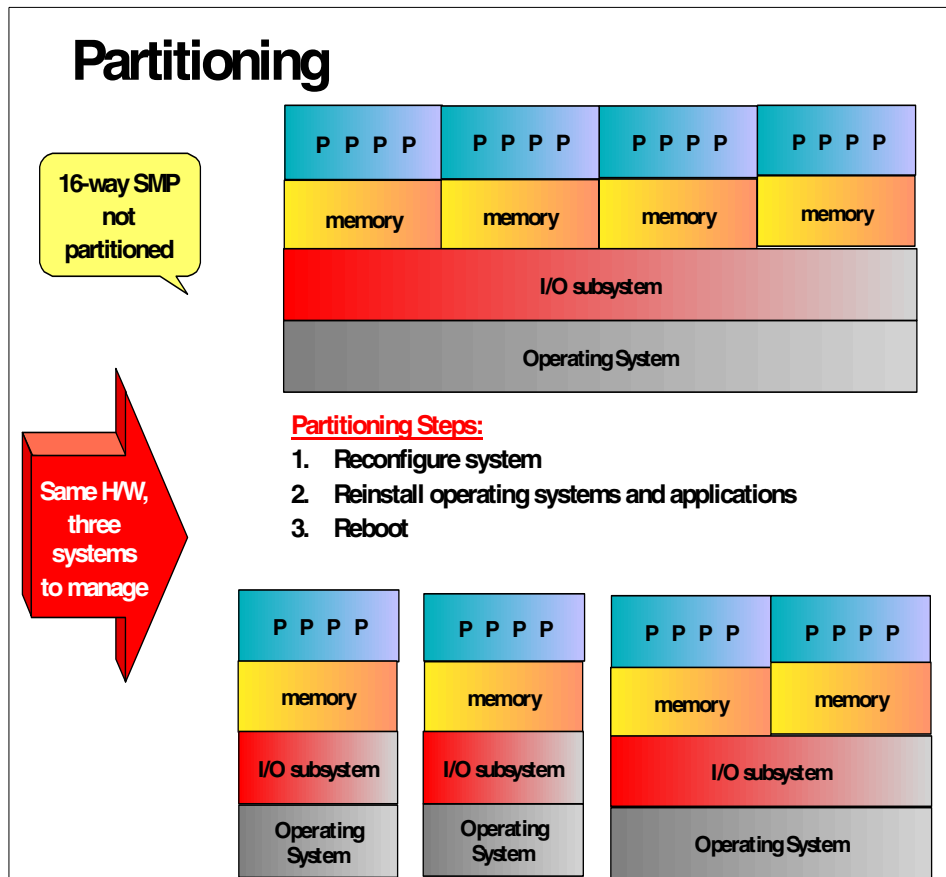


Figure 1. Partitioning

Partitioning can be used to solve several problems, such as running production and test versions of an operating system or application on different partitions for verification or certification purposes. Partitioning can also be used for operating system or application fault isolation, as a software fault condition on a partition does not affect the other partitions. However, global hardware problems such as system down will affect all partitions.

Extra resources are needed because each partition requires its own copy of the operating system, each of which must have to be managed as an individual system.

Because resources - in the current partitioning implementations - have constraints that restricts flexibility, they cannot easily be switched from one partition to another. Free resources on one partition will be wasted.

A more flexible solution to this problem is provided by workload management products such as the AIX Workload Manager (WLM).

### 1.1.3 Workload management

Workload management allows the system administrator to divide resources between jobs without having to partition the system as shown in Figure 2.

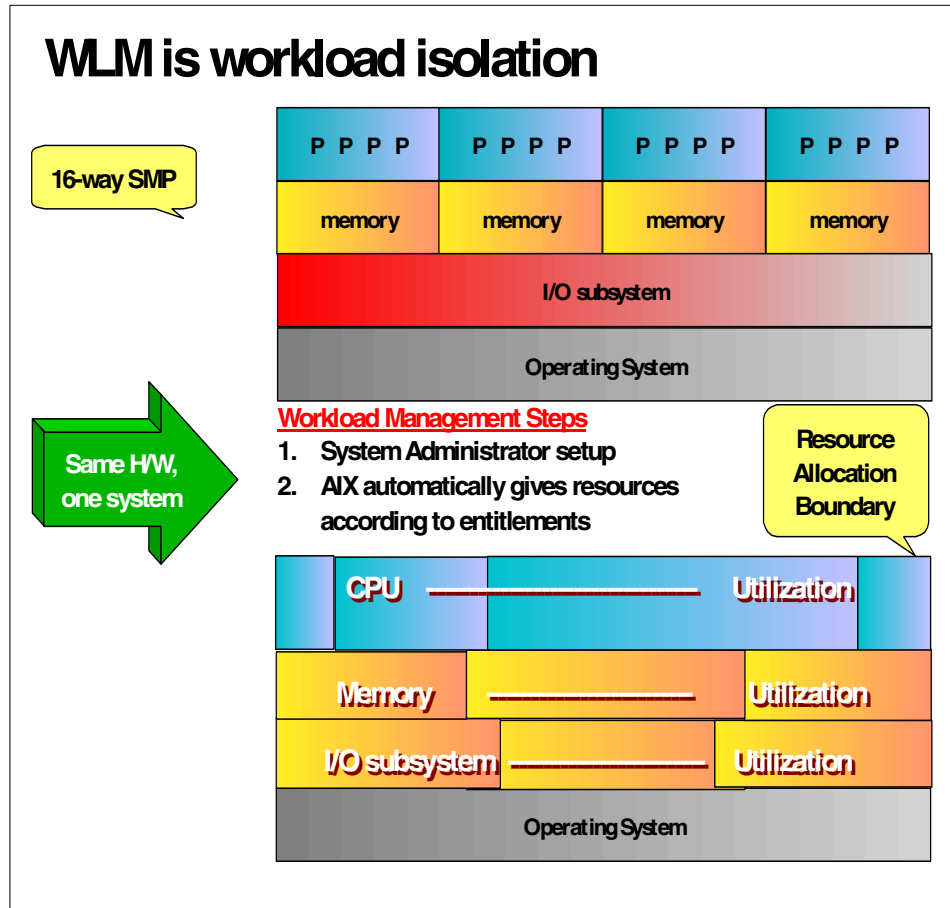


Figure 2. WLM is workload isolation

WLM provides isolation between user communities with very different system behaviors. This can prevent effective starvation of workloads with certain characteristics, such as interactive or low CPU usage jobs, by workloads with other characteristics such as batch or high memory usage jobs.

The setup of WLM is much simpler than partitioning, where reinstallation and reconfiguration are required. With WLM, a single operating system manages the entire system and all jobs, so only one system has to be administered.

WLM manages percentages of CPU time rather than CPUs. This allows control over CPU resources at a finer granularity.

CPU time, memory, and I/O bandwidth are managed separately. Therefore, different styles of applications can be managed.

AIX Workload Manager (WLM) is an operating system feature introduced in AIX Version 4.3.3 and enhanced in AIX 5L. It is part of the operating system kernel at no additional charge.

AIX WLM delivers the basic ability to give system administrators more control over how scheduler, Virtual Memory Manager (VMM), and device driver calls allocate CPU, physical memory, and I/O bandwidth to classes-based users, groups, application paths, process types, or application tags. It allows a hierarchy of classes to be specified, processes to be automatically assigned to classes by their characteristics, and manual assignment of processes to classes.

Classes can be superclasses or subclasses.

AIX WLM self-adjusts when there are no jobs in a class or when a class does not use all the resources that are allocated for it. The resources will automatically be distributed to other classes to match the policies of the system administrator.

Because scheduling is done within a single AIX operating system, system management is far less complex.

Unlike LPAR, workload management does not allow multiple operating systems.

---

## **1.2 The purpose of AIX WLM**

Customers, system administrators, performance consultants, and managers should be aware that Workload Manager is not a tuning tool. AIX WLM is a

resource management tool that specifies the relative importance of each workload by classes, tiers, limits, shares, and rules.

WLM is ideally suited to balance the demands or requests of competing workloads when one or more resources are constrained. It prevents a relatively uncontrolled way of resource scheduling for different applications on the system. Administrators are spared from writing complex scripts.

Before sizing a consolidated system (see Chapter 7, “Sizing recommendations for Workload Manager” on page 199) by putting two or more systems on a single, more powerful server, one thing is vital; know your workload. It is very important that you understand the requirements of the workloads on each individual server that you are planning to incorporate onto the consolidated server.

Your application vendor might provide you with recommendations for system sizing. It is more important is, however, that you create application documentation based on your actual workloads in addition to that, which means gathering detailed information on usage and capacity for your individual systems. This can be done by performance monitoring. Document the workload behavior in a standalone situation, that is, on each traditional single workload server. After migrating from the standalone servers to the consolidated server, which might have improvements in CPU performance or internal and external bus bandwidth, the workload behavior should again be documented so that you can compare future changes to this relative load. After this, you can start implementing different WLM configurations and testing what works best for you.

The same applies if WLM is used on a server that already has several different workloads running. Get a baseline first by monitoring the system performance without WLM, then implement different WLM configurations, and monitor each of them in order to decide which one works best in your environment. Chapter 8, “Practical experience” on page 223, provides helpful examples on how this can be done.

**8** AIX 5L Workload Manager (WLM)

## Chapter 2. AIX Workload Manager functionality

AIX Workload Manager (WLM) is an operating system feature released with AIX Version 4.3.3. This chapter focuses on WLM's functionality, which is available with AIX 5L Version 5.1 and AIX Version 4.3.3 maintenance level 8. The following outlines the enhancements AIX WLM offers over earlier releases:

With AIX Version 4.3.3 maintenance level 2 (APAR IY06844), additional features were added to the first release of WLM. These were:

- Classification of existing processes to avoid stopping and starting applications when stopping and starting WLM.
- Passive mode to allow “before” and “after” WLM comparisons.
- Management of application file names, which allowed WLM to start even if some applications listed in the rules file could not be accessed.

AIX 5L Version 5.0 introduced the following enhancements:

- Management of disk I/O bandwidth in addition to the already-existing CPU cycles and real memory.
- Graphical display of resource utilization.
- Performance Toolbox integration with WLM classes enabling the toolbox to display WLM performance statistics.
- Fully-dynamic configuration including setup of new classes without restarting WLM.
- Application Programming Interface (API) to enable external applications to modify the system's behavior.
- Manual reclassification of processes, which provides the ability to have multiple instances of the same application in different classes.
- More application isolation and control:
  - New subclasses add ten times the granularity of control (from 27 to 270 controllable classes).
  - Administrators can delegate subclass management to other users and groups rather than root or system.
  - Inheritance of classification from parent to child processes.
- Application path name wildcard flexibility extended to user name and group name.

- Resource sets can be defined to limit the set of resources to which a given class has access in terms of CPUs (processor set).
- Tier separation enforced for all resources, enabling a deeper prioritization of applications.

Additions to WLM with AIX 5L Version 5.1:

- A new class attribute was added, `localshm`, that indicates whether memory segments accessed by processes in different classes remain local to the class they were initially assigned to or if they go to the Shared class.
- Class accounting allows to collection and reporting of the use of various system resources by WLM class.
- New graphical performance tools, `wlmmmon` and `wlmpref`, support long term recordings and analysis.

The following additions were introduced in AIX Version 4.3.3 maintenance level 8:

- Manual assignment to a specific superclass or subclass, which is useful to manage different instances of the same application.
- Class inheritance that indicates whether or not a child process should inherit its parent class or be classified according to the automatic assignment rules upon `exec()`.
- Ability to have CPU maximum limits be *hard* limits.
- Ability to prevent shared memory segments to go into the shared class when accessed by processes in different classes.
- Tier separation has been enhanced.

---

## 2.1 Overview

WLM gives the system administrator the ability to create different classes of service for jobs and to specify attributes for those classes. These attributes specify minimum and maximum amounts of CPU, physical memory, and disk I/O throughput to be allocated to a class. WLM then classifies jobs automatically to classes using class assignment rules provided by a system administrator. These assignment rules are based on the values of a set of attributes of a process. The system administrator or a privileged user can also manually assign jobs to classes, thereby overriding the automatic assignment. The basic WLM elements are depicted in Figure 3 on page 11.



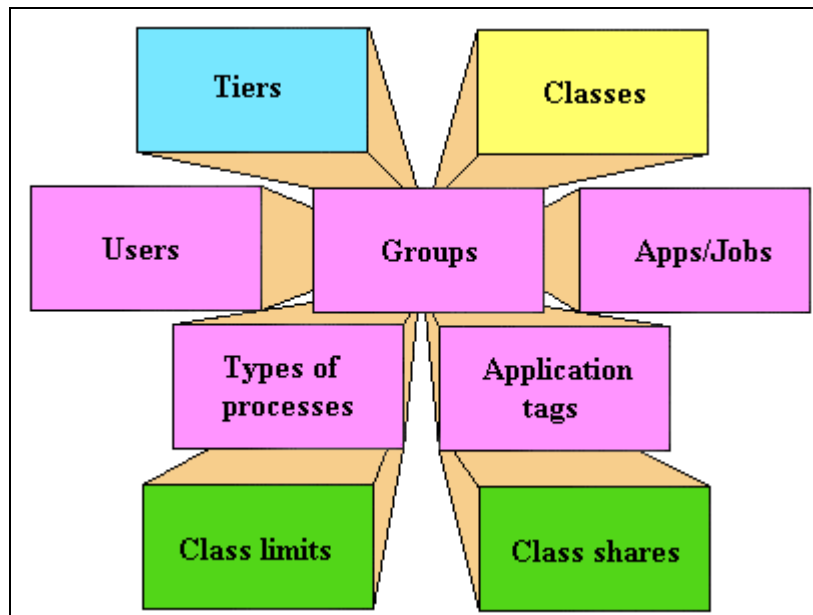


Figure 3. Basic WLM elements

This way, WLM monitors and regulates the CPU utilization of threads, physical memory consumption, and disk I/O bandwidth use of processes active on the system. The manner in which the resources are regulated is dependent on the WLM configuration defined by the system administrator.

There are a number of controlling variables in WLM that facilitate managing classes of jobs to achieve the automatic application of resource entitlement policy you define (Figure 3). The primary concept to remember is that classes are what you manage in WLM, and there are five job attributes available for process identification; *users*, *groups*, *application path names*, *process types*, and *application tags* (application tags are set by the WLM API). Class resource shares and class resource limits allow you to define resource entitlements for each class. Tiers allow you to prioritize groups of classes.

WLM configuration can be performed through direct editing of the configuration files and AIX commands, or through the AIX administration tools, SMIT, or Web-based System Manager (WSM) graphical user interface.

---

## 2.2 Classes

The central concept of WLM is the class. A class is a collection of processes (jobs) that has a single set of resource limits applied to it. WLM assigns processes to the various classes and controls the allocation of system resources among the different classes. For this purpose, WLM uses class assignment rules and per-class resource shares and limits set by the system administrator. The resource entitlements and limits are enforced at the class level. This is a way of defining classes of service and regulating the resource utilization of each class of applications to prevent applications with very different resource utilization patterns from interfering with each other when they are sharing a single server.

### 2.2.1 Hierarchy of classes

WLM allows system administrators to set up a hierarchy of classes with two levels by defining superclasses and subclasses. In other words, a *class* can either be a *superclass* or a *subclass*. The main difference between superclasses and subclasses is the resource control (shares and limits):

- At the superclass level, the determination of resource entitlement based on the resource shares and limits is based on the total amount of each resource managed by WLM available on the machine.
- At the subclass level, the resource shares and limits are based on the amount of each resource allocated to the parent superclass.

The system administrator (the root user) can delegate the administration of the subclasses of each superclass to a *superclass administrator* (a non-root user), thus allocating a portion of the system resources to each superclass and then letting superclass administrators distribute the allocated resources among the users and/or applications they manage.

WLM supports 32 superclasses (27 user defined plus five predefined). In turn, each superclass can have 12 subclasses (10 user-defined and two predefined). Depending on the needs of the organization, a system administrator can decide to use only superclasses or both superclasses and subclasses. He or she can also use subclasses for only some of the superclasses.

Each class is given a name by the WLM administrator who creates it. A class name is up to 16 characters long and can contain only uppercase and lowercase letters, numbers, and underscores (\_). For a given WLM configuration, the names of all the superclasses must be different from one another, and the names of the subclasses of a given superclass must be

different from one another. Subclasses of different superclasses can have the same name. The fully-qualified name of a subclass is *superclass\_name.subclass\_name*.

In the remainder of this chapter, whenever the term *class* is used, it is applicable to both subclasses and superclasses. The following sections describe both super- and subclasses in greater detail as well as the backward compatibility WLM provides to configurations of its first release.

## 2.2.2 Superclasses

A superclass is a class with subclasses associated with it. No processes can belong to the superclass without also belonging to a subclass, either predefined or user-defined. A superclass has a set of class assignment rules that determines which processes will be assigned to it. A superclass also has a set of resource limitation values and resource target shares that determine the amount of resources that can be used by processes belonging to it. These resources will be divided among the subclasses based on the resources limitation values and resource target shares of the subclasses.

Up to 27 superclasses can be defined by the system administrator. In addition, five superclasses are automatically created to deal with processes, memory, and CPU allocation as follows:

- *Default* superclass: The default superclass is named Default and is always defined. All non-root processes that are not automatically assigned to a specific superclass will be assigned to the Default superclass. Other processes can also be assigned to the Default superclass by providing specific assignment rules.
- *System* superclass: This superclass has all privileged (root) processes assigned to it if they are not assigned by rules to a specific class, as well as the pages belonging to all system memory segments, kernel processes, and kernel threads. Other processes can also be assigned to the System superclass. The default is for this superclass to have a memory minimum limit of one percent.
- *Shared* superclass: This superclass receives all the memory pages that are shared by processes in more than one superclass. This includes pages in shared memory regions and pages in files that are used by processes in more than one superclass (or in subclasses of different superclasses). Shared memory and files that are used by multiple processes that all belong to a single superclass (or subclasses of the same superclass) are associated with that superclass. Only when a process from a different superclass accesses the shared memory region or file are the pages placed in the Shared superclass. This superclass

can have only physical memory shares and limits applied to it. It cannot have shares or limits for the other resource types, subclasses, or assignment rules. Whether a memory segment shared by processes in different superclasses, is classified into the Shared superclass, or remains in the superclass it was initially classified into depends on the value of the *localshm* attribute of the superclass the segment was initially classified into. For further information on localshm, refer to Section “Localshm” on page 21.

- *Unclassified* superclass: The processes in existence at the time WLM is started are classified according to the assignment rules of the WLM configuration being loaded. During this initial classification, all the memory pages attached to each process are charged either to the superclass to which the process belongs (when not shared or shared by processes in the same superclass) or to the Shared superclass when shared by processes in different superclasses. However, there are a few pages that cannot be directly tied to any processes (and, thus, to any class) at the time of this classification, and this memory is charged to the Unclassified superclass. An example for that would be pages from a file that has been closed. The file pages will remain in memory, but no process really *owns* these pages; therefore, they cannot be charged to any specific class. Most of this memory will end up being correctly reclassified over time, when it is either accessed by a process or freed and reallocated to a process after WLM is started. Thereafter only a few kernel processes, such as wait or lrud, will appear in the Unclassified superclass. This superclass cannot have physical memory shares and limits applied to it, nor can subclasses or assignment rules be specified on this superclass.
- *Unmanaged* superclass: A special superclass, named Unmanaged, will always be defined. No processes will be assigned to this class. This class will be used to accumulate the memory usage for all pinned pages in the system that are not managed by WLM. The CPU utilization for the waitprocs is not accumulated in any class. This is done deliberately; otherwise, the system would always seem to be at 100 percent CPU utilization, and it could be misleading for users when looking at the WLM or system statistics. This superclass cannot have shares or limits for any resource types, subclasses, or assignment rules specified.

### 2.2.3 Subclasses

A subclass is a class associated with exactly one superclass. Every process in the subclass is also a member of the superclass. Subclasses only have access to resources that are available to the superclass. A subclass has a set of class assignment rules that determine which of the processes assigned to the superclass will belong to it. A subclass also has a set of resource

limitation values and resource target shares that determine the resources that can be used by processes in the subclass. These resource limitation values and resource target shares indicate how much of the superclass' target (the resources available to the superclass) can be used by processes in the subclass.

Up to 10 out of total 12 subclasses can be defined by the system administrator or by the superclass administrator for each superclass. The two special subclasses, Default and Shared, are always defined in each superclass as follows:

- *Default* subclass: The Default subclass is always defined. All processes that are not automatically assigned to a specific subclass of the superclass will be assigned to the Default subclass. You can also assign other processes to the Default subclass by providing specific assignment rules.
- *Shared* subclass: This subclass receives all the memory pages that are used by processes in more than one subclass of the superclass. This includes pages in shared memory regions and pages in files that are used by processes in more than one subclass of the same superclass. Shared memory and files that are used by multiple processes that all belong to a single subclass are associated with that subclass. It is only when a process from a different subclass of the same superclass accesses the shared memory region or file that the pages are placed in the Shared subclass of the superclass. There are no processes in the Shared subclass. This subclass can only have physical memory shares and limits applied to it. It cannot have shares or limits for the other resource types or assignment rules specified. Whether a memory segment shared by processes in different subclasses of the same superclass is classified into the Shared subclass or remains in the subclass it was initially classified into depends on the value of the *localshm* attribute of the subclass the segment was initially classified into. For further information on *localshm* refer to Section "Localshm" on page 21.

#### **2.2.4 Backward compatibility considerations**

System administrators have the option of using only superclasses or both superclasses and subclasses in their WLM configurations. The system administrator can also choose to create subclasses only for some superclasses. So, when starting AIX 5L's WLM with configurations created in AIX Version 4.3.3, only superclasses will be used. The default output of the *wlmstat* command, in this case, will show just the superclasses and will be similar to the one users of the first release are familiar with as shown in the the following example.

```
# wlmstat
CLASS      CPU    MEM    DKIO
Unclassified  0     0     0
Unmanaged  0     0     0
Default    0     0     0
Shared     0     2     0
System     2     12    0
db1        0     0     0
db2        0     0     0
devlt      0     4     2
```

If some of the superclasses have subclasses defined by a WLM administrator, the subclasses will be shown in `wlmstat` output as follows:

```
# wlmstat
CLASS      CPU    MEM    DKIO
Unclassified  0     0     0
Unmanaged  0     0     0
Default    0     0     0
Shared     0     2     0
System     3     11    7
db1        46    0     0
db2        48    0     0
devlt      50    0     0
devlt.Shared  0     0     0
devlt.editors 18    0     0
```

The same thing happens with the output of the `ps` command. For processes in a superclass without any subclasses, `ps` will show the superclass name as the process' class name:

```
# ps -ae -o pid,user,class,args
PID  USER      CLASS      COMMAND
1    root      System     /etc/init
5614 dbadmin   db1        /etc/ora_db_writer
5750 dbadmin   db2        /etc/ora_db_writer
5980 jim       devlt.editors /bin/vi
6714 sue      devlt.build  /bin/cc
```

---

## 2.3 Tiers

Tier configuration is based on the importance of a class relative to other classes in WLM. In other words, tiers define the relative priority of groups of classes to each other. There are 10 available tiers from 0 through to 9. Tier

value 0 is the most important and 9 is the least important. As a result, classes belonging to tier 0 will get resource allocation priority over classes in tier 1; classes in tier 1 will have priority over classes in tier 2, and so on. The default tier number, if the attribute is not specified, is 0.

The tier applies at both the superclass and subclass levels. Superclass tiers are used to specify resource allocation priority between superclasses, and subclass tiers are used to specify resource allocation priority between subclasses of the same superclass. There is no relationship between tier numbers of subclasses of different superclasses.

Tier separation in terms of prioritization is much more enforced in AIX 5L than what was observed in the previous release. A process in tier 1 will never have more priority than a process in tier 0 because there is no overlapping of priorities in tiers. It is highly unlikely that classes in tier 1 will get hold of any resources if processes in tier 0 are using up all of them. This occurs because the control of *leftover resources* is much more restricted than what was happening in WLM's first AIX Version 4.3.3 release.

---

## 2.4 Class attributes

The attributes of a class are as follows:

- *Class name*: Up to 16 characters long. Can contain only uppercase and lowercase letters, numbers, and underscores (\_).
- *Tier*: Number between 0 and 9 for class priority ranking.
- *Inheritance*: Specifies whether or not a child process inherits the class assignment from its parent.
- *Adminuser, admingroup* (superclass only): Used to delegate the administration of a superclass.
- *Authuser, authgroup*: Used to delegate the right to manually assign a process to a class.
- *Resource Set*: Used to limit the set of resources to which a given class has access in terms of CPUs (processor set).
- *Localshm*: Specifies whether memory segments that are accessed by processes in different classes remain local to the class they were initially assigned to, or if they go to the Shared class.

### **Tier**

This attribute holds the tier number to which the class belongs. It is used to prioritize resource allocation between classes. Refer to Section 2.3 on page 16 for further details.

### **Inheritance**

The inheritance attribute indicates whether or not a child process should inherit its parent's class or be classified according to the automatic assignment rules upon `exec`. The possible values are `yes` or `no`, and the default is `no`. This attribute can be specified at both the superclass and subclass level. See Table 1 for details.

Table 1. Inheritance attribute at superclass and subclass level meaning

Superclass level inheritance value	Subclass level inheritance value	Meaning
yes	yes	A child of a process in the subclass will remain in the same subclass upon <code>exec</code> .
yes	no or unspecified	A child of a process in the subclass will remain in the same superclass and will be classified in one of its subclasses according to the assignment rules for the superclass upon <code>exec</code> .
no or unspecified	yes	A child of a process in the subclass will be submitted to the automatic assignment rules for the superclasses upon <code>exec</code> . If the process is classified by the rules in the same superclass, it will remain in the subclass (it will not be submitted to the subclasses assignment rules). If the process is classified by the superclass rules in a different superclass, the subclass assignment rules of the new superclass are applied to determine the subclass of the new superclass to which the process will be assigned.
no or unspecified	no or unspecified	A child of a process in the subclass will be submitted to the standard automatic assignment upon <code>exec</code> .

The inheritance attribute has a different reading when manual assignment is being used. This feature is fully-described in Section 5.1.3, "Interaction with inheritance" on page 179. Additionally, tag inheritance from parent to child



processes is available when application tagging is being used. This subject is covered in Section 6.1, “Application tag” on page 195.

### **Inheritance with AIX Version 4.3.3 maintenance level 8**

AIX Version 4.3.3 maintenance level 8 now supports a new class attribute called inheritance. This attribute can be added to the classes file in the same way the tier attribute is. The default value is no, meaning that new processes will be automatically classified upon calling `exec`. If the inheritance attribute is set to yes, a new process created in a specific class will remain in that class regardless of which application it executes. The syntax in the classes file is similar to the syntax for the other attributes.

Below an excerpt from a configuration in the classes file:

```
student:  
    tier = 1  
    inheritance = yes
```

Inheritance can be used together with manual assignment to assign an application to a class when the application starts and make sure that all the processes spawned by the application remain in the class the application was manually assigned to. It is also possible to use inheritance independently of manual assignment.

#### **Note**

Inheritance on AIX Version 4.3.3 maintenance level 8 can be set/reset only from the command line (`mkclass/chclass`), and is not supported in SMIT and WSM.

### ***Adminuser, admingroup***

These attributes are valid only for superclasses. They are used to delegate the superclass administration to a user and/or group of users:

- *Adminuser* specifies the name of the user (as listed in `/etc/passwd`) authorized to perform administration tasks on the superclass. This can also be an NIS user.
- *Admingroup* specifies the name of the group of users (as listed in `/etc/group`) authorized to perform administration tasks on the superclass. This can also be an NIS group.

Only one value (user/group name) is allowed for each attribute. Any one of them, none, or both can be specified. The user and/or group has authority to create/delete subclasses; change the attributes, resource shares, and limits for the subclasses; define, remove, or modify subclass assignment rules; and

refresh (update) the active WLM configuration for the superclass. In addition, root always has authority on any superclass.

### ***Authuser, authgroup***

These attributes are valid for all the classes. They are used to specify the user name and/or the group name of the user and/or group authorized to manually assign processes to the class. When manually assigning a process (or a group of processes) to a superclass, the assignment rules for the superclass are used to determine which subclass of the superclass each process will be assigned to.

- *Authuser* specifies the name of the user (as listed in /etc/passwd) authorized to manually assign processes to the class.
- *Authgroup* specifies the name of the group of users (as listed in /etc/group) authorized to manually assign processes to the class.

Only one value (user/group name) is allowed for each attribute. Any one of them, none, or both can be specified. In addition, root and the administrators of a superclass specified by adminuser/admingroup can always manually assign processes to a superclass or to a subclass of the superclass.

### ***Resource set (rset)***

This attribute is valid for all the classes. Resource sets are an operating system feature introduced in AIX 5L. This feature allows the system administrator to define subsets of system resources through SMIT or WSM and give them a name using a new registry service.

WLM uses the concept of resource sets (or rsets) to restrict the processes in a given class to a subset of the system's physical memory and processors. A valid resource set is composed of memory (currently only one domain shared by all resource sets) and at least one processor.

Using SMIT or Web-based System Manager, a system administrator has the ability to define and name resource sets containing a subset of the resources available on the system. Then, using the WLM administration interfaces, root or a designated superclass administrator can use the name of the resource set as the *rset* attribute of a WLM class. From then on, every thread assigned to this WLM class is only dispatched on one of the processors in the resource set. This is a very effective way of further separating workloads for the CPU resource. Refer to Section 3.2.7, "Working with resource sets" on page 92 for further information on resource sets.

Because all of the current systems have only one memory domain shared by all the resource sets, this method does not allow the physical separation of workloads in memory.

### ***Localshm***

This attribute can be specified at the superclass and the subclass level. It is used to prevent memory segments belonging to a class to migrate to the Shared superclass or the Shared subclass when they are accessed by processes in different classes. The possible values for the attribute are *yes* or *no*. *Yes* means that shared memory segments in this class should remain local to the class and not migrate to the appropriate Shared class. *No* allows migration, and is the default.

Memory segments are classified on page faults. When a segment is created, it is marked as belonging to the Unclassified superclass. On the first page fault on the segment, this segment is classified into the same class as the faulting process. If, later on, a process belonging to a different class than the segment page faults on this segment, WLM determines whether the segment needs to be reclassified into the appropriate Shared class (superclass or subclass).

If the faulting process and the segment belong to different superclasses:

- If the segment's superclass has the *localshm* attribute set to *yes*, the segment remains in the superclass. If the segment's subclass has the *localshm* attribute set to *yes*, the segment remains in the subclass. Otherwise, it goes into the Shared subclass of the superclass.
- If the segment's superclass has the *localshm* attribute set to *no*, the segment goes to Shared.Default.

If the faulting process and the segment belong to different subclasses of the same superclass:

- If the segment's subclass has the *localshm* attribute set to *yes*, the segment remains in the same class (superclass and subclass). Otherwise, the segment goes to the Shared subclass of the superclass.

Of course, if the faulting process and the segment belong to the same class (same superclass and same subclass) there is no need to reclassify regardless of the values of the *localshm* attribute.

In AIX Version 4.3.3 maintenance level 8 a flag can be set with the `wlmset` command to provide the *localshm* functionality on a system wide basis (for all the classes). Please refer to Section “AIX Version 4.3.3 maintenance level 8 `wlmset` command” on page 81 for further information.

---

## 2.5 Classification process

There are two ways to classify processes in WLM:

- Automatic assignment when a process calls the system call, `exec`, using assignment rules specified by a WLM administrator. This automatic assignment is always in effect (cannot be turned off) when WLM is active. This is the most common method of assigning processes to the different classes.
- Manual assignment of a selected process or group of processes to a class by a user with the required authority on both the process and target class. This manual assignment can be done either by a WLM command, which can be invoked directly, through SMIT or WSM, or by an application using a function of the WLM Application Programming Interface. Manual assignment overrides automatic assignment.

### 2.5.1 Automatic assignment

The automatic assignment of processes to classes uses a set of class assignment rules (see also Section 2.5.3, “Class assignment rules” on page 23) specified by a WLM administrator. There are two levels of assignment rules:

- A set of assignment rules at the WLM configuration level used to determine which superclass a given process should be assigned to
- A set of assignment rules at the superclass level used to determine which subclass of the superclass the process should be assigned to

The assignment rules at both levels have exactly the same format.

When a process is created (`fork`), it remains in the same class as its parent (for more information on inheritance, see Section 5.1.3, “Interaction with inheritance” on page 179). By the time the new process calls `exec`, WLM checks the assignment rules to decide if the process should be assigned to another class and, if necessary, initiates the reclassification. In order to classify the process, WLM starts by examining the top level rules list for the active configuration to find out which superclass the process should belong to. For this purpose, WLM applies the rules one at a time in the order in which they appear in the file and checks the current values for the process attributes against the values and lists of values specified in the rule. When a match is found, the process will be assigned to the superclass named in the first field of the rule. The rules list for the superclass is examined in the same way to determine which subclass of the superclass the process should be assigned to. For a process to match one of the rules, each of its attributes

must match the corresponding field in the rule. The rules to determine whether the value of a process attribute matches the values in the field of the rules list are as follows:

- If the field in the rule has a value of hyphen (-), any value of the corresponding process attribute is a match.
- If the value of the process attribute (for all the attributes except *type*) matches one of the values in the list in a rule and it is not excluded (prefaced by a (!)), it is considered a match.
- When one of the values for *type* attribute in the rule is comprised of two or more values separated by a plus (+) sign, a process will be a match for this value only if its characteristics match all the values mentioned above.

As stated before, at both the superclass and subclass levels WLM goes through the rules in the order in which they appear in the rules list and classifies the process in the class corresponding to the first rule for which the process is a match. This means that the order of the rules in the rules list is extremely important, and caution must be used when modifying it in any way.

### 2.5.2 Manual assignment

In addition to automatic class assignment, a user with the proper authority can manually assign processes or groups of processes to a specific superclass or subclass. This feature is described in greater detail in Chapter 5, “Manual assignment” on page 167.

### 2.5.3 Class assignment rules

After the definition of a class, it is time to set up the class assignment rules so that WLM can perform its automatic assignment. The assignment rules are used by WLM to assign a process to a class based on the *user*, *group*, *application pathname*, *type of process*, and *application tag*, or a combination of these five attributes. These class assignment rules are stored in a WLM configuration file in this explicit order (for more information see Section “rules file” on page 50).

The next sections describe all attributes that constitute a class assignment rule. All these attributes can contain a hyphen (-), which indicates that they are not specified.

#### ***Class name***

This field must contain the name of a class that is defined in the class file corresponding to the level of the rules file we are configuring (either superclass or subclass). Class names can contain only uppercase and

lowercase letters, numbers, and underscores (`_`), and can be up to 16 characters in length. No assignment rule can be specified for the system defined classes *Unclassified*, *Unmanaged*, and *Shared*.

### **Reserved**

Reserved for future use. Its value *must* be a hyphen (`-`), and it must be present in the rule.

### **Users**

The user name (as specified in the `/etc/passwd` file or in NIS) of the user owning a process can be used to determine the class to which the process belongs. This attribute is a list of one or more user names separated by a comma (`,`). Users can be excluded by using an exclamation point (`!`) prefix. Patterns can be specified to match a set of user names using full Korn shell pattern matching syntax.

Applications that use the `setuid` permission to change the *effective* user (UID) under which they run are still classified according to the user that invoked them. The processes are only reclassified if the change is done to the *real* UID.

### **Groups**

The group name (as specified in the `/etc/group` file or in NIS) of a process can be used to determine the class to which the process belongs. This attribute is a list composed of one or more groups separated by a comma (`,`). Groups can be excluded by using an exclamation point (`!`) prefix. Patterns can be specified to match a set of group names using full Korn shell pattern matching syntax.

Applications that use the `setgid` permission to change the *effective* group ID (GID) under which they run are still classified according to the group that invoked them. The processes are only reclassified if the change is done to the *real* GID.

### **Application pathnames**

The full pathname of the application for a process can be used to determine the class to which a process belongs. This attribute is a list composed of one or more applications and separated by a comma (`,`). The application pathnames will be either full pathnames or Korn shell patterns that match pathnames. Application pathnames can be excluded by using an exclamation point (`!`) prefix.

### **Process types**

In AIX 5L, the process type attribute was introduced as one of the ways to determine the class to which a process belongs. This attribute is a comma (,) separated list of single values or combinations of two or more single values joined with plus signs (+). A plus sign (+) means *AND*, and a comma (,) means *OR*. For example:

- 64bit,plock+fixed
- plock+fixed+64bit,32bit
- plock,fixed,64bit

The list of values that can figure on this attribute is shown in the following section. 32 bit and 64 bit are mutually exclusive:

#### **Attribute value Process type**

<b>32bit</b>	The process is a 32 bit process.
<b>64bit</b>	The process is a 64 bit process.
<b>plock</b>	The process called plock() to pin memory.
<b>fixed</b>	The process is a fixed priority process (SHED_FIFO or SCHED_RR).

### **Application tags**

In AIX 5L, the application tag attribute was introduced as one of the forms of determining the class to which a process belongs. This is an attribute meant to be set by WLM's API as a way of further extending the process classification possibilities. This was created with the primary purpose of allowing differentiated classification for different instances of the same application. This attribute can have one or more application tags separated by commas (,). An application tag is a string of up to 30 alphanumeric characters.

#### **2.5.3.1 Process classification**

The classification is done by comparing the value of the attributes of the process at `exec` time against the lists of class assignment rules to determine which rule is a match for the current value of the process attributes. The class assignment is done by WLM:

- When WLM is started for all the processes existing at that time.
- Every time a process calls the system calls `exec`, `setuid` (and related calls), `setgid` (and related calls), `setpri`, and `plock` once WLM is started.

This behavior can be altered using the inheritance attribute as explained in Section “Inheritance” on page 18.

There are two *default* rules that are always defined (that is, hardwired in WLM). These are the default rules to assign all processes started by the user root to the System class, and all other processes to the Default class. If WLM does not find a match in the assignment rules list for a process, these two rules will be applied (the rule for System first), and the process will go to either System (uid root) or Default. These default rules are the only assignment rules in the standard configuration installed with AIX. In the example of Table 2, the rule for Default class is omitted from display, although this class’ rule is always present in the configuration.

Table 2. Examples of class assignment rules

Class	Reserved	User	Group	Application	Type	Tag
System	-	root	-	-	-	-
db1	-	-	-	/usr/oracle/bin/db*	-	_db1
db2	-	-	-	/usr/oracle/bin/db*	-	_db2
devt	-	-	dev	-	32bit	-
VPs	-	bob, sally	-	-	-	-
acctg	-	!ted	acct*	-	-	-

The rule for System is explicit and has been put first in the file. This is done deliberately so that *all* processes started by root will be assigned to the System superclass. By moving the rule for the System superclass further down in the rules file, the system administrator could have chosen to assign to System only the root processes that would not be assigned to another class (because of the application executed, for instance). In the example shown in Table 2, with the rule for System on top, if root executes a program in /usr/oracle/bin/db\* set, the process will be classified as System. If the rule for the System class were after the rule for the db2 class, the same process would be classified as db1 or db2 depending on the tag.

These examples show that the order of the rules in the assignment rules file is very important. The more specific assignment rules should appear first in the rules file, and the more general rules should appear last. An extreme example would be putting the default assignment rule for the Default class, for which every process is a match, first in the rules file. That would cause every process to be assigned to the Default class. The other rules would, in effect, be ignored.



You can define multiple assignment rules for any given class. You can also define your own specific assignment rules for the System and/or Default classes. The default rules mentioned above for these classes would still be applied to processes that would not be classified using any of the explicit rules.

---

## 2.6 WLM class accounting

The AIX accounting system utility allows you to collect and report on individual and group use of various system resources. This accounting information can be used to bill users for the system resources they utilize and to monitor selected aspects of the system's operation. To assist with billing, the accounting system provides the resource-usage totals defined by members of the adm group, and, if the `chargefee` command is included, factors in the billing fee. The accounting system also provides data to assess the adequacy of current resource assignments, set resource limits and quotas, forecast future needs, and order supplies for printers and other devices. The accounting system has been enhanced with AIX 5L Version 5.1 to allow accounting for WLM classes.

### 2.6.1 Process accounting using WLM class

The accounting system collects the resources used by the users' processes. To gather these resources, the accounting system needs the assistance of the kernel. The kernel, whenever a process exits, appends a record with the process' accounting information to the process accounting file, normally `/var/adm/pacct`. The system collects data on resource usage for each process as it runs. These data include:

- User and group numbers under which the process runs
- First eight characters of the name of the command
- A 64 bit numeric key representing the Workload Manager class the process belongs to
- Elapsed time and processor time used by the process
- Memory usage
- Number of characters transferred
- Number of disk blocks read or written on behalf of the process

### 2.6.1.1 Displaying WLM class accounting information

The AIX accounting command `acctcom` allows displaying process resource usage statistics per user, group, or WLM class from the `/var/adm/pacct` file. The default output of the `acctcom` command is shown in the following display:

```
# acctcom | head -10
```

COMMAND			START	END	REAL	CPU	MEAN
NAME	USER	TTYNAME	TIME	TIME	(SECS)	(SECS)	SIZE(K)
#accton	root	pts/9	15:25:55	15:25:55	0.00	0.00	364.00
#bsh	root	pts/9	15:25:55	15:25:55	0.03	0.00	0.00
#rm	root	pts/9	15:25:56	15:25:56	0.02	0.00	0.00
#rm	root	pts/9	15:25:56	15:25:56	0.02	0.00	332.00
#rm	root	pts/9	15:25:56	15:25:56	0.02	0.00	0.00
#bsh	root	pts/9	15:25:55	15:25:55	0.11	0.02	225.00
#bsh	root	pts/9	15:25:55	15:25:55	0.23	0.02	0.00

Two new options have been added to the `acctcom` command to collect accounting information for WLM classes. To select processes belonging to a specific class, you can use the `/usr/sbin/acct/acctcom -c classname` as seen in the display below:

```
## acctcom -c System
```

COMMAND			START	END	REAL	CPU	MEAN
NAME	USER	TTYNAME	TIME	TIME	(SECS)	(SECS)	SIZE(K)
#accton	root	pts/9	15:25:55	15:25:55	0.00	0.00	364.00
#bsh	root	pts/9	15:25:55	15:25:55	0.03	0.00	0.00
#rm	root	pts/9	15:25:56	15:25:56	0.02	0.00	0.00
#rm	root	pts/9	15:25:56	15:25:56	0.02	0.00	332.00
#rm	root	pts/9	15:25:56	15:25:56	0.02	0.00	0.00
#bsh	root	pts/9	15:25:55	15:25:55	0.11	0.02	225.00
#bsh	root	pts/9	15:25:55	15:25:55	0.23	0.02	0.00

You can also use the `/usr/sbin/acct/acctcom -w` option to display the class names to which the processes belong as seen in the next display:

```
# acctcom -w | head -10
```

COMMAND				START	END	REAL	CPU	MEAN
NAME	USER	CLASS	TTYNAME	TIME	TIME	(SECS)	(SECS)	SIZE (K)
#acctcn	root	System.Default	pts/9	15:25:55	15:25:55	0.00	0.00	364.00
#bsh	root	System.Default	pts/9	15:25:55	15:25:55	0.03	0.00	0.00
#rm	root	System.Default	pts/9	15:25:56	15:25:56	0.02	0.00	0.00
#rm	root	System.Default	pts/9	15:25:56	15:25:56	0.02	0.00	332.00
#rm	root	System.Default	pts/9	15:25:56	15:25:56	0.02	0.00	0.00
#bsh	root	System.Default	pts/9	15:25:55	15:25:55	0.11	0.02	225.00
#bsh	root	System.Default	pts/9	15:25:55	15:25:55	0.23	0.02	0.00

For users who wish to write their own reporting and billing applications, the format of the accounting record is published (header file acct.h) and the WLM Application Programming Interface provides the subroutines `wlm_key2class` and `wlm_class2key` to convert the key to a class name or vice versa. These subroutines are explained in detail in Appendix A.9 "WLM accounting" on page 293. The accounting subsystem uses a 64 bit key instead of the full 34 character class name in order to save space (as that would practically double the size of the accounting record). When the accounting command to extract the per process data is run, the key is translated back into a class name using the above mentioned routine. This translation uses the class names that are currently in the WLM configuration files. Therefore, if a class has been deleted between the time the accounting record was written and the time the accounting report is run, the class name corresponding to the key will not be found, and the class will appear as *Unknown*. There are two possibilities for system administrators who wish to keep accurate records of the resource usage of classes deleted during an accounting period:

- Instead of just deleting the class, you can keep the class name in the classes file and remove the class from the rules file so that no process gets assigned to it any more. You can thus delete the class after the accounting report has been generated at the end of the accounting period.
- You can also delete the class from the configuration it belongs to, and keep the class name in the classes file in a *dummy* configuration (which is never activated) until after the accounting records for the period have been generated.

---

## 2.7 Resources

WLM monitors and regulates the resource utilization of the threads and processes active on the system. The monitoring and regulation is done per class. You can set minimum or maximum limits per class for each resource

type managed by WLM. In addition, a target value for each resource per class may be given. This target, named share, is representative of the amount of the resource that would be optimal for the jobs in the class.

The shares and limits at the superclass level refer to the total amount of each resource available on the system. At the subclass level, they refer to the amount of each resource made available to the superclass the subclass is in (superclass' target). The hierarchy of classes is a way for a system administrator to divide up the system resources between groups of users (superclasses) and delegate the administration of this share of the resources to superclass administrators. Each superclass administrator can then redistribute this amount of resources between the users in the group by creating subclasses and defining resource entitlements.

### 2.7.1 Resources managed by WLM

WLM manages three types of resources:

- The CPU utilization of the threads in a class. This is the sum of all the CPU cycles consumed by every thread in the class.
- The physical memory utilization of the processes in a class. This is the sum of all the memory pages that belong to the processes in the class.
- The disk I/O bandwidth of the class. This is the bandwidth (in 512 byte blocks per second) of all the I/Os started by threads of the class on the disk devices accessed.

Once per second WLM calculates the per-class utilization for each resource during the last second as a percentage of the total resource available.

#### ***CPU***

The total amount of CPU time available every second is equal to one second times the number of CPUs on the system. For instance, on an eight-way SMP, if all the threads of a class combined consumed two seconds of CPU time during the last second, this represents a percentage of  $2/8 = 25$  percent. The percentage used by WLM for regulation is a decayed average over a few seconds of this *instantaneous* per-second resource utilization.

#### ***Physical memory***

The total amount of physical memory available for processes at any given time is the total number of memory pages physically present on the system minus the number of pinned pages. The pinned pages are not managed by WLM because these pages cannot be reassigned to another class in order to regulate memory utilization. The memory utilization of a class is simply the ratio of the number of (non-pinned) memory pages being used by all the

processes in the class to the number of pages available on the system, as defined above, expressed as a percentage.

### ***Disk I/O***

For the disk I/O, the main difficulty is to determine a meaningful available bandwidth for a device. When a disk is 100 percent busy, its throughput (in blocks per second) will be very different if one application is doing sequential I/Os than if several applications are doing random I/Os. If the maximum throughput measured for the sequential I/O case was used as a value of the I/O bandwidth available for the device to compute the percentage of utilization under random I/Os, statistical errors would be created. It might lead one to think that the device is, for instance, 20 percent busy, while it is, in fact, at 100 percent utilization.

In order to get more accurate and reliable percentages of per-class disk utilization, WLM uses the data provided by the disk drivers (which are displayed with the AIX `iostat` command) giving, for each disk device, the percentage of the time the device has been busy during the last second. WLM knows how many blocks in total have been read/written on a device during the last seconds by all the classes accessing the device, how many blocks have been read/written by each class, and what the percentage of utilization of the device was. This allows WLM to easily calculate what percentage of the disk throughput was consumed by each class. For instance, if the total number of blocks read or written during the last second was 1000 and the device had been 70 percent busy, it means that a class reading/writing 100 blocks used seven percent of the disk bandwidth. Like CPU time (another renewable resource), the values used by WLM for its disk I/O regulation are also a decayed average of per-second percentages.

For the disk I/O resource, the shares and limits apply to each disk device accessed by the class individually, and regulation is done independently for each device.

## **2.7.2 Class resource shares**

The number of shares of a resource for a class determine the proportion of the resource that is allocated to the processes assigned to the class. In simple terms, the resource shares are specified as relative amounts of usage between different classes in the same tier. One way of thinking about shares is as a self-adapting percentage.

For example, a system has three classes defined, A, B, and C, whose targets are 50, 30, and 20 respectively.

- If all three classes are active, the total number of shares for the active classes is 100. Their targets, expressed as percentages, are 50 percent, 30 percent, and 20 percent.
- If A is not active, the total number of shares is 50 (so, each share represents two percent). The target percentages for B and C are 60 percent and 40 percent.
- If only one class is active, its target is 100 percent.

A class is considered active (regardless of its resource consumption) when it has at least one process assigned to it.

In this example, the sum of the shares for the three classes was 100 simply to make the sample calculation easier. A target can be any number between 1 and 65535.

The preceding example implicitly supposes that:

- A, B, and C are either all superclasses or all subclasses of the same superclass.
- A, B, and C are in the same tier.

The relative share numbers of a subclass and a superclass, of two subclasses of different superclasses, or of classes in different tiers do not give any indication of their relative resource entitlements. As explained earlier, the shares are used by WLM to calculate for each class a percentage goal of resource utilization for each resource type. This goal represents a percentage of resources that can vary widely depending on how many classes are active at any given time. However, WLM makes sure that the dynamic value of this percentage goal remains compatible with the minimum and maximum limits for the class. If the calculated percentage is below the minimum, WLM uses the minimum as the target. If the percentage is above the maximum limit, WLM uses the maximum as the target. If the percentage is between the minimum and the maximum limit, WLM uses the calculated value.

The share number can be specified as a hyphen (-) for any resource type to indicate that the class' resource utilization for this resource type is not regulated by WLM. This is the default when no share value has been specified for a resource type. Note that this default is different from the default value of one share in the first version of WLM.

What exactly does it mean to have a resource type that is not regulated by WLM on a certain class? It means that the resource target for that class will always be 100 percent. WLM will never penalize this class for being above its

target, for there is no such thing as the notion of *WLM target* for this specific resource for this specific class. The consequence is (as expected) that a class with a non-regulated resource in tier 0 is capable of starving all the other classes for this resource. It is, therefore, recommended to reserve this non-regulated value for consistently well-behaved classes, such as System, for instance.

The example shown in Figure 4 displays resource allocation before and after a new class is activated. Initially, there are three active classes that have been allocated five, seven, and two resource shares respectively. These resource shares in combination are allocated 100 percent of the resource in accordance with their relative share values. When the new class, which has three resource shares, is activated, there are four active classes with resource shares of five, seven, two, and three with the total active resource shares equal to 17. As a result, when all four classes are active, the class with five resource shares will be allocated five of the total of 17 shares or 29 percent of the system resource (29.4 percent will be rounded down to 29 percent).

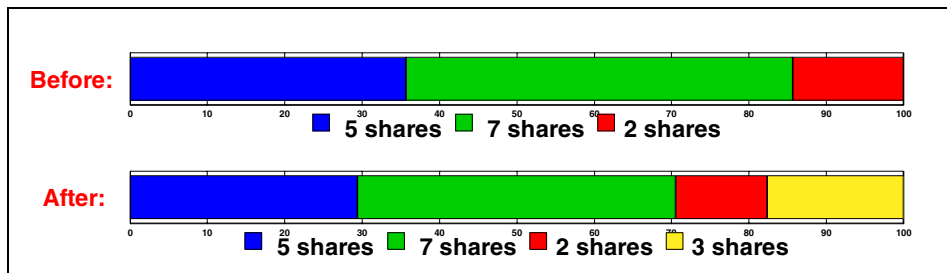


Figure 4. Example of share distribution automatically adjusting resources

### 2.7.3 Class resource limits

The class resource limits define the minimum and maximum amount of a resource that may be allocated to a class as a percentage of the total system resources. The different resources can be limited by the following values:

- The minimum percentage of the resource that must be made available when requested. The possible values are integers from 0 to 100. If unspecified, the default value is 0.
- The maximum percentage of a resource that can be made available when there is contention for the resource. If the contention no longer exists, this maximum limit can be surpassed. This is called a *soft* maximum because it is possible for a class to get more resource than this soft maximum value

if there is no contention. The possible values are integers from 1 to 100. If unspecified, the default value is 100.

- The maximum percentage of a resource that can be made available *even* if there is no contention for the resource. This is called a *hard* maximum. A class will never get more resource than its hard maximum limit, even if it is the only one active on the system. The possible values are integers from 1 to 100. If unspecified, the default value is 100.

WLM does not impose hard constraints on the values of the resource limits. The following are the only constraints:

- The minimum limit must be less than or equal to the soft maximum limit.
- The soft maximum limit must be less than or equal to the hard maximum limit.
- The sum of the minimum of all the superclasses within a tier cannot exceed 100.
- The sum of the minimum of all the subclasses of a given superclass within a tier cannot exceed 100.
- WLM will not let users set a hard memory limit on the *System* class because of potential deadlock situations.

For instance, consider the case in which performing file system I/Os involves a system daemon (a good example is NFS). If there is a hard maximum limit on the System class and the class reaches its maximum limit, the VMM page replacement algorithm (LRU) will be started and will initiate page-outs. No page will be given to processes in the System class until those page-outs complete, thus bringing the System class below its maximum limit. Because there is intensive stealing of the pages belonging to the processes in the System class due to the maximum limit, it is entirely possible that the file system daemon needs a page to start processing the I/Os. So, VMM will not give it a page until the I/Os are complete, and the daemon will not process any I/O until it gets its page(s). From then on, no System process will ever be given a memory page, and the system will halt in a matter of seconds.

When a class (other than System) has reached its hard memory limit and requires more pages, the VMM page replacement algorithm (LRU) is initiated to steal pages from the class at limit, lowering its number of pages below the hard maximum before handing out new pages (the class pages against itself). This is, of course, the desired behavior, but this extra paging activity, which can take place even where there are plenty of free memory pages available, will impact the general performance of the system. Memory minimums for



other classes should be used before imposing a memory hard maximum for any class.

This constraint about the sum of the minimum limits within a tier being less than or equal to 100 means that a class in the highest priority tier is always allowed to get resources up to its minimum limit. However, WLM cannot guarantee that the class will actually reach its minimum limit. This depends on how the processes in the class use their resources and on other limits that may be in effect. For example, a class may not be able to reach its minimum CPU entitlement because it cannot get enough memory.

For physical memory, setting a minimum limit gives some protection to the pages of the processes in the class (again, at least for the highest priority tier). Pages should not be stolen from a class below its minimum limit unless all the active classes are below their minimum limit and one of them requests more pages.

With this constraint, it means that pages should never be stolen from a class in the highest tier below its minimum limit. Therefore, setting a memory minimum limit for a class of interactive jobs helps ensure that their pages will not all have been stolen between consecutive activations (even in cases where the memory is tight) and improves response time.

In WLM with AIX Version 4.3.3, the maximum limit for the CPU resource is a soft maximum, and the maximum for the memory resource is a hard maximum. With AIX Version 4.3.3 maintenance level 8, it is possible to set hard maximum limits for the CPU resource as a global option with the `wlmset` command. The `wlmset` command is described in more detail in Section “AIX Version 4.3.3 maintenance level 8 `wlmset` command” on page 81.

**Note**

Resource limits take precedence over class resource share values.

#### **2.7.4 Backward compatibility considerations**

As mentioned earlier, in the first release of WLM the system default for the resource shares was one share. In AIX 5L it is (-), which means that the resource consumption of the class for this particular resource is not regulated by WLM. This changes the semantics quite a bit, and it is advised that system administrators review their existing configurations and consider if the new default is good for their classes or if they would be better off either setting up a default of one share to go back to the previous behavior or setting explicit values for some of the classes.

For the limits, the first release of WLM only had one maximum, not two. This maximum limit was, in fact, a *soft* limit for CPU and a *hard* limit for memory. Limits specified with the old format, *min percent-max percent*, will have, in AIX 5L, the max interpreted as a softmax for CPU, and a max that was set for memory will become both hardmax and softmax for memory in AIX 5L (which will give hardmax and softmax an equal value in this case). All interfaces (SMIT, AIX commands, and WSM) will convert all existing data from its old format to the new one.

The disk I/O resource is new for the current version, so, when activating AIX 5L's WLM with configuration files of the first WLM release, the values for the shares and the limits will be the default ones for this resource. The system defaults are as follows:

- shares = -
- min = 0 percent, softmax = 100 percent, hardmax = 100 percent.

Therefore, for existing WLM configurations the disk I/O resource will not be regulated by WLM, which should lead to the same behavior for the class as with the first version.

---

## 2.8 WLM interaction with the kernel

WLM's management of system resources interacts with the AIX kernel control mechanisms. These mechanisms are the scheduler for the CPU, the Virtual Memory Manager for memory, and device driver calls for the disk I/O bandwidth. They all use the allocation priority value calculated by WLM for each resource in each WLM class. This value is called Uniform Resource Access Priority (URAP).

### 2.8.1 Resource usage statistics

Once per second WLM gathers resource usage statistics per class for each resource type.

#### 2.8.1.1 CPU

When WLM is active, the dispatcher keeps track of the actual execution time of each thread using microsecond timing. WLM monitors the sum of the CPU time consumed by all the threads in the class, adds all the per class CPU counters, and then calculates the percentage of the total CPU time used by the threads in the class within the last second. This gives a time-decayed average of this per class CPU utilization, which is used for regulation.

### 2.8.1.2 Memory

WLM keeps track of the number of memory pages used by each class. The WLM accounting is done by segment. If a segment is accessed by only one process or by processes in the same class, the segment's pages are added to the counter for the process or processes the class belongs to. If the segment is accessed by processes belonging to different classes, the segment's pages are added to the counter of one of the Shared classes (Shared superclass or Shared subclass of a superclass), unless the `localshm` attribute is set. WLM calculates the percentage of real memory used by each class.

### 2.8.1.3 Disk I/O

WLM keeps track of the number of 512 blocks of data read/written by each class on each disk device accessed by that class. WLM collects the statistics given by the device drivers (through `iostat` and `dkstat`), which contain the percentage of time the device was busy, and uses this information together with the number of blocks read/written on each device by each class to calculate the percentage of the total bandwidth of each device. WLM then calculates a time-decayed average which is used for regulation.

## 2.8.2 Uniform Resource Access Priority (URAP)

Once a second, after calculating the percent usage of each resource per class, WLM compares these values to the class' target and limits, and calculates for each resource a Universal Resource Access Priority (URAP). The URAP is then used to favor the classes that do not use all the resources they are entitled to, and restrict those that tend to use too much compared to their entitlements.

- The URAP is a resource access priority. It is a positive number. Like AIX priorities, lower value means higher priority. The scheduling priority of the threads managed by WLM is between `P_NICE_DEFAULT` (60) and `P_IDLE` (255). Priority 255 is only used to "freeze" the threads in classes above their hard CPU maximum limit.
- It is calculated for each class, taking into account the superclass and subclass tiers, and the class resource usage.
- WLM reserves a priority range for each superclass tier and then a sub-range for each subclass tier. The subclasses in a tier are then given priorities within their range according to their resource usage, (Figure 5 on page 38).
- WLM fixes a maximum range for the URAP for each resource type.

### 2.8.2.1 Tier regulation

The total priority range for the URAP value of a given resource is limited. With 10 possible superclass tiers and 10 subclass tiers, the priority range for a tier could be very limited (total range/100). In most cases only a very limited number of tiers will be used in order to maximize the priority range per tier and get a better regulation, so WLM calculates effective tier numbers for each superclass and subclass, and keeps track of the number of superclass tiers in the system and the number of subclass tiers for each superclass (updated every time a class is added or deleted). See Figure 5.

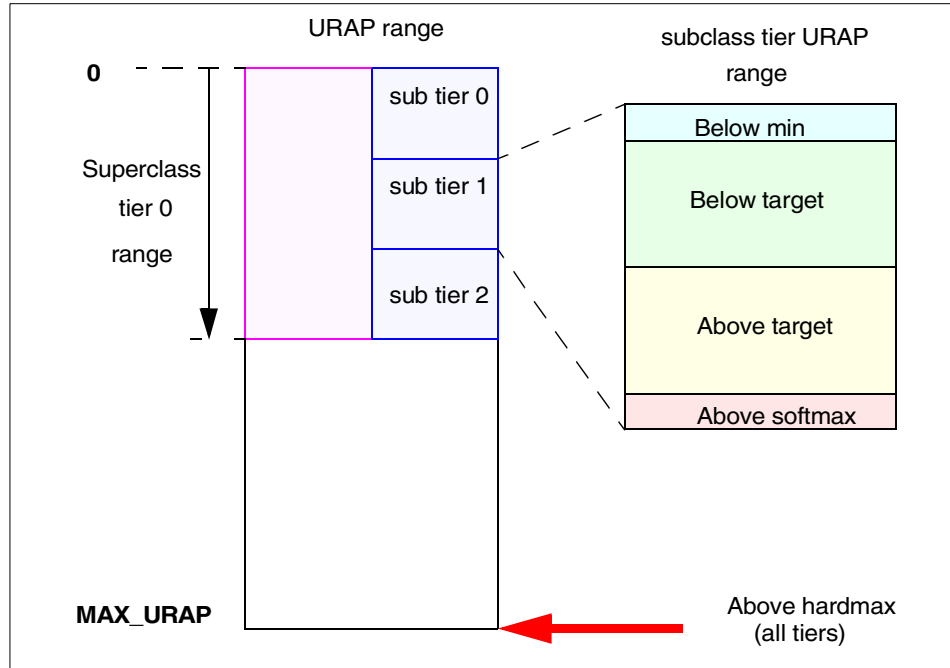


Figure 5. URAP

### 2.8.3 Interaction with the scheduler

AIX scheduler calls a WLM routine to inquire about the scheduling priority of each thread. This priority is determined by WLM using the URAP algorithm, which calculates the CPU allocation priority for the class of the thread in question.

In a WLM environment, the `nice` command will cause a process to have its CPU usage selectively favored or penalized with respect to other processes in the same class. The `nice` command will not affect the CPU utilization of

processes in other classes because WLM will work to have the class' resources meet the requested number of resource shares and resource limits.

The `schedtune` command can be used to modify the behavior of the scheduler. All options to `schedtune` continue to work in a WLM environment. The use of `schedtune` options will not significantly impact the ability of WLM to manage CPU usage.

**Note**

It is recommended that any tuning with `schedtune` be done prior to using WLM.

#### 2.8.4 Interaction with VMM

WLM controls the memory used by each class according to the tier where the class resides and the minimum, share, and maximum thresholds defined for the class. Regulation is based on memory URAPs computed from class consumption rates each second by `wlmsched`. This value is then used by VMM to control memory allocation to threads.

The `vmtune` command can be used to modify the behavior of VMM. All `vmtune` options work in a WLM environment. Some of the options to `vmtune`, particularly `minperm`, `maxperm`, `minfree`, and `maxfree`, can hamper WLM's ability to achieve the specified physical memory usage goals.

**Note**

It is recommended that any tuning with `vmtune` be done prior to using WLM.

#### 2.8.5 Interaction with disk device drivers

WLM intercepts the call to `devstrat` and executes its own algorithm for the regulation of disk I/O bandwidth. If the class needs to be restricted (over target, for example), WLM delays the I/O. The delay is adjusted to regulate the I/O throughput utilization of the class (on a per-device basis). The algorithm is based on the allocation priority value calculated for disk I/O bandwidth for the class of the thread being controlled.

---

## 2.9 WLM Application Programming Interface

The WLM Application Programming Interface (API) supplies applications with the ability to perform every task a system administrator does through WLM commands. The API is described in Chapter 6, “WLM Application Programming Interface (API)” on page 195.

---

## 2.10 Additional characteristics

The following points depict some additional characteristics of WLM:

### ***Overhead***

The activation of WLM on a system executing AIX 5L does not necessarily represent an increased system load. Nevertheless, one could expect some overhead on heavy loaded systems, which may increase based on the number and complexity of the rules configured in WLM.

### ***Passive mode***

When configuring WLM, know your users and applications. It is important to understand the user base and their computing needs. It is also important to have an understanding of the resources required by all applications in the system. This is where the WLM passive mode can help.

The passive mode provides a way to monitor the impact WLM brings to the system. By comparing system behavior between active and passive modes, the system administrator can easily redefine WLM configuration strategies.

### ***Monitoring***

AIX and WLM have different approaches for gathering CPU usage statistics and performance reports generated with common AIX tools, such as sar, vmstat, and ps, that might produce slightly different results from statistics gathered with wlmstat.

The common AIX performance monitoring tools use a sampling approach. Every clock tick (10 ms), AIX charges one CPU tick to the current process. It also determines the mode of execution (user/system) and increments the user or system time accordingly.

WLM, on the other hand, uses microsecond timing and records precisely how long a thread has been active. However, it does record user/system breakouts and accumulates the times on a per class basis only (it does not save individual thread CPU usage information).

In some situations, this can create differences between the statistics shown by `wlmstat` and `sar`, for example.

Another difference is that tools such as `sar` report “raw” values, while `wlmstat` reports a decayed average. For example, on a system running a CPU bound load (all processes in one class) `wlmstat 1` and `sar 1` will both show 100 (99) percent busy. If the load suddenly terminates, `sar` shows almost instantaneously a 0 percent busy, while `wlmstat` shows the CPU usage slowly going down to 0 (over 5 - 10 seconds). It is therefore recommended to run the performance monitoring tools at time intervals of 5 seconds or more when monitoring WLM workloads.

### ***Dynamic update***

Tiers, resource soft and hard limits, resource shares, rules, and every sort of WLM configuration can be modified while WLM is running and take immediate effect without the need to stop and restart WLM.

### ***Dump analysis***

The `snap` command is used to collect system information and dump files for problem determination. In AIX 5L, this command has a `-w` option, which is used to gather WLM information and join it to the already-existing database (basically, assembles the contents of the `/etc/wlm` directory). This feature can be very useful when analyzing an unknown technical problem that might or might not be WLM-related.

The substitute for the kernel-debugging `crash` command in AIX 5L, the `kdb` command incorporates options to analyze the behavior of WLM configuration at the kernel level when, for instance, a dump occurs.





## Chapter 3. AIX Workload Manager administration

WLM can be administered using three different methods:

- Web-based System Manager (WSM) graphical user interface, initiated with the AIX command `wsm` (Figure 6)
- System Management Interface Tool (SMIT), initiated with the AIX command `smit wlm` or `smitty wlm`
- Command line and file editing

Throughout this chapter, you will find descriptive examples of each of these methods' functionality.

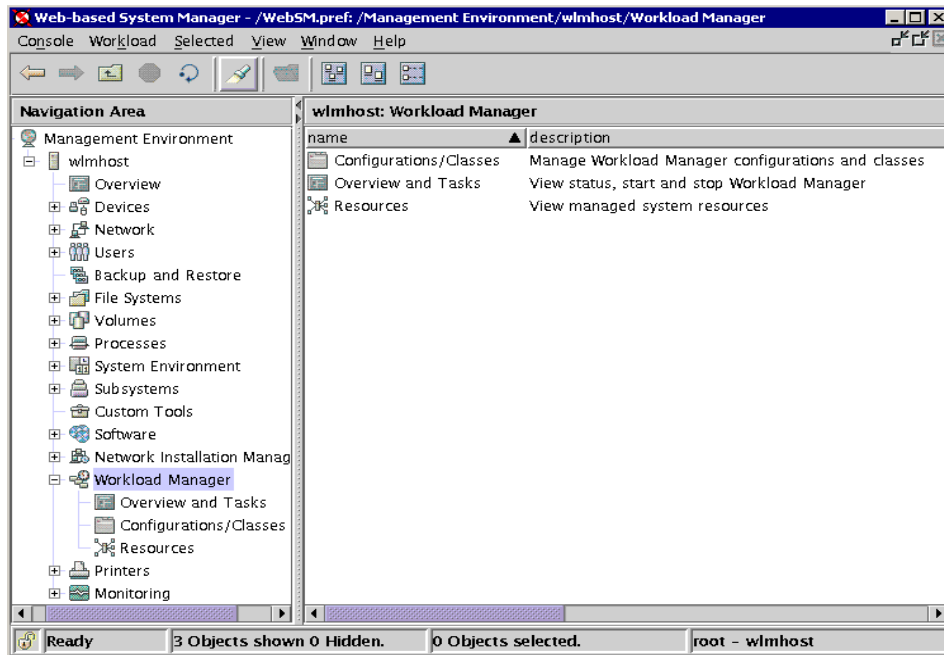


Figure 6. WLM screen in WSM

WLM commands can also be initiated through crontab entries to take advantage of WLM's various configuration capabilities. This way, job rankings can be changed at specific times of day and/or days of the week.

### 3.1 Property files

The WSM and SMIT interfaces record the configuration information in the same flat text files. These files are called the WLM *property files*, and reflect WLM's two-layered class configuration. The various WLM configurations are placed in subdirectories of /etc/wlm. A symbolic link, /etc/wlm/current, points to the directory containing the current configuration files. For example, the current running rules file is stored in a file, /etc/wlm/current/rules. This link is updated by the `wlmcntrl` command when WLM starts with a specified set of configuration files. The sample configuration files shipped with AIX are in the /etc/wlm/standard directory.

The example in Figure 7 shows a configuration, called *Config*, which is, therefore, placed in /etc/wlm/Config.

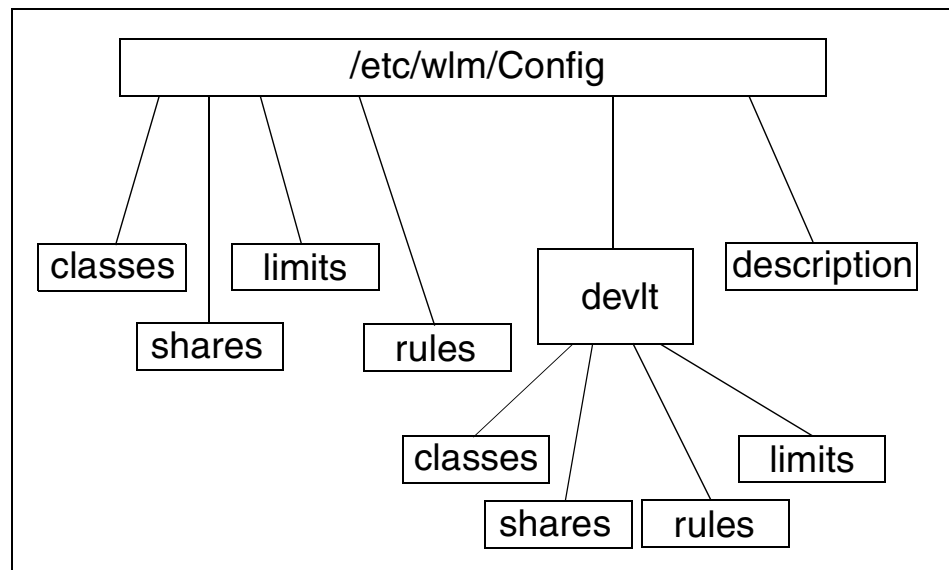


Figure 7. WLM's property files

The various files are explained below:

- *description* contains the description of the configuration.
- *classes* contains the class definitions of the configuration.
- *shares* contains the resource entitlements of the configuration.
- *limits* contains the resource limits of the configuration.
- *rules* contains the assignment rules of the configuration.

The configuration, *Config*, has a superclass, named *devlt*. Each superclass is represented by a subdirectory in the configuration directory named after the superclass. For each superclass, this subdirectory contains the *classes*, *shares*, *limits*, and *rules* files corresponding to the superclass's subclasses, resource entitlements, limits, and assignment rules.

The WLM property files for a WLM configuration must have write permission only for root. The WLM property files for superclasses must have write permission for the adminuser and admingroup for the superclass. If there is no adminuser for the superclass, the files should be owned by root. If there is no admingroup for a superclass, the WLM property files for the superclass should be the group *system*, and have no write permission for the group.

### ***classes file***

This file contains the definition of WLM superclasses or subclasses for a given configuration. This file is organized into stanza names, which are WLM class names, and contents, which are attribute-value pairs specifying characteristics of the class. Each stanza names a WLM class. The only names that have a special meaning to the system are Default, Shared, Unclassified, Unmanaged, and System. Unclassified and Unmanaged cannot appear as class names in this file. The superclasses, Default, Shared, and System, are always defined. The subclasses Default and Shared are always defined. The class attributes that can be defined in the *classes* file are tier, inheritance, adminuser, admingroup, authuser, authgroup, resource set (rset), and localshm. Refer to Section 2.4, "Class attributes" on page 17, for further details about these attributes. The attributes that have not been explicitly set by the system administrator are omitted from this file. The default values for these attributes can be changed using a special *default* stanza at the very top of this file. Be extra careful when using this default stanza because it can lead to starvation of your *System* superclass.

The following is part of a typical */etc/wlm/Config/classes* file for the example in Table 2 on page 26. In this example, the tier for db2 would have to be set to 1 because the default value, specified for the tier attribute in the special *default* stanza at the top of this file, has been set to tier 0:

```

* This is the default special stanza, valid for all classes
* if not specified otherwise:
*
default:
    tier=0

* System defined classes
* All attributes to default value
* Attribute values can be specified

System:
Default:
Shared:

* User defined classes

db1:
    inheritance = "yes"
    localshm = "yes"
    authgroup = "devlt"
    adminuser = "bob"

db2:
    tier = 1
    authuser = "sally"
    admingroup = "sales"

devlt:
    authgroup = "dev"
    rset = ""

```

#### Note

The asterisk (\*) is the comment character used in the classes file. The example shows comments in the classes file for clarity only. Comments can be added by directly editing the file. However, users should be aware that all other interfaces to create/modify/delete classes (command line, SMIT, WSM) will remove the comments when updating the file.

#### **shares file**

This file contains the definition of the number of shares of all the resources allocated to superclasses or subclasses for a given configuration. This file is organized into stanza names, which are WLM class names, and contents,

which are attribute-value pairs specifying the number of shares allocated to the class for the various resources. The attribute names identify the resource. The shares value is either an integer between 1 and 65535 or a hyphen (-) to indicate that WLM does no regulation for the class for the given resource. This is the system default. Each stanza names a WLM class, which *must* exist in the classes file at the corresponding level (superclass or subclass). The class attributes defined in the *shares* file are CPU, memory, and disk I/O. Refer to Section 2.7.1, “Resources managed by WLM” on page 30 for further detail on these attributes. The values just mentioned as being the system default can be modified using a special stanza, called *default*, at the very top of the shares file. Be extra careful when using this default stanza because it can lead to starvation of your *System* superclass.

The following is part of a typical `/etc/wlm/Config/shares` file for the example in Table 2 on page 26. In this example, *System*, *Shared*, and *db2* would get four shares of CPU as specified by the *default* special stanza:

```

* This is the default special stanza, valid for all classes
* if not specified otherwise
*
default:
    CPU      = 4

* System Defined Classes
* In this example, the system administrator uses
* only default values for the System and Shared
* superclasses, which are omitted in the file.
* The system administrator gives non default values
* only for the Default class:

Default:
    CPU      = 2
    memory   = 10

*
* User defined classes
*

db1:
    CPU      = 2
    memory   = 1
    diskIO   = 6

db2:
    CPU      = -
    memory   = 12
    diskIO   = 6

devlt:
    CPU      = 30
    memory   = 1
    diskIO   = 1

VPs:

```

**Note**

The asterisk (\*) is the comment character used in the shares file.

The example shows comments in the shares file for clarity only. Comments can be added by directly editing the file. However, users should be aware that all other interfaces to create/modify/delete shares (command line, SMIT, WSM) will remove the comments when updating the file.

### **limits file**

Contains the specification of the minimum and maximum limits for the resources allocated to superclasses or subclasses of a given configuration. Although the limits at the superclass level represent a percentage of the total amount of resource available on the system, and the limits at the subclass level represent a percentage of the target usage configured for the superclass, the description of resource limits for the superclasses and subclasses have the same format. This file is organized into stanza names, which are WLM class names, and contents, which are attribute-value pairs specifying the minimum and maximum resource allocated to the class for the various resources. The attribute names identify the resource. For each resource, three values must be provided: The minimum limit (m), a soft maximum limit (SM), and a hard maximum limit (HM). Refer to Section 2.7.3, “Class resource limits” on page 33, for further details about these values. The limits are expressed as percentages. The minimum limit is a number between 0 and 100, and the maximum limits are numbers between 1 and 100. The hard maximum must be greater than or equal to the soft maximum, which must be greater than or equal to the minimum. The system default values, when the limits are not specified for a class or a resource type, are 0 for the minimum and 100 for both the soft and hard maximum.

The syntax is:

```
attribute = m%-SM%;HM%
```

Each stanza names a WLM class, which *must* exist in the classes file at the corresponding level (superclass or subclass). The class attributes defined in the *limits* file are CPU, memory, and disk I/O. Refer to Section 2.7.1, “Resources managed by WLM” on page 30, for further details about these attributes. The values mentioned above as being the system default can be modified using a special stanza, called *default*, at the very top of the limits file. Be extra careful when using this default stanza because it can lead to starvation of your *System* superclass.

The following is part of a typical `/etc/wlm/Config/limits` file for the example in Table 2 on page 26. In this example, *db2* and *Default* would be assigned a minimum of zero percent, a soft maximum of 50 percent, and a hard maximum of 70 percent for CPU resource because of the special *default* stanza:

```

* This is the default special stanza, valid for all classes
* if not specified otherwise:
default:
    CPU = 0%-50%;70%
* System Defined Classes
* In this example, the system administrator uses
* default values for the Shared
* superclass (memory only) .
* The system administrator gives non default values
* for the Default and System classes. The System class
* has a memory minimum of 1% by default. This value
* can be increased by system administrator:
System:
    memory = 1%-100%;100%
Default:
    memory = 0%-50%;100%
devlt:
    memory = 10%-70%;80%
    diskIO = 10%-80%;100%

```

#### Note

The asterisk (\*) is the comment character used in the limits file.

The example shows comments in the limits file for clarity only. Comments can be added by directly editing the file. However, users should be aware that all other interfaces to create/modify/delete limits (command line, SMIT, WSM) will remove the comments when updating the file.

#### Note

If the maximum limit configured for a class exceeds the system wide limit in /etc/secure/limits, processes in the class will be killed if they reach the system wide limit regardless of the WLM configured limit.

#### **rules file**

This file defines the automatic class assignment rules for the superclasses or subclasses of a given configuration. Each line of this file represents an assignment rule for a given class. There can be several assignment rules for the same class. Each rule lists the name of a class and a list of values for some attributes of a process that are used as classification criteria. The various fields of a rule are separated by blank spaces. Attributes whose values are not specified will be represented by a hyphen (-). The fields of an assignment rule, listed in the order in which they *must* appear in the *rules* file, are class name, reserved, user, group, application, type, and application tag.



Refer to Section 2.5.3, “Class assignment rules” on page 23 for further detail on these attributes. The class name and the first two attribute fields (reserved and user) are mandatory. The other fields, if not present, will default to (-). Remember, however, that WLM recognizes the fields by their position on the line. It is, therefore, not possible to omit one field in the middle of the line. For example, if you skip a group name and enter an application name, WLM will take the application name as a group name and give error messages about invalid groups.

WLM will scan this file from top to bottom, looking for the first rule that is a match for the set of process attributes (user, group, application, type, and tag) for each application:

- If the value in the rule is a hyphen (-), any value of the corresponding process attribute is a match.
- If the value of a process attribute other than *type* appears in the list of values specified in the corresponding field in the rule and is not preceded by the exclusion character (!), this is a match for the specified attribute.
- If the values of the process *type* attribute (32bit/64bit, plock, fixed) match all the values (separated by (+) signs) provided in the list of one or more comma-separated values for the *type* field in the rule, this is a match for the process type. For example, for *32bit,plock+fixed*, it must match either *32bit* or *plock and fixed*.
- The process will be classified in the class specified in the *class* field of the rule if all the values of the process attributes in the table above match the values in the corresponding field of the rule.

When classifying a process, WLM will first scan the rules file for the superclasses of the current configuration to determine which superclass the process will be assigned to, and then WLM scans the rules file for this specific superclass to determine which subclass of the superclass the process will be assigned to.

There are implicit rules for the Default superclass and the Default subclass of all superclasses (whether or not they are present in the *rules* files), which will classify all processes that did not match any of the other rules.

The following is an example of a `/etc/wlm/Config/rules` file for the configuration given in the example in Table 2 on page 26:

```

* This file contains the rules used by WLM to
* assign a process to a superclass
*
* classresvdusergroupapplicationtypetag
System-root- - - -
db1- - - /usr/oracle/bin/db*_db1
db2- - - /usr/oracle/bin/db*_db2
devlt- - dev- 32bit-
VPs- bob,sally--- -

```

The following is an example of the rules file for the *devlt* superclass in */etc/wlm/Config/devlt/rules* of the previous example:

```

* This file contains the rules used by WLM to
* assign a process to a subclass of the
* superclass devlt
*
* classresvdusergroupapplicationtypetag
hackers-jim,liz--- -
hogs- - - - 32bit+plock-
editors-!sue-/bin/vi,/bin/emacs--
build- - - /bin/make,/bin/cc--
Default--- - - - -

```

#### Note

The asterisk (\*) is the comment character used in the rules file.

The example shows comments in the rules file for clarity only. Comments can be added by directly editing the file. However, users should be aware that all other interfaces to create/modify/delete rules (command line, SMIT, WSM) will remove the comments when updating the file.

In the */etc/wlm/.running* directory, the system administrator can find an image of the currently-running configuration in the kernel.

The class definitions, shares, and limits in effect at a given time (that is, known to the kernel at this time) may be different from the class definitions, shares, and limits in the *current* configuration (the set of files in the directory pointed to by */etc/wlm/current*) for several reasons:

1. The configuration files could have been modified, but WLM has not been refreshed yet.
2. Classes have been created and/or shares and limits were changed directly into the kernel (without updating the configuration files) either by

an application using the API, or the command line interface (by specifying an empty string as the configuration name (-d "").

This is why WLM keeps a set of configuration files in a special directory, /etc/wlm/.running, which at any given time reflects the class definitions, shares, limits, and rules exactly as they are known to the kernel.

---

## 3.2 WLM configuration

This section discusses some of the steps a system administrator needs to take to configure WLM on a system. First, it points out the method to follow to configure WLM in a manner that is easy to maintain and update. Afterwards, it shows how the configuration can be done in practice, using any of the three methods provided to configure WLM; command line, SMIT, and WSM.

### 3.2.1 Steps for a WLM configuration

In order to successfully configure WLM on a system, it is recommended that the system administrator follow a set of steps described in the following sections.

#### ***Step 1 - Design your classification***

The first step is to define your classes (superclasses first). In order to define which classes you need, you must know your users and their computing needs, the applications on your system and their resource needs, and the requirements of your business (that is, which tasks are critical and which can be given a lower priority). This depends a lot on what you'll be using WLM for. If this is a case of server consolidation, you probably already know the applications, their users, and their resource requirements, and you may be able to skip or shorten some of the steps.

WLM is very flexible, and allows you to classify processes by user/group, application (besides type and application tag), or any possible combination. Because WLM regulates the resource utilization among the classes, you should group in the same classes applications and/or users with the same resource utilization patterns. For instance, you generally want to separate the interactive jobs that typically consume very little CPU time but require quick response time when activated from batch type jobs that, typically, are very CPU- and memory-intensive. It is the same in a database environment where you probably need to separate the online transaction processing (OLTP) type traffic from the heavy queries of data mining, for example.

WLM cannot help much in this initial design phase. You will probably have to go through a few iterations to refine your classification and optimize your

class definitions. At the end of this step, you should be able to set up your class definitions and the corresponding assignment rules.

### ***Step 2 - Create the superclasses and assignment rules***

This step is done using the WLM administration interfaces, WSM, SMIT, or command line interface. In the next sections, the process of configuring WLM using these tools will be covered. The first few times, it is probably a good idea to use WSM or SMIT. They will take you through the steps of creating your first WLM configuration including defining the superclasses and setting their attributes. For the first pass, you can set up only some of the attributes and leave the others at their default value. The same thing is applicable for the resource shares and limits. All these characteristics of the classes can be dynamically modified later on. The goal is to have a basic set of superclasses and the associated assignment rules defined. When that is done, you can start WLM in passive mode, check your classification, and start looking at the resource utilization patterns of your applications.

### ***Step 3 - Use WLM to refine your class definitions***

When Step 2 above is complete, you can check your configuration using the `wlmcheck` command or the corresponding SMIT or WSM menus, and start WLM in passive mode on the newly-defined configuration. This means that WLM will classify all the existing processes (and all processes created from then on) and start getting statistics on the CPU, memory, and disk I/O utilization of the various classes, but will not try to regulate this resource usage. This is, basically, what needs to be accomplished at that point; check that the various processes are classified in the right class as expected by the system administrator (using the `-o class` option of the `ps` command). If some of the processes are not classified as you expect, you can modify your assignment rules and/or set the inheritance bit for some of the classes (if you want the new processes to remain in the same class as their parent) and update WLM. You can repeat the process until you are satisfied with this first level of classification (superclasses).

Running WLM in passive mode and refreshing WLM (always in passive mode) is a very low-risk, low-overhead operation, and can be done safely on a production system without disturbing normal system operation.

### ***Step 4 - Gather resource utilization data***

For this purpose, WLM should be run in passive mode (using the class definitions resulting from Step 3) and gather statistics using the `wlmstat` command. This command can be started to display the per class resource utilization (as a percentage of the total resource available for superclasses) repeatedly and at regular time intervals. You can thus monitor your system for

extended periods of time to look at the resource utilization of your main applications over time.

With this data and your business goals defined in Step 1 (which applications and/or system users are critical for your business and which are less important), you can start deciding (or refining) which tier number will be given to every superclass and what share of each resource should be given to the various classes.

#### ***Step 5 - Turn WLM on***

You are now ready to start WLM in active mode and monitor the system again with the `wlmstat` command to check if the regulation done by WLM is in line with your goals and if applications are not unduly deprived of resources while others get more than they should. If this is the case, adjust the shares and refresh WLM.

For some specific cases, you may have to use minimum and/or maximum limits. If possible, try to adjust the shares (and potentially tier numbers) to get closer to your resource allocation goals first and reserve limits for cases that cannot be solved with shares only. Use minimum limits for applications that typically have low resource usage but need a quick response time when activated by an external event. One of the problems faced by interactive jobs in situations where memory becomes tight is that their pages get stolen during the periods of inactivity (waiting for user input, for instance). A memory minimum limit can be used to protect some of the pages of interactive jobs (up to the minimum limit) if the class is in tier 0. Use maximum limits to contain some resource-hungry, low-priority jobs. Again, unless you want to partition your system resources for other reasons, a hard maximum will make sense mostly for a non-renewable resource, such as memory, because of the time it would take to write data out to the paging space if a higher priority class would suddenly need pages that this other class would have used. For CPU, you can use tiers or soft maximum to make sure that if a higher priority class needs the CPU, it gets it right away. Again, monitor and adjust the shares, limits, and tier numbers until you are satisfied with the system's behavior.

#### ***Step 6 - Fine tune your configurations***

In this step, you can decide whether you need to use subclasses and, if you do, whether you want to delegate the subclasses administration for some or all of the superclasses. When creating and adjusting the parameters of subclasses, you can refresh WLM only for the subclasses of a given superclass without affecting users and applications in the other superclasses. The administrator of each superclass can repeat the same process described above (Steps 1 through 5) for the subclasses of the superclass. The only

difference is that it is not possible to run WLM in passive mode at the subclass level only. The subclass configuration and tuning might have to be done with WLM in active mode. In this case, one way of not impacting users and applications in the superclass is to start with the tier number, the shares and limits for the subclasses at their default value ((-) for shares, 0 percent for min, and 100 percent for soft and hard max) so that WLM will not regulate the resource allocation between the subclasses. The administrator can then monitor and set up the subclasses shares, limits, and tier number as explained in the steps above.

#### **Step 7 - Create other configurations as needed**

When you are done with your initial configuration, you can repeat the process to define other configurations with different parameters for nights and weekends, for instance, according to the needs of the business. When doing so, you can probably take shortcuts for some steps because you will be modifying existing configurations.

### **3.2.2 Working with WLM configurations**

WLM allows the setup of various configurations. They can be used interchangeably, for instance, manually or by configuring `crontab` to change WLM into a particular configuration at a specific point of time (night time or weekends, for example).

Let us consider an example; a system runs an interactive job that is heavily used during daytime, a batch calculation job that must not interfere with the previous one, and a backup that must not interrupt or steal resources from any of the jobs mentioned previously. Nevertheless, all these jobs are to perform their tasks eventually. So, the system administrator might want to make sure the calculation job runs every night from midnight to 3:00 a.m., and the backup is done from 4:00 to 6:00 a.m. One way to set up all this is:

- Create a configuration, *daytime*, with classes for these jobs; *interactive* in tier 0, *batch* in tier 1, and *backup* in tier 2.
- Create a second configuration, *nighttime1*, with *batch* in tier 0, *interactive* in tier 1, and *backup* in tier 2.
- Create a third configuration *nighttime2* with *backup* in tier 0, *batch* in tier 1, and *interactive* in tier 2.
- Set up `crontab` to change WLM from *daytime* to *nighttime1* at midnight and from *nighttime1* to *nighttime2* at 4:00 a.m.

This is only one of the ways this setup can be implemented. The idea here is to illustrate the use of WLM's various configurations capability.

### Command line

Using the command line, the way to create configurations in WLM is simply to create new directories under `/etc/wlm`, to copy the contents from one configuration to the new one (if this is the first configuration, use `/etc/wlm/templates`) and edit the files manually. The name of the configurations will be the subdirectory names under `/etc/wlm`.

So, if in our example if you already had the configuration, *daytime*, which could have been created by setting up the subdirectory, `/etc/wlm/daytime`, and creating the classes in it, you could now copy this configuration into newly-created `/etc/wlm/nighttime1` and `/etc/wlm/nighttime2` subdirectories and edit the files manually to alter the tier attribute of the classes. The `mkclass` command could be used to set up new classes (see also “Adding a class - `mkclass`” on page 62), and the `lsclass` command could be used to list the contents of our new configurations (see also “Listing the classes - `lsclass`” on page 64).

### SMIT

To set up our example in SMIT, you could access the *Work on alternate configurations* screen, shown in Figure 8, or you could use the following fastpath:

```
# smitty wlmconfig
```

```
Work on alternate configurations
Move cursor to desired item and press Enter.
Show all configurations
Copy a configuration
Create a configuration
Select a configuration
Enter configuration description
Remove a configuration

F1=Help          F2=Refresh      F3=Cancel      Esc+8=Image
Esc+9=Shell     Esc+0=Exit     Enter=Do
```

Figure 8. `smitty wlmconfig`

From this screen, the system administrator can:

- See all the existing configurations (*Show all configurations*).
- Copy an existing configuration into a new one (*Copy a configuration*).
- Create a brand new configuration (*Create a configuration*).
- Select a configuration to work with (*Select a configuration*). Also see Figure 9. The output of this option is the listing of the superclasses and subclasses of the selected configuration. All changes made from this option on (create, change, delete classes, or rules) will apply to the selected configuration, leaving the currently-running configuration unchanged. The scope is returned to the currently-running configuration if SMIT is exited and restarted.
- Enter a description for the configuration (*Enter configuration description*).
- Remove a configuration (*Remove a configuration*).

```

COMMAND STATUS
Command: OK          stdout: yes          stderr: no
Before command completion, additional instructions may appear below.
[TOP]
System
Default
Shared
db1
devlt
devlt.Default
devlt.Shared
devlt.hackers
devlt.hogs
devlt.editors
devlt.build
VPs
[MORE...3]
F1=Help          F2=Refresh      F3=Cancel      F6=Command
F8=Image        F9=Shell       F10=Exit      /=Find
n=Find Next

```

Figure 9. Select a configuration screen in SMIT

### WSM

Configurations can be managed in WSM from the *Configurations/Classes* screen. As you enter this screen and all existing configurations are listed, you can right-click on the configuration to be updated, and all the configuration options will be listed in a pop-up window (Figure 10 on page 59). Some of these options, namely, the ones related to WLM management, are described in Section 3.3.2, “Start/Stop/Update WLM - wlmctrl” on page 99. Only the options regarding WLM *configurations* are described in this section.



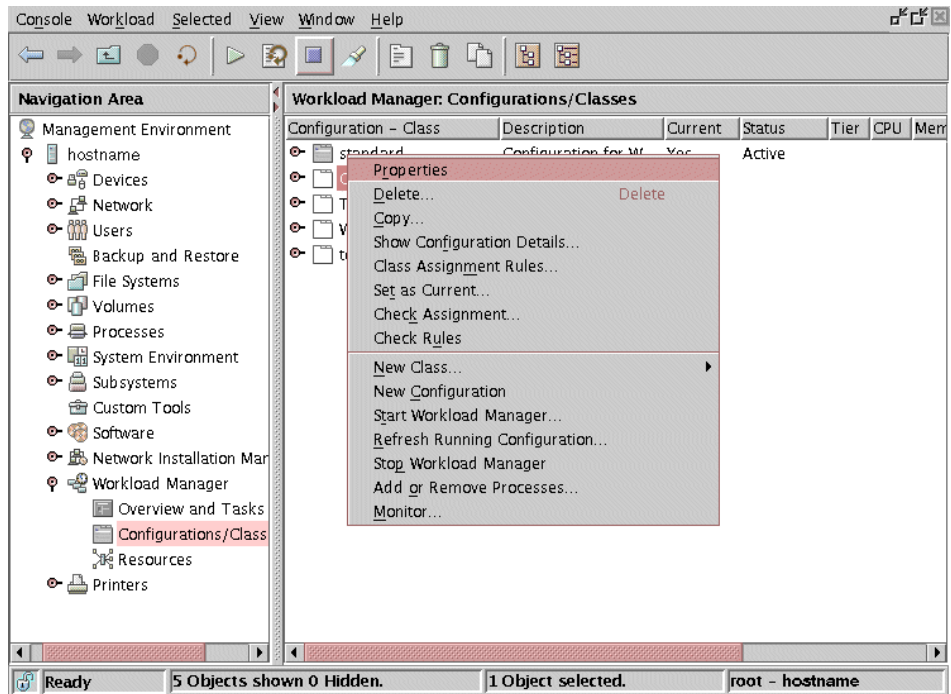


Figure 10. Configuration options

- The *Properties* option allows the system administrator to visualize general characteristics of the configuration (name, description, and whether it is the currently-running configuration or not) and change the description. Alternatively, the properties icon in the upper part of the WSM window can be clicked:



*Properties*

- The *Copy* option allows the system administrator to create a new configuration out of an already-existing one. Alternatively, the copy icon at the top of the WSM window can be clicked:



*Copy*

- *Show Configuration Details* shows general characteristics of the configuration, such as its classes and their shares and limits. In Figure 11 on page 60, you can see a possible output for the example in Table 2 on page 26.

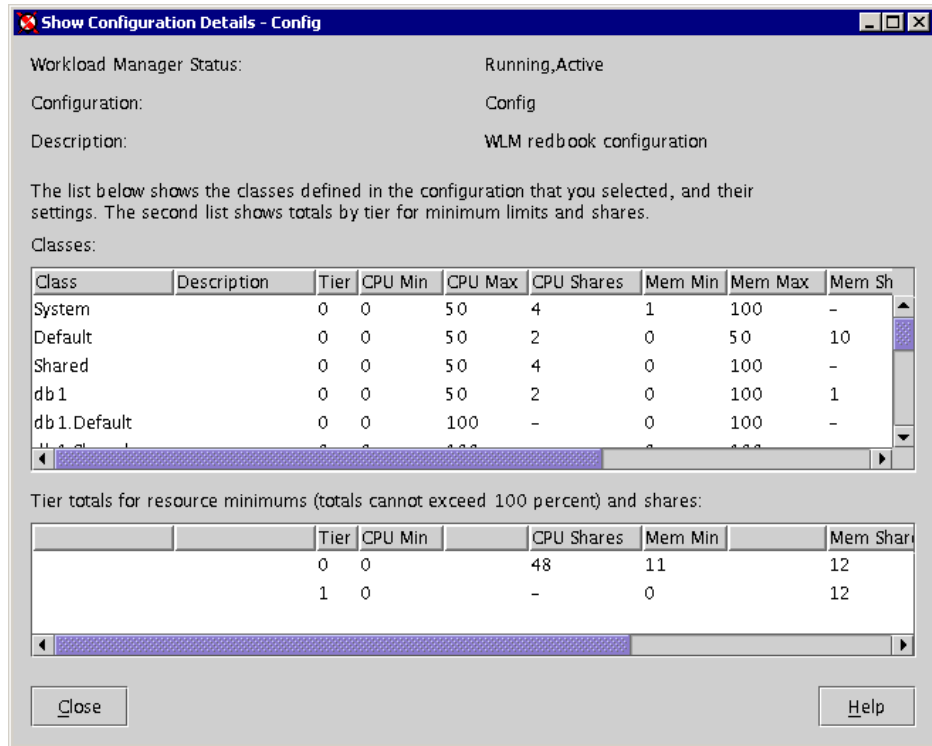


Figure 11. Show Configuration Details screen in WSM

- *New Configuration* allows the system administrator to set up a new configuration to work with.
- *Refresh Running Configuration* updates the configuration with the changes made. This screen is shown in Figure 12 on page 61.

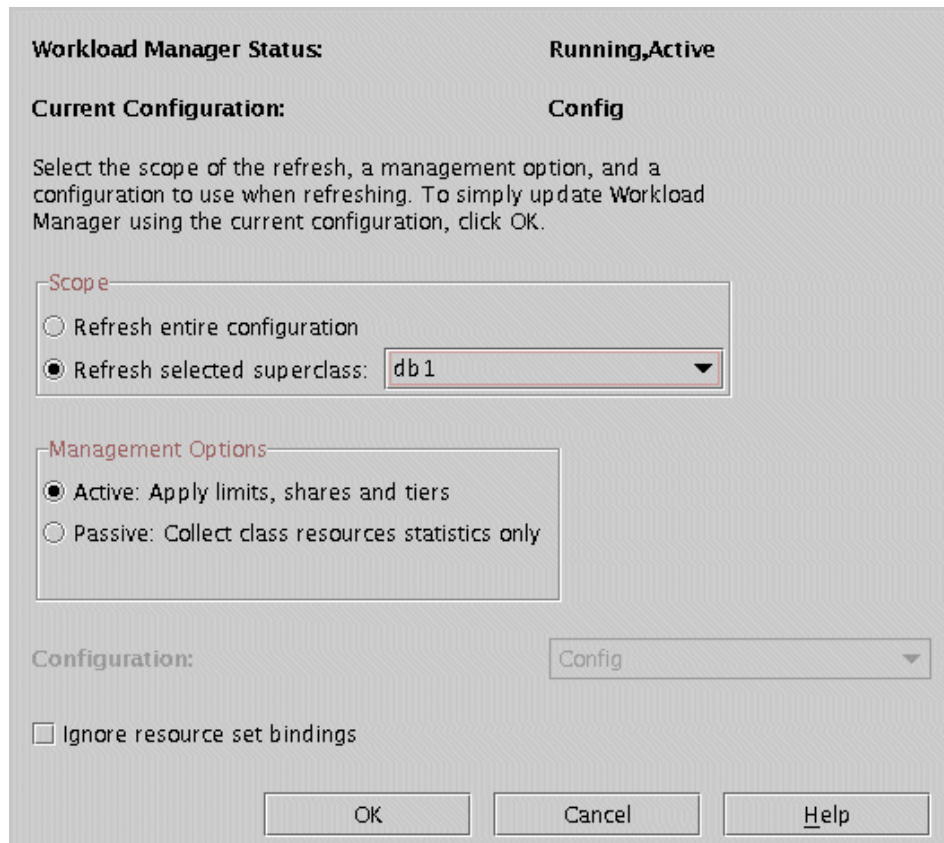


Figure 12. Refresh Current Configuration screen in WSM

Alternatively, one of the icons at the top of the WSM window can be used to perform this task:



To remove a configuration, click on the configuration to be deleted and then press the delete icon at the top of the window:



### 3.2.3 Working with classes

After defining the configuration name, superclasses must be added to it, and then subclasses can be configured. This section will show how the system administrator can deal with both superclasses and subclasses.

#### 3.2.3.1 Using the command line

This section explains the execution on the command line.

##### **Adding a class - *mkclass***

The command to create classes in WLM is `mkclass`. The syntax of this command is as follows:

```
mkclass [ -a Attribute=Value ... ] [ -c | -m | -b KeyWord=Value ] [ -d Config_Dir ] [ -S SuperClass ] Name
```

The `mkclass` command creates a superclass or a subclass identified by the `Name` parameter. The class must not already exist. The name parameter can contain only uppercase and lowercase letters, numbers, and underscores (\_). The name is in the format, *supername* or *subname* (with the `-S supername` option) or *supername.subname*. Supername and subname are each limited to 16 characters in length. The names Default, System, and Shared are reserved. They refer to predefined classes. Any `Attribute=Value` or `KeyWord=Value` argument will initialize the specified attribute or resource limit.

The options for this command are:

- `-a Attribute=Value` To set up an attribute value. The valid names for attributes are *tier*, *inheritance*, *authuser*, *authgroup*, *rset*, *adminuser*, *admingroup*, and *localshm*.
- `-b KeyWord=Value` Changes a limit or share value for disk I/O throughput. Possible KeyWords are *min*, *softmax*, *hardmax*, and *shares*.
- `-c KeyWord=Value` Changes a limit or share value for a CPU. Possible KeyWords are *min*, *softmax*, *hardmax*, and *shares*.
- `-d Config_dir` To use `/etc/wlm/Config_dir` as an alternate directory for the properties files. When this option is not used, `mkclass` uses the configuration files in the directory pointed to by `/etc/wlm/current`.
- `-m KeyWord=Value` Changes a limit or share value for memory. Possible KeyWords are *min*, *softmax*, *hardmax*, and *shares*.

- S Superclass      To specify the name of the superclass when creating a subclass. There are two ways of creating the subclass, *sub*, of the superclass, *Super*:
- Specify the full name of the subclass as *Super.Sub* for Name, and do not use *-s*.
  - Use the *-s* option to give the superclass name, and use the short name for the subclass: `mkclass <options> -S Super Sub`

So, to set up the *devlt* superclass and the subclass, *hackers*, from the example in Table 2 on page 26, the following commands could be run:

```
# mkclass -a inheritance=yes -a tier=0 -a adminuser=bob devlt
# mkclass -a inheritance=no -a tier=0 -S devlt hackers
```

or

```
# mkclass -a inheritance=no -a tier=0 devlt.hackers
```

### **Updating a class - *chclass***

The command to update a class is *chclass*. The syntax of this command is:

```
chclass -a Attribute=Value [[-a Attribute=Value]...] [-c|-m |-b
Keyword=Value] [-d Config_dir] [-S Superclass] Name
```

The *chclass* command changes attributes for the class identified by the Name parameter. The class must already exist. To change a class attribute (tier, inheritance, adminuser, admingroup, rset, authuser, authgroup, and localshm), specify the attribute name and the new value with the *-a Attribute=Value* option. To change/set a limit or shares value, use option *-c* for *cpu*, *-m* for *memory*, and *-b* for *disk I/O* (stands for *block I/O*), with the Keyword value in *min*, *softmax*, *hardmax*, or *shares*.

The options for this command are:

- a Attribute=Value      To change a class attribute (attribute in *tier*, *inheritance*, *adminuser*, *admingroup*, *rset*, *authuser*, *authgroup*, and *localshm*).
- c Keyword=Value      To change CPU resource limits or shares (keyword in *min*, *softmax*, *hardmax*, or *shares*).
- m Keyword=Value      To change memory resource limits or shares (keyword in *min*, *softmax*, *hardmax*, or *shares*).

- b Keyword=Value To change Disk I/O resource limits or shares (keyword in min, softmax, hardmax, or shares).
- d Config\_dir To use /etc/wlm/Config\_dir as an alternate directory for the properties files. If this option is not present, the current configuration files in the directory pointed to by /etc/wlm/current are used.
- S Superclass To specify the name of the superclass when changing the attributes of a subclass. There are two ways of specifying that the change is to be applied to the subclass, *Sub*, of the superclass, *Super*:
  - Specify the full name of the subclass as *Super.Sub* and not use *-s*.
  - Use the *-s* option to give the superclass name and use the short name for the subclass:  
`chclass <options> -S Super Sub`

So, to change the *devlt* class from the example in Table 2 on page 26, you could run the following command to give it 20 CPU shares, change the administration user to *bob*, and set 10 percent as the memory minimum limit:

```
# chclass -a adminuser=bob -c shares=20 -m min=10 devlt
```

### **Listing the classes - *lsclass***

The command to list classes is *lsclass*. The syntax of this command is:

```
lsclass [ -C |-D |-f ] [ -r ] [ -d Config_dir ] [ -S Superclass ] [ Class ]
```

With no arguments, *lsclass* simply lists all superclasses in the current configuration. This command is accessible to all users in the system.

The options for this command are:

- C To display the class attributes, shares, and limits in colon-separated records:

```
#lsclass -C devlt
#name:description:tier:inheritance:authuser:authgroup:adminuser:
admingroup:rset:CPUshares:CPUmin:CPUsoftmax:CPUhardmax:
memoryshares:memorymin:memorysoftmax:memoryhardmax:diskIOshares:
diskIOmin:diskIOsoftmax:diskIOhardmax:localshm
devlt::0:no::dev:::30:0:100:100:1:10:70:80:1:10:80:100:no
```

- D To display the default values for the class attributes, shares, and limits in colon-separated records:

```
#lsclass -D devlt
#name:description:tier:inheritance:authuser:authgroup:adminuser:
admingroup:rset:CPUshares:CPUmin:CPUsoftmax:CPUhardmax:
memoryshares:memorymin:memorysoftmax:memoryhardmax:diskIOshares:
diskIOmin:diskIOsoftmax:diskIOhardmax:localshm
::0:no::::-:0:100:100:-:0:100:100:-:0:100:100:no
```

**-f** To display the output in stanzas, with each stanza identified by a class name. Each Attribute=Value pair is listed on a separate line:

```
Class:
  attribute1=value
  attribute2=value
  attribute3=value
```

- r** To recursively display the superclasses with all their subclasses. When specifying **-r**:
- If no class name is given, `lsclass` will show all the superclasses with all their subclasses.
  - If the name of a superclass is given, `lsclass` displays the superclass with all its subclasses.
  - If the name of a subclass is given, **-r** is ineffective (displays only the subclass).
- d Config\_dir** To use `/etc/wlm/Config_dir` as alternate directory for the definition files. If this option is not present, the current configuration files in the directory pointed to by `/etc/wlm/current` are used.
- S Superclass** To restrict the scope of the command to the subclasses of the specified superclass. When **-s** is used, only subclasses are shown.

### **Removing a class - `rmclass`**

The command to remove classes is `rmclass`. The syntax of this command is:

```
rmclass [-d Config_dir] [-S Superclass] Name
```

The `rmclass` command removes the superclass or the subclass identified by the `Name` parameter from the class definition file, the class limits file, and the class shares file. The class must already exist. The predefined Default, System, Shared, Unmanaged, and Unclassified classes cannot be removed. In addition, when removing a superclass, *Super*, the directory, `/etc/wlm/Config_dir/Super`, and all the WLM property files it contains (if they

exist) are removed. Removing a superclass will fail if any user created subclass still exists (subclass other than Default and Shared).

Only root can remove a superclass. Only authorized users whose user ID or group ID matches the user name or group name specified in the attributes adminuser and admingroup of a superclass can remove a subclass of this superclass.

The options for this command are:

- d Config\_dir To use /etc/wlm/Config\_dir as alternate directory for the properties files. If this flag is not used, the configuration files in the directory pointed to by /etc/wlm/current are used.
- S Superclass To specify the name of the superclass when removing a subclass. There are two ways of specifying the subclass *Sub* of superclass *Super*:
  - Specify the full name of the subclass as *Super.Sub* and not use *-s*.
  - Use the *-s* option to give the superclass name and use the short name for the subclass:  

```
rmclass <options> -S Super Sub
```

### 3.2.3.2 Using SMIT

This section explains the execution on the SMIT interface.

#### ***Working with sets of subclasses***

This method of working with sets of subclasses is only applicable to SMIT. WSM uses a different approach to work with classes. It consists of changing the context you are currently working in into the superclass environment. This is the best way for a superclass administrator to work because he or she does not have any privileges to work in any other environment besides the scope of his or her own superclass. Once inside the context of a superclass A, every class that is listed, created, changed, or removed (even specifying only its short name) will always be treated as a subclass of superclass A.

The context to a specified superclass in SMIT can be changed through the *Work on a set of Subclasses* screen. After selecting the superclass to be worked on, the list of its subclasses is displayed. From this point on, any work in other SMIT screens in this same SMIT session is done *inside* this superclass environment.

To know in which context the current work is, the *Show current focus (Configuration, Class Set)* screen can be accessed in the SMIT session



where the context was changed. The configuration shown in the output of this command is, by default, the currently-running one. However, you can work on other configurations (leaving the currently-running one untouched) if you select a configuration to work with inside the *Work on alternate configurations* screen (see Section 3.2.2, “Working with WLM configurations” on page 56 for further details on how to work with alternate configurations).

**Note**

After exiting SMIT and reentering it, the context is drawn back to the root of the currently-running WLM configuration.

So, if the context is changed into the *devlt* superclass in configuration *Config*, from the example in Table 2 on page 26, the *focus* is the output shown in Figure 13. Note that because the configuration focus has not been changed, the working configuration is presented as being the currently-running one.

```
COMMAND STATUS
Command: OK          stdout: yes          stderr: no
Before command completion, additional instructions may appear below.
Configuration: current
Class set: Subclasses of devlt/
current -> Config

F1=Help          F2=Refresh          F3=Cancel          F6=Command
F8=Image         F9=Shell            F10=Exit           /=Find
n=Find Next
```

Figure 13. Show current focus screen in SMIT

If the configuration focus had been changed into, for instance, a configuration named *Config\_2*, and the class focus had been changed into the set of subclasses of superclass *OLTP*, then the:

- Configuration focus in Figure 13 was *Config\_2*
- Class set was *Subclasses of OLTP*
- Currently-running configuration was *Config*

### Adding a class

To create a class through SMIT, simply access the *Add a class* screen, or use the following fastpath:

```
# smitty wlmaddclass
```

The screen shown in Figure 14 will appear.

```

                                General characteristics of a class
Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                [Entry Fields]
* Class name                      [devlit]
Description                        []
Tier                               [0]                +-#
Resource Set                       +
Inheritance                        [Yes]            +
User authorized to assign its processes to this cl []  +
  ass
Group authorized to assign its processes to this c []  +
  lass
User authorized to administrate this class [bob]      +
(Superclass only)
Group authorized to administrate this class []        +
(Superclass only)
Localshm                           [No]             +

F1=Help      F2=Refresh    F3=Cancel    F4=List
F5=Reset     F6=Command    F7=Edit     F8=Image
F9=Shell     F10=Exit      Enter=Do
```

Figure 14. *smitty wlmaddclass*

In this screen, the system administrator can create a superclass by entering its name or a subclass by entering its *full* name (superclass.subclass). The superclass must already exist for this to work. Every other attribute works exactly the same for both superclasses and subclasses.

If the screen, *Work on a set of subclasses*, has been accessed to change into a superclass' context (see "Working with sets of subclasses" on page 66, for further information about how to change the focus), the *Add a class* screen will operate on the environment of the chosen superclass. While operating under a superclass' scope, the short name can be specified when creating a subclass for that superclass.

**Note**

Remember that the scope will be returned to the root of the currently-running configuration if the SMIT session is exited and restarted.

**Updating a class**

In SMIT, the characteristics of a class can be changed in the *Change/Show Characteristics of a class* screen, shown in Figure 15, or with the following fastpath:

```
# smitty wlmchclass
```

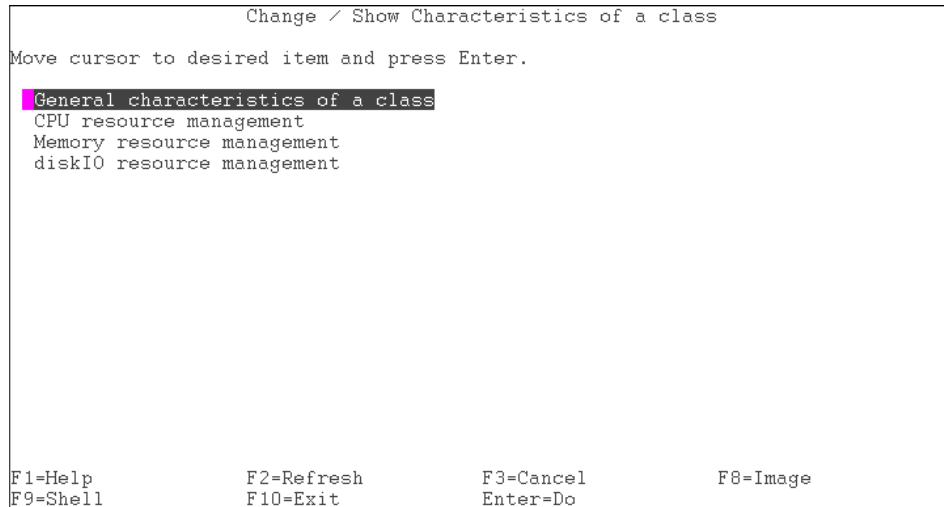


Figure 15. *smitty wlmchclass*

In the *General characteristics of a class* screen, shown in Figure 16 on page 70, the class attributes (tier, inheritance, adminuser, admingroup, rset, authuser, authgroup, and localshm) can be changed or set.

```

                                General characteristics of a class
Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                [Entry Fields]
Class name                        devlt
Description                        []
Tier                              [0]                + #
Resource Set                       +
Inheritance                        [No]             +
User authorized to assign its processes to this cl [] +
ass
Group authorized to assign its processes to this c [dev] +
lass
User authorized to administrate this class      [] +
(Superclass only)
Group authorized to administrate this class      [] +
(Superclass only)
Localshm                            [No]             +

F1=Help      F2=Refresh      F3=Cancel      F4=List
F5=Reset     F6=Command     F7=Edit       F8=Image
F9=Shell     F10=Exit        Enter=Do

```

Figure 16. General characteristics of a class screen in SMIT

Any of the resources' relative attributes (shares and minimum and maximum limits) can be changed under the option referring to the required resource (CPU, memory, or disk I/O).

This way, to change CPU's shares to 20 in *devlt* class from the example in Table 2 on page 26, you need to access the *CPU resource management* screen, shown in Figure 17 on page 71.

```

CPU resource management

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                [Entry Fields]
Class name                       devlt
Shares                           [20] #
Minimum (%)                       [0] #
Maximum (%)                       [100] #
Absolute Maximum (%)              [100] #

F1=Help      F2=Refresh      F3=Cancel      F4=List
F5=Reset     F6=Command     F7=Edit       F8=Image
F9=Shell     F10=Exit       Enter=Do

```

Figure 17. CPU resource management screen in SMIT

To change the memory minimum limit to 10 percent, you need to access the *Memory resource management* screen shown in Figure 18.

```

Memory resource management

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                [Entry Fields]
Class name                       devlt
Shares                           [-] #
Minimum (%)                       [10] #
Maximum (%)                       [100] #
Absolute Maximum (%)              [100] #

F1=Help      F2=Refresh      F3=Cancel      F4=List
F5=Reset     F6=Command     F7=Edit       F8=Image
F9=Shell     F10=Exit       Enter=Do

```

Figure 18. Memory resource management screen in SMIT

If the *Work on a set of subclasses* screen has been accessed to change into a superclass' context (see Section "Working with sets of subclasses" on page 66, for further information on how to change the focus), the *Change/Show Characteristics of a class* screen will operate on the

subclasses of the chosen superclass. While operating under a superclass' scope, the short name can be specified when changing a subclass of that superclass.

**Note**

Remember that the scope will be returned to the root of the currently-running configuration if the SMIT session is exited and restarted.

**Listing the classes**

In SMIT, the classes can be listed through the *List all classes* screen or the following fastpath:

```
# smitty wmlsclass
```

When under the scope of the general configuration, the screen will show all superclasses configured as shown in Figure 19.

```
COMMAND STATUS
Command: OK          stdout: yes          stderr: no
Before command completion, additional instructions may appear below.
System
Default
db1
db2
devlt
VPs
acctg
Shared

F1=Help          F2=Refresh      F3=Cancel      F6=Command
F8=Image         F9=Shell       F10=Exit      /=Find
n=Find Next
```

Figure 19. smitty wmlsclass

If the *Work on a set of subclasses* screen has been accessed to change into a superclass' context (see Section "Working with sets of subclasses" on page 66, for further information on how to change the focus), the *List all classes* screen will print out the subclasses of the chosen superclass.

**Note**

Remember that the scope will be returned to the root of the currently-running configuration if the SMIT session is exited and restarted.

**Removing a class**

In SMIT, a class can be removed by accessing the *Remove a class* screen or using the following fastpath

```
# smitty wlmrmclass
```

A superclass is removed by specifying its name, and a subclass is removed by specifying its full name.

If the *Work on a set of subclasses* screen has been accessed to change into a superclass' context (see Section "Working with sets of subclasses" on page 66 for further information about changing the configuration's focus), the *Remove a class* screen will operate on the subclasses of the chosen superclass. While operating under a superclass' scope, the short name can be specified when removing a subclass of that superclass.

**3.2.3.3 Using WSM**

This section explains the execution on the WSM interface.

**Adding a class**

To add a class in WSM, several paths can be taken. The first way is to create a new class in the currently running configuration inside the *Overview and Tasks* screen (see Figure 42 on page 104). In this screen, click on the *Create a new class in the default configuration* link. This will guide you through the New Class wizard (Figure 20 on page 74), which sets up a new class and its attributes (tier, inheritance, adminuser, admingroup, resource set, authuser, authgroup, and localshm). The class can be a superclass with the name, *supername*, or a subclass of an already-existing superclass with the name, *supername.subname*:

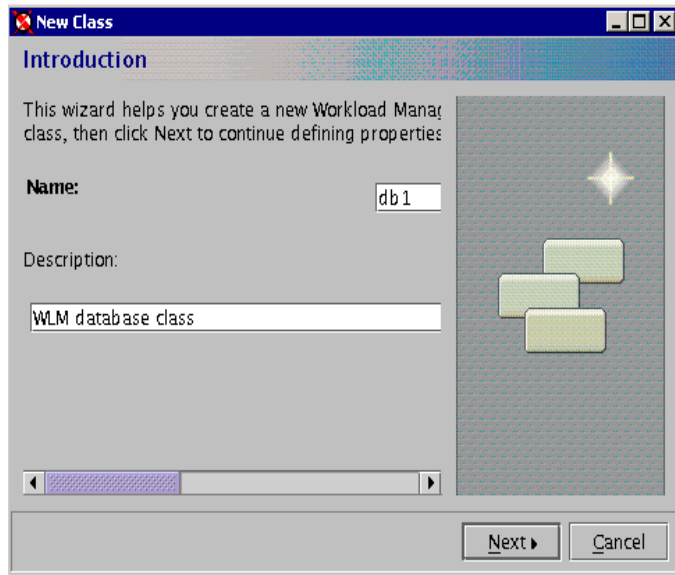


Figure 20. New Class wizard in WSM

Another way to create a class is to right-click on the configuration to be altered (see Figure 21 on page 75) inside the *Configurations/Classes* screen, and choose the *New Class* option.



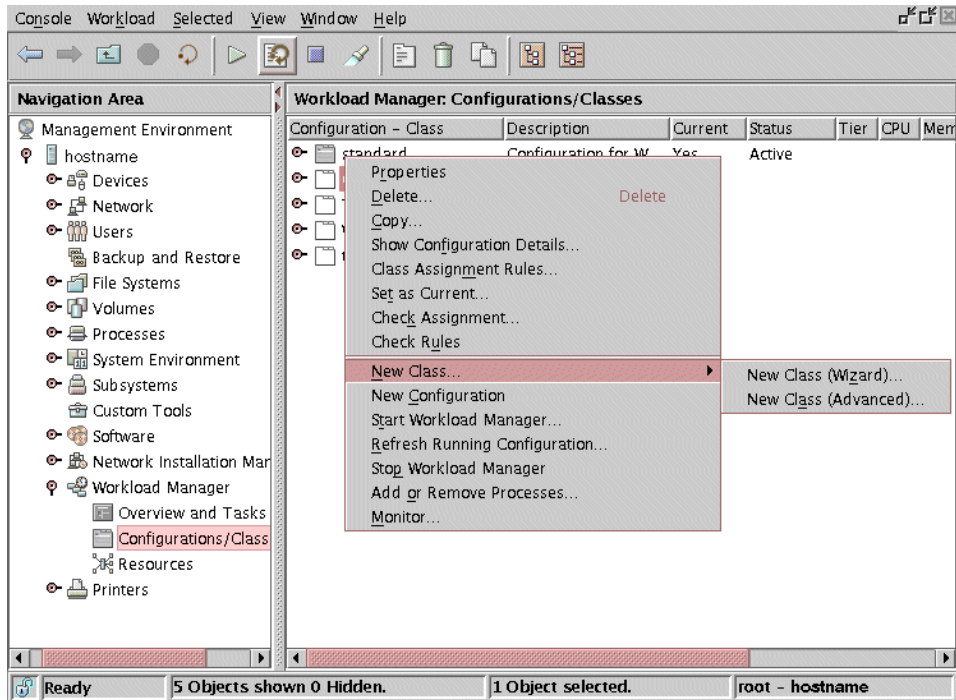


Figure 21. Create a class in Configurations/Classes screen in WSM

From here, you can choose to use the wizard mentioned earlier (see Figure 20 on page 74) or the Advanced configuration tool, which, in addition, allows other class attributes to set up shares and limits for the class being created. The class can be a superclass with the name, *supername*, or a subclass of an already-existing superclass with the name, *supername.subname*. Create the class shown in Figure 22 on page 76.

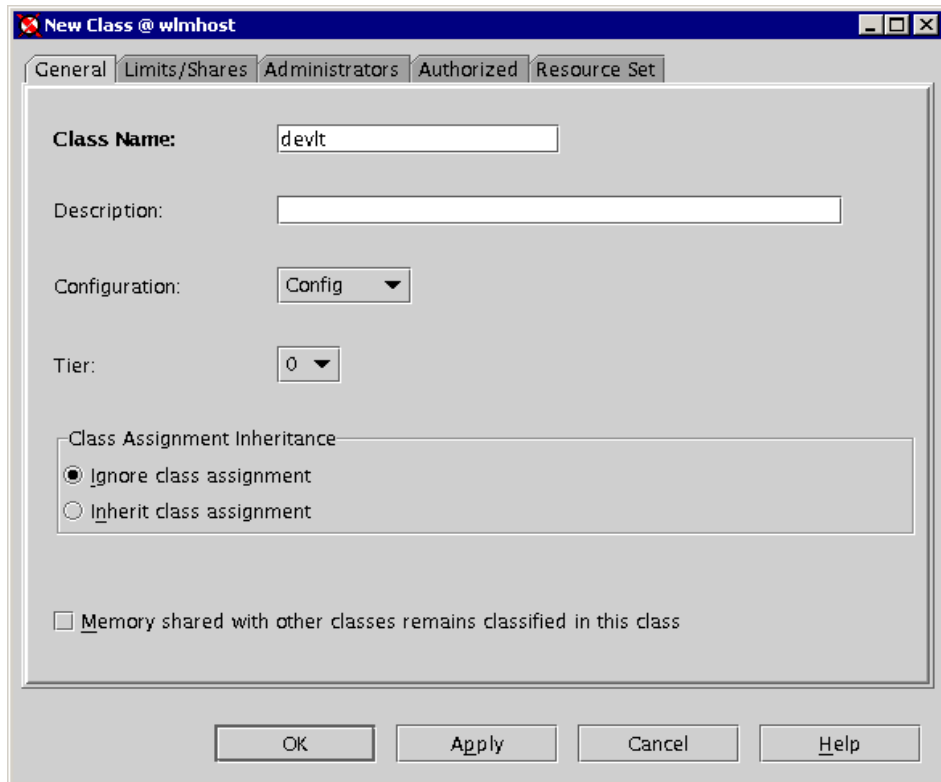


Figure 22. New Class advanced in WSM - Setting up limits and shares

The third way to create a class is to click the *expand* icon, found at the right hand side of the configuration selected, to expand the view to all the configured classes in that configuration.



All the superclasses with subclasses will also be shown with an expand icon that can be selected to extend the view into the subclass level. Right clicking the name of a class displays the class options screen as shown in Figure 23 on page 77.



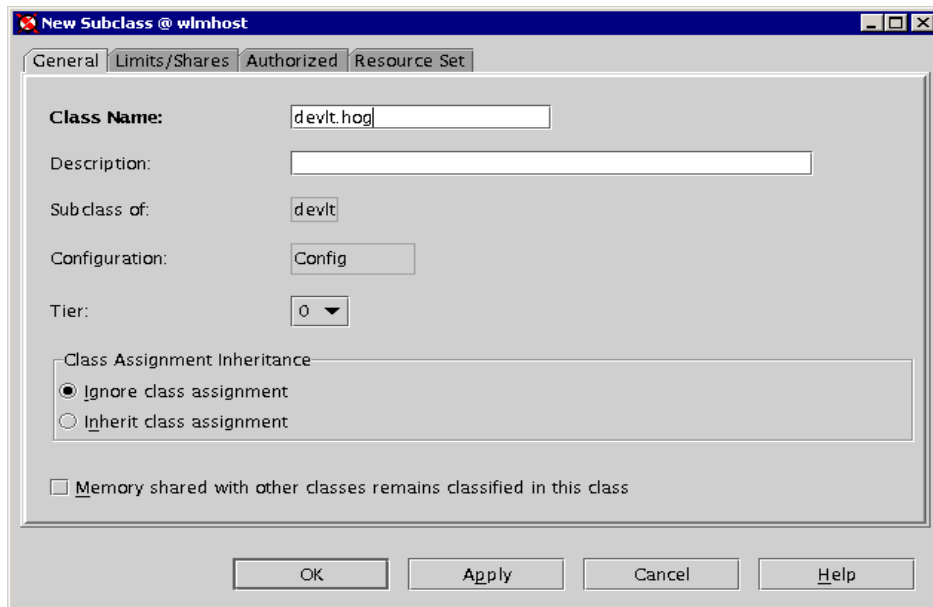


Figure 24. New Subclass Advanced in WSM

- Copy the selected class into another configuration (*Add to Another Configuration* as shown in Figure 25):

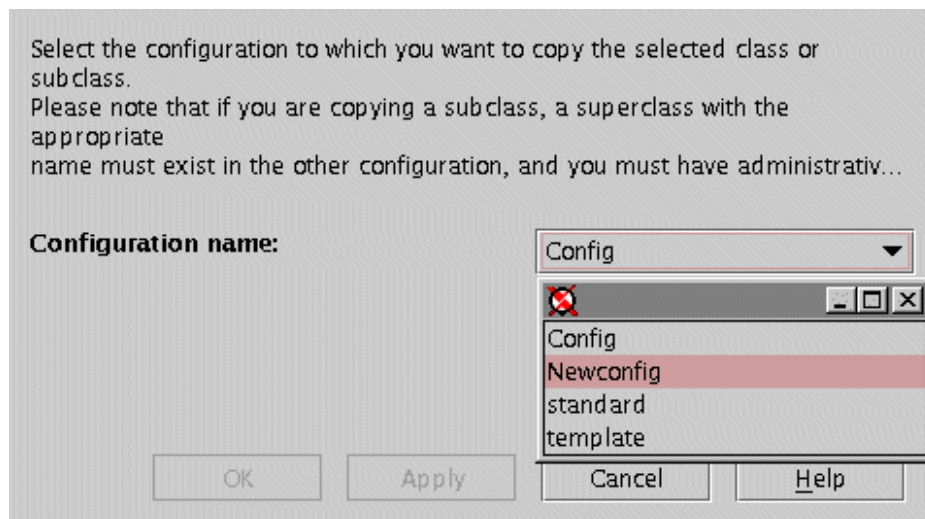


Figure 25. Add to another configuration screen in WSM

### Updating a class

In WSM, the classes attributes can be changed in the *Configurations/Classes* screen in the classes view (or the subclasses view for a specific superclass) by right-clicking the name of the class to update and selecting *Properties*. This can also be done by simply double-clicking the name of the class.

An example of changes that can be made to shares and limits in this screen is shown in Figure 26.

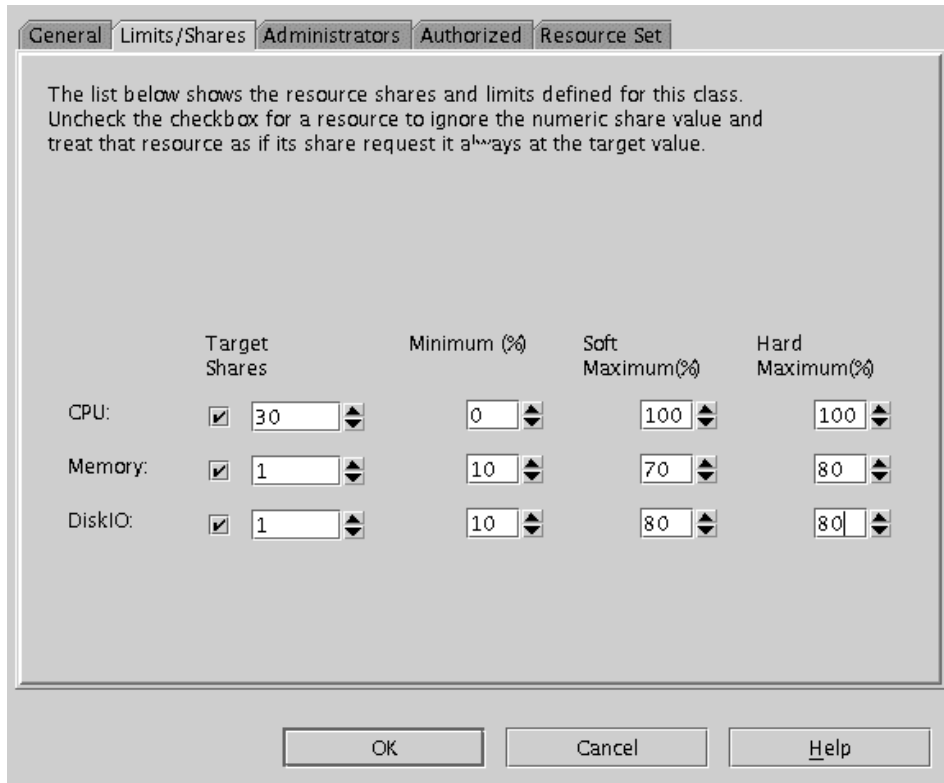


Figure 26. Changing class properties in WSM

Alternatively, the properties icon at the top of the WSM window can be clicked for the same purpose:



### Listing the classes

There are two views in WSM where the classes and their attributes for the chosen configuration can be seen:

- In the *Configurations/Classes* screen, select the configuration option, *Show Configuration Details* (see also Figure 11 on page 60).
- In the *Configurations/Classes* screen, two icons can be seen at the top of the WSM window. They are Tree and Tree-Details:



The first icon sets up a view that only shows the tree of configurations, superclasses, and subclasses. The second one creates a view in which some of the class attributes can be seen as shown in Figure 27.

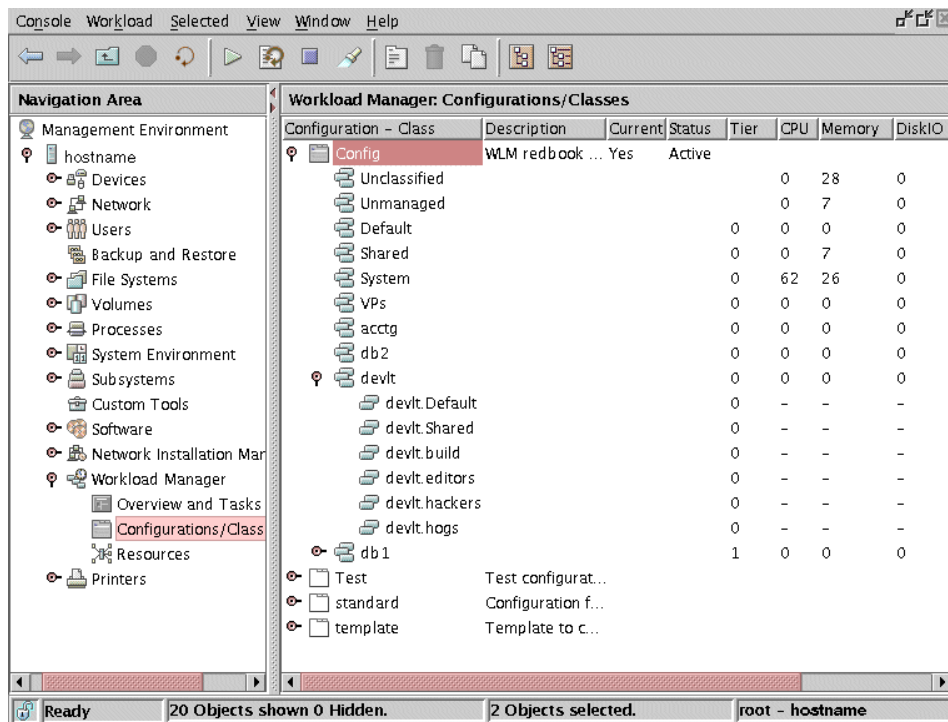


Figure 27. Tree-Details view in WSM

### Removing a class

To remove a class in WSM, the system administrator can highlight the class to be deleted and press the Delete key, right-click on the class name and chose the *Delete* option, or click on the delete icon at the top of the WSM' window:



### 3.2.4 AIX Version 4.3.3 maintenance level 8 wlmset command

A new command `wlmset` is used to customize WLM by setting global option flags that alter the standard behavior.

The syntax is:

```
wlmset [-a hardcpumax=[yes|no]] [-a shared=[yes|no]]
```

Flags:

- |                                     |  |
|-------------------------------------|--|
| <code>-a hardcpumax=[yes no]</code> | Possible values are <i>yes</i> or <i>no</i> . <i>Yes</i> means that the CPU maximum limits should be treated as absolute limits and should never be exceeded. <i>No</i> means that the CPU maximum limits should be treated as <i>soft</i> limits and can be exceeded if there is no contention for the CPU resource. <i>No</i> is the default when <code>wlmset</code> is not used or is used without specifying the <code>hardcpumax</code> keyword.   |
| <code>-a shared=[yes no]</code>     | Possible values are <i>yes</i> or <i>no</i> . <i>Yes</i> means that shared memory segments should <i>migrate</i> to the Shared class when accessed (page fault) by a process belonging to a different class as that of the segment. This is the default when <code>wlmset</code> is not used or is used without specifying the <code>shared</code> keyword. <i>No</i> means that the segment should remain in the class it was first classified into, regardless of the class of the processes accessing it. |

The fact that these options are global WLM flags means that they apply to all the classes when set. The `wlmset` command can be used whether or not WLM is active. For consistent results, it is recommended that `wlmset` be used to customize Workload Manager prior to starting it. Otherwise, in the case of the shared memory for instance, shared memory segments accessed by

processes in different classes will go into the Shared class (and remain there) prior to the shared flag being set, and the shared memory segments accessed by processes in different classes after the flag has been set will remain in their class of origin. The customization of WLM is done by setting global flags in memory, and thus has to be done every time a system is rebooted. The best way to do it, when using the same set of flags for each reboot, is to run `wlmset` from the `inittab` prior to starting WLM. The `wlmset` command is restricted to the root user. This command is provided in AIX Version 4.3.3 maintenance level 8 to give 4.3.3 users early access to some of the features available in Workload Manager with AIX 5L.

**Note**

The `wlmset` command is not supported by AIX 5L. AIX 5L supports both *hard* and *soft* maximum limits that can be set independently for all resource types managed by WLM, including CPU. AIX 5L also provides a *per class* attribute (`localshm`) to prevent shared memory segments to go into the Shared class on a per class basis. It is expected that users of `wlmset` will modify their WLM configuration files to take full advantage of the corresponding AIX 5L features when they upgrade from AIX Version 4.3.3 to AIX 5L.

### 3.2.5 Working with rules

After configuring the needed classes, the process assignment criteria must be set up to have the applications classified according to the configuration design. This is done by creating the class assignment rules.

#### 3.2.5.1 Editing the rules files on the command line

As shown in Section 2.5.3, “Class assignment rules” on page 23, an assignment rule is a set of attributes with which the characteristics of a given process can be matched (or not). The rules file has the same format for both superclasses and subclasses, the only difference being the non-existence of a System class rule in the subclasses’ rules files due to the non-existence of System subclasses.

The rules file for the example shown in Table 2 on page 26 would be:



```

* IBM_PROLOG_BEGIN_TAG
* This is an automatically generated prolog.
* bos43N src/bos/etc/wlm/rules 1.1
* Licensed Materials - Property of IBM
* (C) COPYRIGHT International Business Machines Corp. 1999
* All Rights Reserved
* US Government Users Restricted Rights - Use, duplication or
* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
* IBM_PROLOG_END_TAG
* class resvd user group application type tag
db1 - - - /usr/oracle/bin/db* - _db1
db2 - - - /usr/oracle/bin/db* - _db2
devlt - - dev - 32bit -
VPs - bob - - - -
acctg - !ted - acct* - - -

```

The *resvd* attribute (reserved for future use) must always exist and must always be set to hyphen (-).

Any hyphens (-) at the end of a rule can be omitted, as long as no subsequent attribute is set. For instance, the rule for the *acctg* superclass could be

```
acctg - - acct*
```

but the rule for the *db1* superclass could *not* be

```
db1 - - - /usr/oracle/bin/db* _db1
```

because *\_db1* would be interpreted by WLM as the *type* attribute, returning an invalid type attribute error.

For the type attribute position, one or more values can be specified either divided with commas (,) for 'or', or with plus signs (+) for 'and'. For instance, the rule for the *devlt* class in the previous example could be:

```
devlt - - dev - 32bit,plock+fixed -
```

specifying that the processes classified under this class needed to be either 32 bit processes or have called *plock* and be fixed priority at the same time.

### 3.2.5.2 Using SMIT

This section explains the execution on the SMIT interface.

#### **Adding a rule**

In SMIT, a rule can be created by accessing the *Class assignment rules* and *Create a new rule* screens, or by using the following fastpath:

```
# smitty crewlms
```

If the *Work on a set of subclasses* screen has been accessed to change into a superclass' context (see Section "Working with sets of subclasses" on page 66 for more information about changing the configuration's focus), the *Create a new rule* screen will work under the scope of the chosen superclass. It will, therefore, create the rules for the superclass' subclasses. While operating under a superclass' scope, the short name can be specified when creating rules for a subclass of that superclass.

**Note**

Remember that the scope will be returned to the root of the currently-running configuration if the SMIT session is exited and restarted.

In Figure 28, you can see an example of the creation of a rule for the *hogs* subclass of the *devlt* superclass (from the example in Table 2 on page 26), after changing into *devlt* superclass' scope.

```
                                Create a new Rule

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

* Order of the rule          [2]          #
* Class name                 hogs          +
* User                       [-]          +
* Group                      [-]          +
Application
  Type                       [32bit,plock+fixed]  +
  Tag                        [-]

F1=Help      F2=Refresh  F3=Cancel    F4=List
F5=Reset     F6=Command  F7=Edit     F8=Image
F9=Shell     F10=Exit    Enter=Do
```

Figure 28. Create a new Rule screen in SMIT

The *type* for the *hogs* subclass in Figure 28 could be configured as *32bit,plock+fixed* to specify that a process classified under this subclass must be either 32 bit or have called *plock* and have fixed priority at the same time.



```

Change / Show Characteristics of a Rule

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                [Entry Fields]
Order of the rule                 2
New Order of the rule            [2]                #
* Class name                     db1                +
* User                           [-]                +
* Group                           [dbw]            +
Application                      [/usr/oracle/bin/db*]
Type                             [-]                +
Tag                              [_db1]

F1=Help      F2=Refresh    F3=Cancel    F4=List
F5=Reset     F6=Command    F7=Edit     F8=Image
F9=Shell     F10=Exit      Enter=Do

```

Figure 30. Change/Show Characteristics of a Rule screen in SMIT

If the *Work on a set of subclasses* screen has been accessed to change into a superclass' context (see Section "Working with sets of subclasses" on page 66 for more information about changing the configuration's focus), the *Change/Show Characteristics of a Rule* screen will operate on the rules of the subclasses of the chosen superclass.

**Note**

Remember that the scope will be returned to the root of the currently-running configuration if the SMIT session is exited and restarted.

If the F4 function key is pressed on the *type* attribute and more than one value chosen, they get comma-separated (or ORed). If the AND option is required, the plus sign must be *manually* entered in this attribute.

**Listing the rules**

The rules in SMIT can be listed by accessing the *Class assignment rules* screen, followed by the *List all Rules* screen shown in Figure 31 on page 87.

```

                                COMMAND STATUS
Command: OK                stdout: yes          stderr: no

Before command completion, additional instructions may appear below.

_# Class      User      Group      Application      Type      Tag
001 db1       -         -          /usr/oracle/bin/db*  -         _db1
002 db2       -         -          /usr/oracle/bin/db*  -         _db2
003 devlt     -         dev        -                 32bit    -
004 VPs      bob,sally -         -                 -         -
005 acctg    !ted     acct*     -                 -         -
006 testacct -         -          /usr/bin/vmstat     -         -
007 System   root     -         -                 -         -
008 Default  -         -         -                 -         -

F1=Help      F2=Refresh   F3=Cancel   F6=Command
F8=Image     F9=Shell     F10=Exit    /=Find
n=Find Next

```

Figure 31. List all Rules screen in SMIT

If the *Work on a set of subclasses* screen has been accessed to change into a superclass' context (see Section "Working with sets of subclasses" on page 66 for more information about changing the configuration's focus), the *List all Rules* screen will print out the rules of the subclasses of the chosen superclass.

**Note**

Remember that the scope will be returned to the root of the currently-running configuration if the SMIT session is exited and restarted.

**Removing a rule**

In SMIT, a rule can be deleted by accessing the *Class assignment rules* screen and choosing *Delete a Rule*. See Figure 29 on page 85 for details about selecting a rule.

**3.2.5.3 Using WSM**

This section explains the execution on the WSM interface.

### Adding a rule

Working with rules in WSM (at general configuration or superclass levels) can be done in the *Configurations/Classes* screen by right clicking on the configuration to be changed and choosing the *Class Assignment Rules* option. The Class Assignment Rules screen is shown in Figure 32.

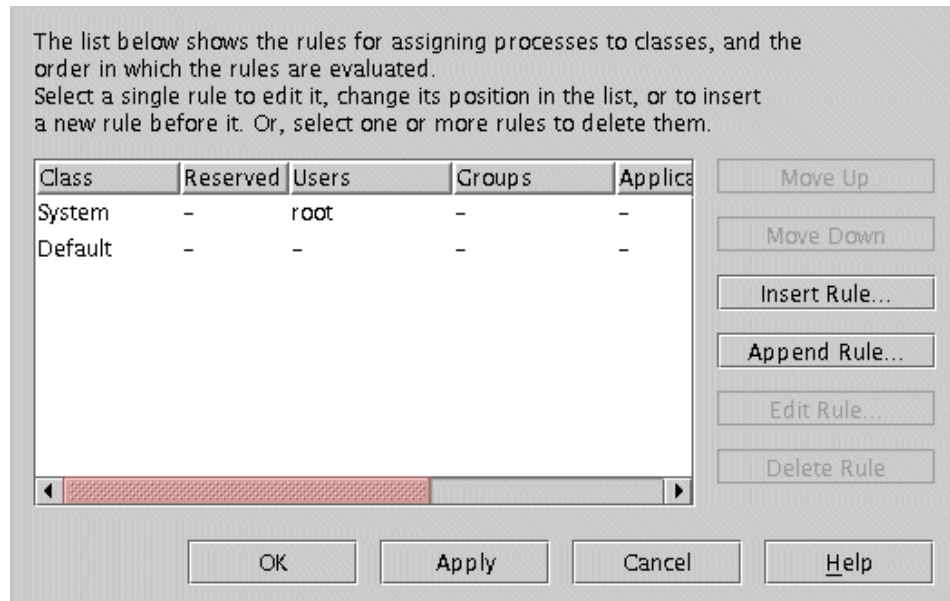


Figure 32. Class Assignment Rules screen in WSM

To add a rule, click on *Insert Rule*. The attributes of a rule can now be set (user, group, application, process type, and tag) as shown in Figure 33 on page 89.

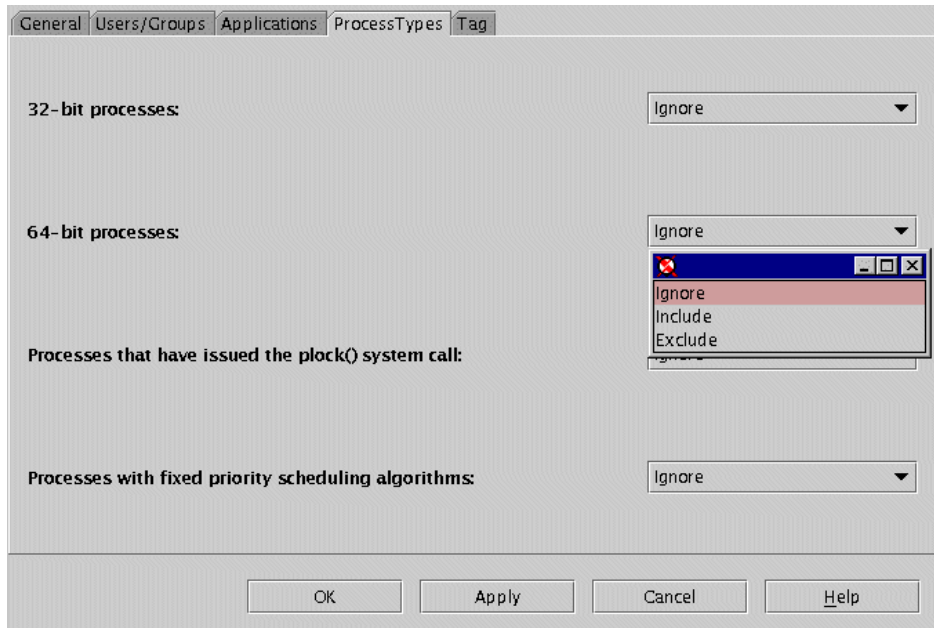


Figure 33. Adding a rule in WSM - setting process type attribute

After a rule has been created, it can be moved up or down the rules list by clicking on the options **Move Up** or **Move Down** in the *Class Assignment Rules* screen.

#### **Changing a rule**

To change a rule in WSM, click **Edit Rule** in the *Class Assignment Rules* screen (Figure 32 on page 88) at either the superclass or subclass level. A screen similar to the one in Figure 33 will be shown, allowing the system administrator to alter the rule's attributes.

#### **Listing the rules**

Listing the rules in WSM is done by simply accessing the *Class Assignment Rules* screen (see Figure 32 on page 88) either at the superclass or subclass level.

#### **Removing a rule**

In WSM, a rule can be deleted by accessing the *Class Assignment Rules* screen, highlighting the rule to be removed, and clicking **Delete Rule**.

### 3.2.6 Checking the configuration - wlmcheck

The configuration is set and running. Now is probably a good time to use the WLM checking command, `wlmcheck`. This command checks automatic assignment rules and/or determines the class in which a process with a specified set of attributes will be classified.

#### **Using the command line**

The syntax for the `wlmcheck` command is as follows:

```
wlmcheck [ -d config_dir ] [ -a <process attributes> ] [ -q ]
```

The following are the options for this command:

- d config      Uses the WLM property files in `/etc/wlm/config` instead of the values currently loaded into the kernel (active configuration).
- a attributes    Used to pass a set of values for the classification attributes of the process in order to determine which class the process will be put into. This is a way to check that the assignment rules are correct and classify processes as expected.
- q              Suppresses the output of the status of the latest activation/update of WLM (stands for quiet).

The `wlmcheck` command with no arguments returns the status of WLM and makes some coherency checks:

- Displays the current status of WLM (running/non running, active/passive, resets bindings active/non active).
- Displays the status files that report the last loading errors, if any.
- Checks the coherency of the assignment rules files (syntax, existence of the classes, validity of user and group names, application path names, and so on).

If the `-d config_dir` option is not specified, the checks are performed on the configuration that is loaded into the kernel at this time. If WLM is not active, an error message is displayed. Specifying a configuration with `-d config_dir` allows you to perform the checks on configuration files, including the ones in `/etc/wlm/current`.

Used with the `-a` option, the `wlmcheck` command displays the class the process would be assigned to according to the set of assignment rules of the specified configuration.

The attributes are given as a string similar to the format used in the assignment rules file (single string with several space-separated fields) and



should be enclosed in quotes. The fields are the same as in the rules file and appear in the same order; *reserved*, *user name*, *group name*, *application path name*, *process type*, and *application tag* (see Section 3.1, “Property files” on page 44, for more information about the rules property file).

The difference is that, unlike in the assignment rules:

- The class field is omitted (it is actually an output of the `wlmcheck` command).
- Each field can have, at most, one value. Exclusions (!), comma-separated lists, and wild cards are not allowed.
- At least one field must be specified (have a value different from a hyphen (-)).

In addition, the first two fields are mandatory. The other fields, if not present, will default to a hyphen (-), which means that any value in the corresponding field of an assignment rule is a match. When one or more of the fields in the attribute string are either not present or specified as a hyphen (-), the string is likely to match more than one rule. In this case, the `wlmcheck` command will display all the classes corresponding to all the possible matches.

Example of valid attribute strings:

```
# wlmcheck -a "- root system /usr/bin/vi - -"
# wlmcheck -a "- - staff - 32bit"
# wlmcheck -a "- bob"
```

By default, the `wlmcheck` command outputs the contents of the status files for the last activation or update of WLM.

**Note**

There is no dedicated SMIT interface for `wlmcheck`.

**Using WSM**

In WSM, `wlmcheck` can be invoked to check on the assignment rules coherency and to evaluate to which class a specific process would be assigned. This can be done in the *Configurations/Classes* screen by right clicking the configuration to be checked and choosing *Check Assignment* for the process classification evaluation and *Check Rules* for the coherency test. The Check Assignment screen, shown in Figure 34 on page 92, appears.

Specify or select the attributes of a process to determine to which class a process with those attributes would be assigned. Any text field may be left empty.

**User:**

**Group:**

**Application:**

**Tag:**

**Process Type:**

- 32-bit process
- 64-bit process
- Process has issued the plock() system call
- Process uses a fixed priority scheduling algorithm

OK    Apply    Cancel    Help

Figure 34. Check Assignment in WSM

### 3.2.7 Working with resource sets

WLM uses the concept of resource sets (or rsets) to restrict the processes in a given class to a subset of the system's physical resources. In AIX 5L, the physical resources managed are the memory and the processors. A valid resource set is composed of memory and at least one processor. By default, the system creates one resource set for all physical memory, one for all CPU's, and one separate set for each individual CPU in the system.

#### 3.2.7.1 Rset registry

As mentioned earlier, some resource sets are created for memory and CPU by default. It is possible to create different resource sets by grouping two or more resource sets and storing the definition in the rset registry.

The rset registry services enable system administrators to define and name resource sets so that they can then be used by other users or applications. In order to alleviate the risks of name collisions, the registry supports a two-level naming scheme. The name of a resource set is in the form, `name_space/rset_name`. Both the namespace and `rset_name` may each be 255 characters in size, are case-sensitive, and may contain only upper and lowercase letters, numbers, underscores, and periods (.). The namespace of `sys` is reserved by the operating system and used for rset definitions that represent the resources of the system.

The `smitty rset` command has options to list, remove, and show a specific resource set used by a process, and contains the management tools as shown in Figure 35.

```
Resource Set Management

Move cursor to desired item and press Enter.

List All Resource Sets
List All Resource Sets in a given namespace
List All System RADs
List Application-defined Resource Sets
Remove Application-defined Resource Sets
Show a Process Partition
Manage Resource Set Database

F1=Help      F2=Refresh   F3=Cancel    F8=Image
F9=Shell     F10=Exit    Enter=Do
```

Figure 35. SMIT main panel for Resource Set Management

To create, delete, or change a resource set in the `rset` registry, you must select *Manage Resource Set Database* in the SMIT panel. In this panel, it is also possible to reload the `rset` registry definitions to make all changes available to the system. Figure 36 on page 94 shows the SMIT panel for `rset` registry management.

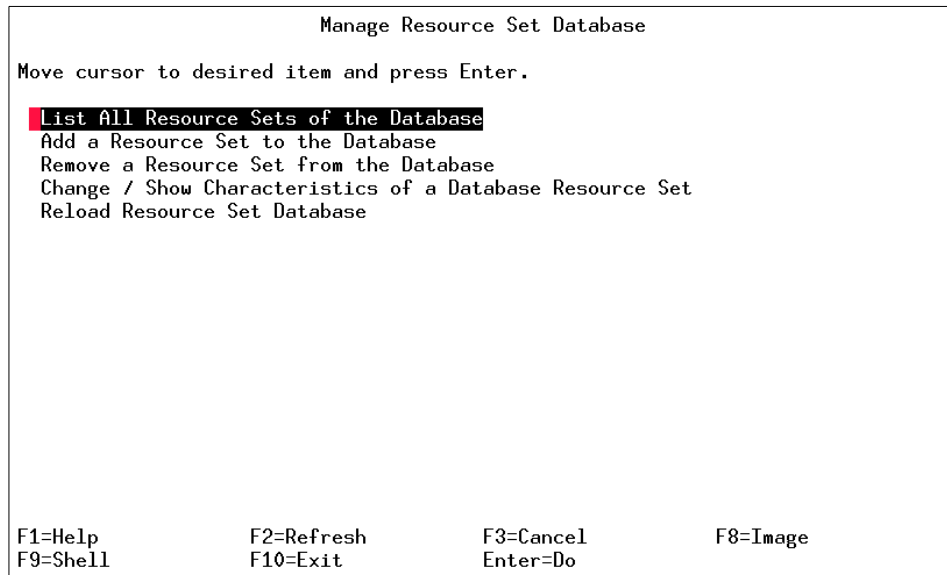


Figure 36. SMIT panel for rset registry management

To add a new resource set, you must specify a name space, a resource set name, and the list of resources. It is also possible to change permissions for the owner and group of this rset. In addition, the permissions for the owner, groups, and others can be specified. Figure 37 on page 95 shows the SMIT panel for this task.

```

Add a Resource Set to the Database

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

* Name Space          [Redbook]          +
* Resource Set Name   [CPU0and1]         +
* Owner               root                +
* Group              system              +
* Owner Permissions   rw                  +
* Group Permissions   r-                  +
* Others Permissions  r-                  +
* Resources           sys/cpu.00001,sys/cpu.> +

F1=Help      F2=Refresh    F3=Cancel    F4=List
F5=Reset     F6=Command    F7=Edit      F8=Image
F9=Shell     F10=Exit      Enter=Do

```

Figure 37. SMIT panel to add a new resource set

Whenever a new rset is created, deleted, or modified, a reload in the rset database is needed to make the changes effective.

### 3.2.7.2 Using the command line

The `lsrset` command lists all resource sets defined. The following is a sample output for the `lsrset` command:

```

# lsrset -av
T Name                Owner  Group  Mode  CPU  Memory  Resources
r sys/sys0            root   system r----- 4    511    sys/sys0
sys/node.00000 sys/mem.00000 sys/cpu.00003 sys/cpu.00002 sys/cpu.00001
sys/cpu.00000
r sys/node.00000      root   system r----- 4    511    sys/sys0
sys/node.00000 sys/mem.00000 sys/cpu.00003 sys/cpu.00002 sys/cpu.00001
sys/cpu.00000
r sys/mem.00000       root   system r----- 0    511    sys/mem.00000
r sys/cpu.00003       root   system r----- 1    0      sys/cpu.00003
r sys/cpu.00002       root   system r----- 1    0      sys/cpu.00002
r sys/cpu.00001       root   system r----- 1    0      sys/cpu.00001

```

To change a resource set for a specific class, you could use the following steps.

To list the current rset of a class enter:

```
# lsclass -C db2

#name:description:tier:inheritance:authuser:authgroup:adminuser:admingroup
:rset:CPUshares:CPUmin:CPUsoftmax:CPUhardmax:memoryshares:memorymin:memory
softmax:memoryhardmax:diskIOshares:diskIOmin:diskIOsoftmax:diskIOhardmax:
localshm

db2::0:yes:::::sys/cpu.00003:-:0:50:70:12:0:100:100:6:0:100:100:no
```

To change the rset of the class enter:

```
# chclass -a rset=sys/cpu.00002 db2
```

After changing the rset, run the `wlmcheck` command to check if this change is correct:

```
# wlmcheck -d Config
WLM is not running.
Checking classes and rules for 'Config' configuration...
System
Default
Shared
db1
db2
devlt
VPs
acctg
testacct
```

If `wlmcheck` completes without any errors as shown in the example above, you can list the result of the rset change with the following command:

```
# lsclass -C db2

#name:description:tier:inheritance:authuser:authgroup:adminuser:admingroup
:rset:CPUshares:CPUmin:CPUsoftmax:CPUhardmax:memoryshares:memorymin:memory
softmax:memoryhardmax:diskIOshares:diskIOmin:diskIOsoftmax:diskIOhardmax:
localshm

db2::0:yes:::::sys/cpu.00002:-:0:50:70:12:0:100:100:6:0:100:100:no
```

If you use a resource name that does not exist in the resource set database, `wlmcheck` will return an error message as shown below:

```
# wlmcheck -d Config
WLM is not running.
Checking classes and rules for 'Config' configuration...
1495-586 Bad Rset attribute for class db2
```

### 3.2.7.3 Using SMIT

Figure 38 shows the SMIT panel where a resource set can be specified for a specific class, which is opened with the following command:

```
#smitty wlmclass_gal
```

```

                                General characteristics of a class
Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                [Entry Fields]
Class name                        devlt
Description                       []
Tier                               [0]                +#
Resource Set                      +
Inheritance                       [No]              +
User authorized to assign its processes to this cl [] +
ass                               +
Group authorized to assign its processes to this c [dev] +
lass                             +
User authorized to administrate this class      [] +
(Superclass only)                 [ ]               +
Group authorized to administrate this class     [ ]               +
(Superclass only)                 [ ]               +
Localshm                           [No]              +

F1=Help      F2=Refresh      F3=Cancel      F4=List
F5=Reset     F6=Command     F7=Edit       F8=Image
F9=Shell     F10=Exit       Enter=Do
```

Figure 38. Resource Set definition to a specific class in SMIT

### 3.2.7.4 Using WSM

In WSM, the resource set can be changed in the *Configurations/Classes* screen in the classes view (or the subclasses view for a specific superclass) by right-clicking the name of the class to update and selecting *Properties*. This can also be done by simply double-clicking the name of the class.

Figure 39 on page 98 shows the WSM window where a resource set can be specified for a specific class.

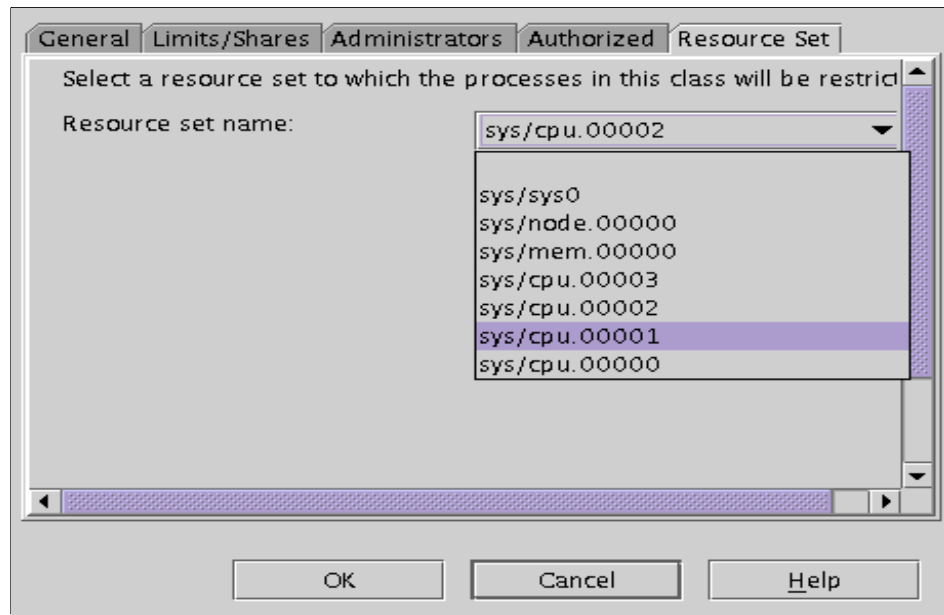


Figure 39. Resource Set definition to a specific class in WSM

### 3.3 WLM operation

Operating WLM consists, basically, of turning it on and off, and refreshing its running configuration for any changes made. The main issue when operating WLM is the three different modes in which it can be started. The following sections focus on all these points.

#### 3.3.1 Modes of operation

WLM can be turned on in one of three modes:

- In the *active* mode, WLM classifies new and existing processes and regulates their resource usage for CPU, Memory, and Disk I/O according to the class shares and resource limits defined in the active WLM configuration. This is the normal mode of operation.
- In active mode with CPU regulation only. In this mode WLM does not regulate Memory and Disk I/O.
- A *passive* mode is provided to help system administrators understand what the resource requirements of their applications on a system are, helping them better tune their WLM configurations.



In this mode, WLM classifies new and existing processes and gathers statistics about their resource usage, but does not regulate this usage. In this mode, the processes compete for resources exactly as they would if WLM was off. The `wlmstat` command can then be used to get snapshots of the resource usage for the different classes.

The `wlmcntrl` command lets you switch the mode of operation at any time. In addition, rset binding can be turned on or off, so that all classes have access to the whole resource set of the system use (use `wlmcntrl -g` to turn it off). Possible combinations are:

- active mode + rset on
- active mode + rset off
- active mode (CPU only) + rset on
- active mode (CPU only) + rset off
- passive mode + rset on
- passive mode + rset off

The *passive* mode can be used for various purposes. Here are a few examples:

- Before fully enabling WLM on a production system, the system administrator could use the passive mode to check the assignment rules.

With WLM started in the passive mode, all the processes would be classified according to the assignment rules, and the system administrator could use `ps` to check that the various applications are classified in the correct class. Because there is no regulation in this mode, this has virtually no impact on the users of the system.

- When the system administrator is satisfied with the classification, the system can be allowed to run for some time in *passive* mode to gather base line resource usage statistics with the `wlmstat` command. These statistics provide a reference that can be used to determine how to apply the shares and, if necessary, resource limits to favor critical applications and/or restrain less important work to match the business goals.

### 3.3.2 Start/Stop/Update WLM - `wlmcntrl`

WLM is not enabled at system installation and must be activated by the system administrator. This may be performed from the command line with the `wlmcntrl` command or from the administration tools SMIT or WSM. Either way, the `wlmcntrl` command does some very important processing of the

WLM property files before passing the configuration information to the operating system. In particular:

- It converts all the user and group names into numerical user IDs and group IDs.
- It expands the wild cards (if applicable) in the users, groups, or application pathnames in the rules files and accesses all the target application files to transform the pathnames into information usable by the kernel, such as device identifiers and inode numbers.

The `wlmcntrl` command will issue an error message and will not start WLM if it cannot translate a user or group name in a rule. If one or more of the application file names cannot be accessed, the `wlmcntrl` command will issue warning messages identifying the files causing a problem, but will still start WLM. The problem files' names will just be ignored. Even though this condition is not fatal, you should try to understand why some of the application files cannot be accessed and take corrective actions. The problem could be due to a file system that was not mounted or an NFS server being down, for example. If none of the application files listed in an assignment rule can be accessed, the entire rule is ignored.

The following describes the functionality of each of the aforementioned WLM operating methods.

### 3.3.2.1 Using the command line

From the command line, WLM can be started, updated, stopped, and queried by running the `wlmcntrl` command with the appropriate option.

The syntax for this command can take two forms:

```
wlmcntrl [[ -a | -p ] [ -c ] [ -g ] [ -d Config_dir ]] [ -o | -q ]
```

or

```
wlmcntrl -u [ -S Superclass | -d Config_dir ]
```

The options of the `wlmcntrl` command are:

- |                 |   |
|-----------------|---|
| <code>-a</code> | To start WLM in active mode or to switch from passive to active mode. This is the default when no option other than <code>-d</code> is specified. |
| <code>-c</code> | To start WLM in active mode for CPU and passive mode for memory and disk I/O.   |
| <code>-p</code> | To start WLM in passive mode, or to switch from active to passive mode.   |

<code>-d Config_dir</code>	To set <code>/etc/wlm/Config_dir</code> as the directory to use for the classes, resource limits, resource shares, rules files, and superclasses directories.
<code>-g</code>	To disable the enforcement of resource set bindings at WLM startup.
<code>-o</code>	To stop WLM.
<code>-u</code>	To send an update request to change the attributes of the running classes, or to change the current configuration in use. Can be used alone or in conjunction with <code>-s</code> and <code>-d</code> options.
<code>-S Superclass</code>	To specify the running superclass whose attributes are to be updated. Can only be used in conjunction with the <code>-u</code> option.
<code>-q</code>	To query the WLM state. Returns 0 if WLM is running in active mode, 1 if WLM is not started, and 2 if WLM is running in passive mode. A message indicating the current state of WLM is the output.

**Note**

The `-c` option of `wlmcntrl` is not accessible through SMIT or WSM.

A system administrator has the option of modifying the current configuration files and making the changes active without stopping WLM by using:

```
# wlmcntrl -u
```

Any attributes of classes in the current configuration can be changed, and are then used to reclassify the processes to which they apply.

Administrators also have the option of creating a new configuration with different classes, shares, limits, and/or tier numbers and making this new configuration active without stopping WLM by using:

```
# wlmcntrl -u -d <new_config>
```

This second option is particularly useful because it allows administrators to create different configurations, such as a *day\_config* and a *night\_config*, and flip from one to the other at given times using the AIX `cron` facility.

Starting WLM by a direct invocation of the `wlmcntrl` command, however, only causes WLM to be initialized at that moment, not on every system boot. To configure WLM to start automatically at system boot, manually edit `/etc/inittab`. Make sure the WLM entry is placed directly after the mounting of filesystems so that the maximum number of processes are classified.

The line to add to /etc/inittab is:

```
wlm:2:once:/usr/sbin/wlmcntrl > /dev/console 2>&1 #Start WLM
```

**Note**

Always perform tests in non-production environments.

### 3.3.2.2 Using SMIT

WLM can be started, stopped, updated, or queried by accessing the SMIT *Start/Stop/Update WLM* screen, shown in Figure 40, or by using the following fastpath:

```
# smitty wlmmanage
```



Figure 40. smitty wlmmanage

Under *Start Workload Management*, you will be able to specify the Management Mode (active or passive) if you want WLM to enforce resource set bindings and if WLM is supposed to start now, at the next boot, or both.

Under *Update Workload Management*, you will be asked to specify a superclass name (you can leave this blank if you wish to do a general update) not bound to a specific superclass only. You cannot use the SMIT interface to change the currently-running configuration.

Under *Stop Workload Management*, you are able to stop WLM now, at the next boot, or both.

*Show WLM Status* will give you information about WLM's mode of operation (active, passive, or inactive), as well as whether WLM was started with resource set bindings enforced. It will also display the currently-configured superclasses. Figure 41 shows the WLM status screen in SMIT.

```
COMMAND STATUS
Command: OK          stdout: yes          stderr: no
Before command completion, additional instructions may appear below.
WLM is running in active mode, Rset bindings not active.
Checking classes and rules for 'current' configuration...
System
Default
Shared
db1
devlt
VPs
acctg
db2

F1=Help          F2=Refresh      F3=Cancel      F6=Command
F8=Image        F9=Shell       F10=Exit      /=Find
n=Find Next
```

Figure 41. *Show WLM status screen in SMIT*

### 3.3.2.3 Using WSM

WLM can be controlled from inside the *Overview and Tasks* screen of WSM shown in Figure 42 on page 104.

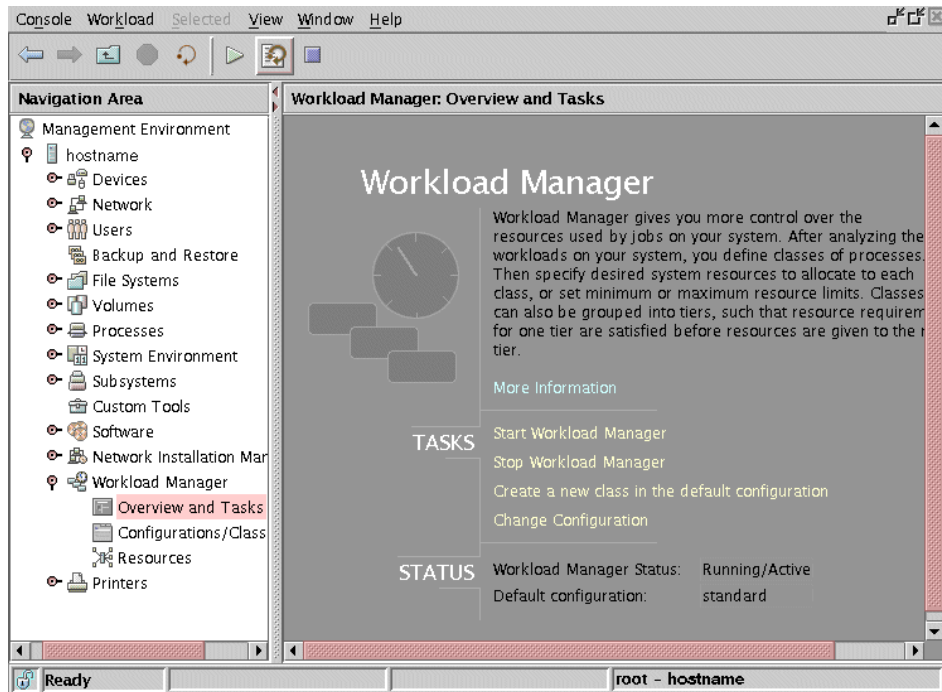


Figure 42. Overview and Tasks screen in WSM

The WLM status and currently-running configuration are shown as you enter the screen. In Figure 42, you can observe that WLM status is Started and Active, and the current configuration is Config. From this screen:

- WLM can be started in active or passive mode, now, at system boot, or both with or without resource set bindings (specifying the chosen configuration) by clicking on *Start Workload Manager*. Figure 43 on page 105 shows the Start Workload Manager screen in WSM.

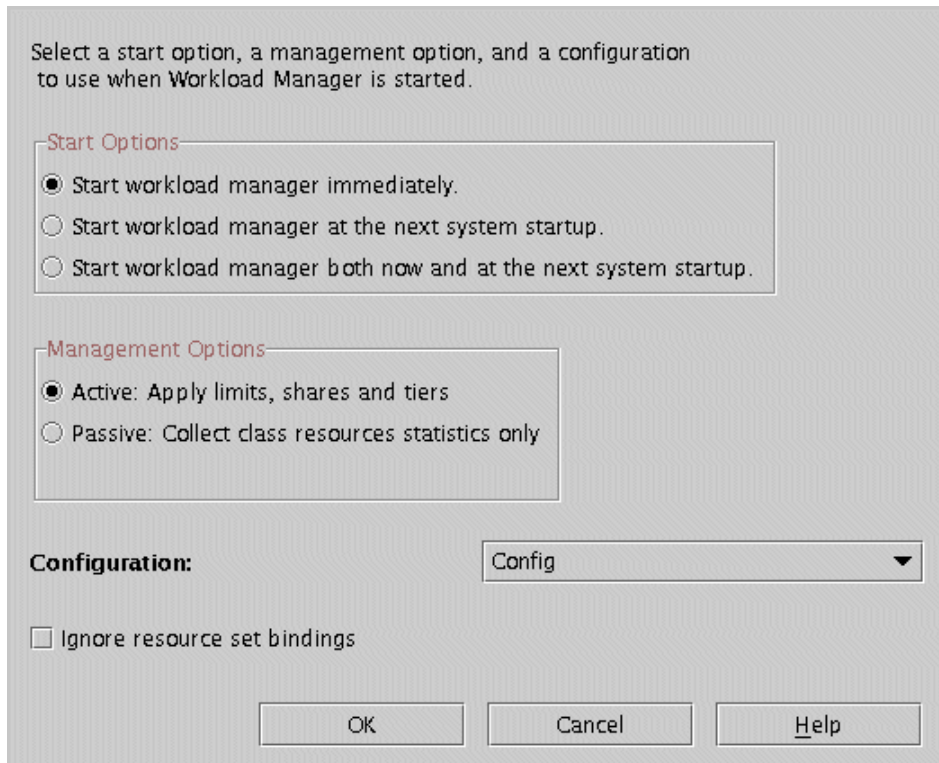


Figure 43. Start Workload Manager screen in WSM

- WLM can be stopped by clicking *Stop Workload Manager* (confirmation is requested). Figure 44 shows the Stop Workload Manager screen in WSM.

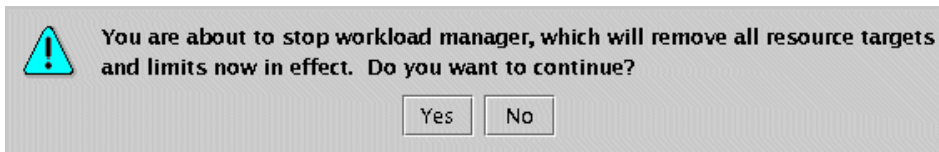


Figure 44. Stop Workload Manager screen in WSM

- A new class for this configuration can be created by clicking *Create a new class in the default configuration* (the class management subject is discussed later).
- The currently running configuration can be modified by clicking *Change Configuration*. The Change Configuration screen appears as shown in Figure 45 on page 106.

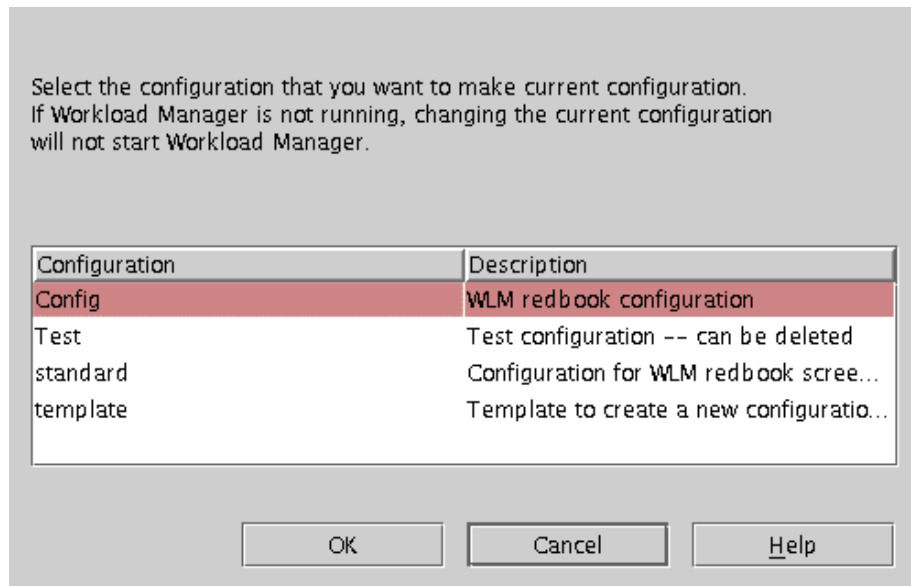




Figure 45. Change Configuration screen in WSM

Alternatively, inside the *Configurations/Classes* screen, some of the icons displayed at the top of the WSM window can be used for WLM management:

-  *Start Workload Manager* (opens screen in Figure 43 on page 105)
-  *Stop Workload Manager* (opens screen in Figure 44 on page 105)

As a third option, WLM can be managed in the *Configurations/Classes* screen by right clicking in a selected configuration and choosing any of the management options, shown in Figure 46 on page 107. In this section, only the options related to WLM management are mentioned. All others are described in later sections.



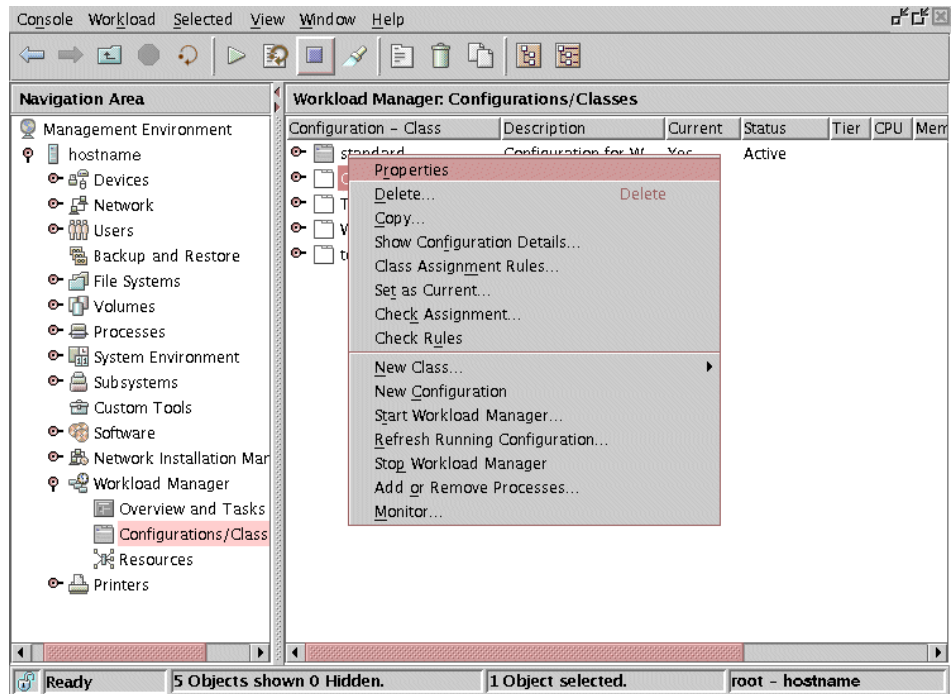


Figure 46. Configuration options in WSM

The WLM management options in the configuration options screen are:

- *Start Workload Manager*
- *Stop Workload Manager*

---

### 3.4 Hints and tips

Practical use of WLM provided a collection of configuration and utilization hints and tips that will help you take better advantage of the feature and avoid some identified problems. Some additional characteristics of WLM will also be pointed out in this section.

#### 3.4.1 Things to do

The following points are some hints that can help you configure and use WLM.

**Before you start**

Always study and anticipate the behaviors of your applications before beginning to use WLM. Know your applications' needs for disk, memory, and CPU use. Otherwise, you could end up giving unnecessary CPU cycles to a memory-bound application, instead of giving it the memory space it really needs.

**A starting point**

Keep it very simple at first, then build. A good starting point for a configuration of WLM would be to create a batch jobs class, an On-line Analytical Processing (OLAP) class, an On-line Transaction Processing (OLTP) class, a backup tasks class, and a Transaction Program class. Depending on the set of applications that are to be run on the system, the OLAP class could take DB2 UDB, or ORACLE; the OLTP class could contain SAP or Baan, and the Transaction Program class could hold MQSeries or Encina, for example. Classifying the processes per function gives the system administrator the ability to more easily decide where to change the configuration and progressively make it meet its original performance objective. This can be done by either gradually partitioning it into additional super or subclasses, or by changing the rules or values of shares and limits. Additionally, it also helps to better determine where the source of a problem might reside. An unclear configuration gets too complex to manage as the number of classes or rules goes up.

**Configuration steps**

When configuring WLM on a server, perform the following steps:

1. Balance the load using only shares at first. Monitor WLM and the system for a reasonable period of time to assess application performance, and tune these values if necessary.
2. Set minimum limits for the applications that do not appear to be given their share of resources.
3. Prioritize workloads using tiers, if necessary, to promote a ranking among jobs. For greater impact, increase the separation of tiers. For example, the impact of a tier 1 and tier 7 separation will be greater than the impact of a tier 1 and tier 4 separation.
4. Set soft or hard maximum limits only if absolutely necessary to control poorly-behaved applications. Remember, a class at its memory maximum limit will cause paging activity even if there are plenty of free memory pages available.

**Note**

WLM configuration should be tested in a non-production environment to avoid possible disruption to users and applications.

**Tiers**

Tiers are used when a high-level of separation of processes' priorities is needed. This happens when there is a defined priority ranking among the applications. Configuration in tiers must be done bearing in mind that the processes assigned to higher numbered tiers will not compete for resources with the processes assigned to lower tiers. If process A from tier 0 has a high number of shares for resources and uses them all (running, for instance, a tight loop), process B from tier 1 might never get any CPU time during the execution of process A. This may be desirable; the system administrator should not allow a backup to stall a heavily-loaded e-business application during regular work hours, for instance. To make sure that the backup eventually happens, the system administrator can take advantage of the ability to have several WLM setups ready to run. He or she can configure `crontab` to change WLM, at a chosen time, into a configuration where the backup process is assigned to tier 0. This way, it will run at a non-disruptive time.

**System and Default superclasses**

For a given program, WLM chooses the first in the list of rules that matches the process' configuration, either by USERID, GROUPID, the name of the executable itself, the type, tag, or any possible combination of these attributes. Because every process is considered to belong to, at least, the *Default* superclass, and because system jobs should not be classified differently than they are in the *System* superclass, you should have the *System* class' rule placed as close to the top of the rules list as possible, and *Default* class' rule should be placed at the very bottom. The only circumstance in which the list of rules may and must have other classes before the *System* class' rule is when the root user is supposed to launch a program that you want placed in a specific, user-created class. There are no reasons why any rule should come after *Default's* rule; it would never be used.

The *System* superclass should not be anywhere other than in tier 0. Placing it on a different tier would ruin the normal functioning of the system. You must not forget that, besides the user applications, kernel processes are being controlled by WLM as well, and if they do not get their share of resources and therefore are not allowed to do their work, nothing else will run properly. Keep this in mind when configuring the values for shares and limits for the *System* superclass. These values should never be so low as to impair the system's

work or so high that they substantially subtract performance from the applications.

### ***Shares versus hard and soft limits***

It is recommended to use resource shares rather than limits to start with. WLM sees resource shares as goals to be achieved, which allows greater system flexibility than imposed limits. If the resource shares set up by a system administrator are not optimal, the system should still be able to balance the load reasonably well. With hard limits set, WLM can do little to prevent applications from being starved of resources. For example, if the maximum memory limit is set smaller than the average working set of the application, significant performance degradation will occur. In summary, it is better to wait to assign limits until after experience has been gained with the results from setting resource shares, and when setting resource limits, start by setting only the minimums. It is also suggested that memory minimums for *all* classes be used before imposing a memory maximum for any class. This is for performance reasons, basically. A class that reaches its maximum memory limit starts paging against itself, which causes the paging algorithm (LRU) to run even when there are plenty of memory pages available. This, by itself, causes some performance impact. The recommended minimum limit for other classes is to make sure that LRU will not steal pages from these classes below those limits, which would cause an even greater performance impact. This would happen if, by some chance, some last and most probable next accessed pages were stolen from a non-minimum limited class.

### ***Rules***

The more specific assignment rules should appear first in the *rules* file, and the more general rules should appear last.

### ***High-availability clustering multiprocessing program (HACMP)***

It is recommended to make the HACMP startup entry in WLM systems as close to the end of the */etc/inittab* file as possible in order to make sure WLM is fully initialized before the cluster manager starts. Otherwise, the deadman switch might trigger a false failover while something, such as WLM, initializes.

### ***WLM on the SP systems***

WLM cannot be used to provide distributed workload management over multiple nodes on the SP systems. Nevertheless, if some nodes are similar in applications structure and configurations, having WLM working in all of them is as easy as performing the following steps:

1. Configure WLM in one node.
2. Use the `tar` command to gather all text files which make up the configuration.

3. Use `dsh` to distribute them to every node applicable.
4. Use the `tar` command to unpack the files.
5. Start up WLM, specifying the configuration files directory.

### 3.4.2 Things to be aware of

The following points are descriptions of some difficulties found:

#### ***WLM memory regulation***

WLM's memory regulation may have a negative impact on page replacement performance. The memory regulation is done by stealing pages, preferably from classes above their memory target. Whenever the page replacement (LRU) is activated, it asks WLM which pages it should steal. All the pages in memory belong to a segment and thus to a class. According to how the actual memory usage of the class compares to its target and limits, each class is given a priority for memory allocation. Classes below their minimum memory limit have the highest priority. Classes between minimum and target have a lower priority, classes above their target and below their maximum limits an even lower priority and classes above their hard maximum memory limit, the lowest possible priority. WLM will thus instruct the LRU that it should steal only pages at or below a given priority.

This means that in order to find the number of pages it needs to free, the LRU will inevitably go through more memory pages than it would if the WLM memory regulation is off (because it will skip all the pages with a higher priority than the one it is looking for).

However, WLM tries to minimize the extra overhead by not sending the LRU looking for a given priority if there is not a sufficient number of pages at or below this priority level.

For customers who wish to use WLM only for CPU regulation, it could be advantageous to use the CPU only mode of WLM to not incur any unwanted page replacement overhead.

The other impact is due to classes above a memory hard limits. When such a class page faults, WLM has no choice than starting the LRU to steal pages from this class, bringing it back below its maximum, before it can give it the new page. This is a performance overhead in two ways:

- First, this may start the LRU in situations where it would normally not run (because there may be plenty of free pages)
- Second, this can be a costly operation, especially if the hard maximum is small, because there are less pages in memory eligible for stealing.

This is the reason why hard memory maximums should be used wisely; that is, do not set a hard maximum limit so low that the class is sure to constantly be hitting this maximum limit.

For further information, please refer to Section 2.8, “WLM interaction with the kernel” on page 36.

### ***svmon***

A problem with the `svmon` command is observed while submitting a heavy memory workload on an 64 bit machine with more than 2 GB memory running AIX Version 4.3.3 system at maintenance level 2 and `perfagent.tools` at the 2.2.33.15 level. `svmon` needs to allocate real memory to work, and being unable to do so, it halts the system. Though this is not a problem directly connected to WLM, it is bound to be observed in WLM environments, so the use of the `svmon` tool is only recommended in WLM systems with the `perfagent.tools` fileset at the 2.2.33.16 level or later.

### ***wlmstat***

On WLM’s first release, when using `tee` and `wlmstat` commands together to monitor performance on the screen and gather the information on a file at the same time, the output of `wlmstat` was not immediate. It only displayed information on the screen or wrote something on the file every 4 KB of data gathered. This problem is solved in AIX 5L.

### ***vmtune***

Unless done with extreme caution, changing some `vmtune` options, such as `minperm`, `maxperm`, `minfree`, and `maxfree` to anything other than default values might impair WLM and degrade system performance. Any potential tuning of these values should be done before using WLM.

#### **Note**

If a problem is experienced with WLM after changing any `vmtune` values, these settings should be moved back to default options.

### ***Non-configured WLM startup***

WLM is not started on AIX by default. Its startup must be issued manually or placed in `/etc/inittab` to be launched upon reboot. The system administrator must make sure, however, that this does not happen before WLM is fully configured and ready to run. A non-configured WLM startup degrades system performance significantly.

### ***Setuid inside applications***

If an application runs a `setuid` while launched to change its *effective* UID, its classification stays related to the UID of the user that originally started it, because no reclassification occurs. A dynamic reclassification only occurs in those cases when the change is made to the process' *real* UID.

The same situation is observed for groups and `setgid`.

#### **Note**

If any undesirable behavior occurs when WLM is running, it can be stopped using the `wlmcntrl -o` command. Stopping WLM will turn off all WLM management of resources, and the system behavior will quickly return to the normal state.

### **3.4.3 LoadLeveler and WLM**

LoadLeveler for AIX is a workload management system for serial and parallel batch jobs running on the RS/6000 SP Systems and IBM @server pSeries. The participating servers in a LoadLeveler environment are also known as nodes, and are grouped in one or more clusters. LoadLeveler manages the jobs across multiple nodes, whereas WLM, which is a part of the AIX operating system, manages the execution of jobs within the node. LoadLeveler for AIX is a Licensed Program Products (LPP). This section discusses the distribution of the workload using LoadLeveler Version 2.2 and AIX WLM.

#### **3.4.3.1 How does LoadLeveler work**

Figure 47 on page 114 is a schematic description of LoadLeveler job handling.

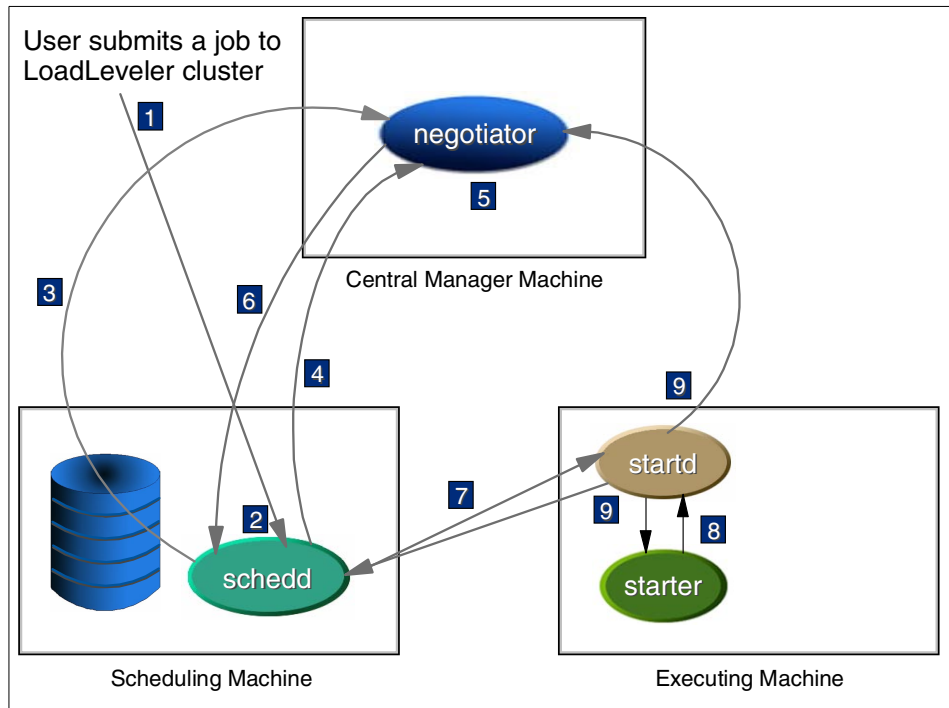


Figure 47. How LoadLeveler works

1. The user submits a job to the LoadLeveler cluster.
2. The job is received by the scheduling machine.
3. The scheduler contacts the negotiator on the central manager to report that a new job is in the queue.
4. The scheduler sends a description of the job to the negotiator.
5. The negotiator decides which nodes the job should execute on, based on the job description defined by the user.
6. The negotiator contacts the scheduler to begin taking steps to run the job.
7. The scheduler contacts the startd daemon on the executing node to run the job.
8. The startd daemon invokes the job on the node.
9. The startd daemon informs the negotiator and the scheduler.



### 3.4.3.2 LoadLeveler functionality

The main characteristics of LoadLeveler functionality are:

- Distributed network-wide job management for job scheduling
- Helps users build, submit, and manage batch jobs
- Used for workload balancing of both serial and parallel jobs

The LoadLeveler administrator can specify:

- The consumable resources to be considered by LoadLeveler's scheduling algorithms
- The quantity of resources available on specific machines
- The quantity of floating resources available on machines in the cluster
- The consumable resources to be considered in determining the priority of executing machines
- The default amount of resources consumed by a job step of a specified job class

The user submitting jobs can specify:

- The resources consumed by each task of a job step

Consumable resources are resources available on machines in your LoadLeveler cluster. They are called *resources* because they model quantities of commodities or services available on machines (that is CPUs, real memory, virtual memory, software licenses, disks, and so forth). They are considered *consumable* because job steps use some specified amount of these commodities when they are running. Once the step is completed, the resource becomes available for reuse by another job step. Consumable resources that model the characteristics of a specific machine (that is its number of CPUs, or the number of specific software licenses available only on that machine) are called machine resources. Consumable resources which model resources that are available across the LoadLeveler cluster (such as software licenses) are called floating resources. For example, consider a configuration with 10 licenses for a given program (which can be used on any machine in the cluster). If these licenses are defined as floating resources, all 10 can be used on one machine, or they can be spread across as many as 10 different machines.

### 3.4.3.3 LoadLeveler and WLM interaction

Using LoadLeveler functionality with WLM does not require changes to the existing configuration of LoadLeveler or WLM. LoadLeveler is classified within WLM like any other application, as shown in Section 8.2, "Customer

experience - WLM and a compute server for research” on page 252. From the status of the cluster, LoadLeveler will determine what physical resources are available on each node in the cluster, and submits the jobs accordingly to the nodes. The class keyword in the local LoadLeveler configuration file on the node describes which class of jobs can be run on that node and how many jobs can be run at a time. Once the job has been submitted to a node for execution, LoadLeveler has no control over the CPU, memory, and I/O subsystem resources in that node, but continues to monitor the execution. The resources used by the jobs in each node may vary depending on the number of jobs running on the node. When multiple jobs are executing on the node, the priority at which a specific job will execute is to be decided by the AIX scheduler. WLM provides the system administrator with greater control over how the scheduler, virtual memory manager, and I/O subsystem allocate resources to process. This is specific to each node.

**Note**

The installed version of LoadLeveler and WLM must match the same level across the cluster.

## Chapter 4. WLM performance tools

This chapter presents tools to monitor and analyze WLM activity. The real time performance tools, such as `wlmstat`, `ps`, `topas`, and `svmon`, are components of the AIX base operating system. System administrators who need a long-term analysis tool and a method to collect trend values should use `wlmpervf`, `xmtrend`, and `jazizo`. They are delivered with the Performance Trend Toolbox feature.

---

### 4.1 wlmstat

To monitor the statistical resource utilization by each superclass and subclass and to display the status of WLM, use the `wlmstat` command. This command shows the contents of WLM data structures that are retrieved from the kernel.

The syntax is:

```
wlmstat [-l class | -t tier] [-S | -s] [-c | -m | -b] [-B device] [-q]
[-T] [-a] [-w] [-v] [interval] [count]
```

Where:

<code>-l class</code>	Indicates the resource utilization for a specific class. If not specified, all classes are displayed.
<code>-t tier</code>	Displays statistics only for the given tier.
<code>-S</code>	Displays statistics for superclasses only.
<code>-s</code>	Displays statistics for subclasses only. If neither <code>-S</code> nor <code>-s</code> are specified, the statistics for both superclasses and subclasses are displayed. In this case, the statistics for each superclass are listed followed by the statistics for the subclasses belonging to that superclass.
<code>-c</code>	Shows only CPU statistics.
<code>-m</code>	Shows only physical memory statistics.
<code>-b</code>	Shows only disk I/O statistics.
<code>-B device</code>	Displays statistics for the given disk I/O device. Statistics for all the disks accessed by the class are displayed by passing an empty string ( <code>-B ""</code> ).
<code>-q</code>	Represses output of status files of last action (quiet).
<code>-T</code>	Returns the total numbers for resource utilization since each class was created (or WLM started). The units are:

- Number of CPU ticks per CPU (seconds) used by each class
- Number of memory pages multiplied by the number of seconds used by each class
- Number of 512 byte blocks sent/received by a class for all the disk devices accessed

- a Delivers absolute figures (relative to the total amount of the resource available to the whole system) for subclasses, with a 0.01 percent resolution. By default, the figures shown for subclasses are a percentage of the amount of the resource used by the superclass, with a one percent resolution. For instance, if a superclass has a CPU target of seven percent and the CPU percentage shown by `wlmstat` without `-a` for a subclass is five percent, `wlmstat` with `-a` will show the CPU percentage for the subclass as 0.35 percent.
- w Displays the memory *high water mark*; that is, the maximum number of pages that a class had in memory since the class was created (or WLM started).
- v Shows most of the attributes concerning the class. The output includes internal parameter values intended for AIX support persons. Table 3 shows a list of some attributes that may be of interest to users.

Table 3. `wlmstat` - selection of internal parameters

Column header	Description
CLASS	Class name
tr	Tier number from 0...9
i	Value of the inheritance attribute; 0 = no, 1 = yes
#pr	Number of processes in the class. If no process is assigned to a class, the following values may not be significant.
CPU	CPU utilization of the class in percent
MEM	Physical memory utilization of the class in percent
DKIO	Disk I/O bandwidth utilization for the class in percent
sha	Number of shares. If no ( "-" ) shares are defined, then sha = -1
min	Resource minimum limit in percent
smx	Resource soft maximum limit in percent

Column header	Description
hmx	Resource hard maximum limit in percent
des	Desired percentage target calculated by WLM using the numbers of the shares in percent
npg	Number of memory pages owned by the class

interval                Specifies an interval in seconds (default to 1).  
count                    Specifies how many times `wlmstat` will print a report (default to 1).

The results of `wlmstat` in the normal (non verbose) case are tabulated with the following fields:

CLASS                Class name  
CPUtotal            CPU time used by the class in percent  
MEM                 Physical memory used by the class in percent  
DKIO                Disk I/O bandwidth used by the class in percent

#### Disk I/O

DKIO is the average of the disk bandwidth on all the disk devices accessed by the class. It is not very significant. For instance, A class uses 80 percent of the bandwidth of one disk and 5 percent of the bandwidth of two other disks. Then the value of DKIO for this class is 30 percent:

$$\frac{(80 \text{ percent (disk1)} + 5 \text{ percent (disk2)} + 5 \text{ percent (disk3)})}{3(\text{number of disks})} = 30 \text{ percent}$$

To achieve a detailed output of the utilization per disk, use the `-B` device option.

### Examples:

To get a printout of current WLM activity, enter:

```
(0) itsosrv1:/# wlmstat -a
      CLASS CPU MEM DKIO
Unclassified  0  0  0
  Unmanaged   0  0  0
    Default   0  0  0
    Shared    0  0  0
    System    0  6  0
      oltp    75 18  0
oltp.Default  68 17  0
oltp.Shared   0  0  0
  oltp.spray  7  1  0
    dss      10 27  0
    backup   13 28  0
(0) itsosrv1:/#
```

To get a report for the superclass *oltp*, enter:

```
(0) itsosrv1:/# wlmstat -a -l oltp
      CLASS CPU MEM DKIO
      oltp  74 17  0
oltp.Default  67 16  0
oltp.Shared   0  0  0
  oltp.spray  7  2  0
(0) itsosrv1:/#
```

To get a report for the subclass *spray* of the superclass *oltp*, updated every 10 seconds for one minute, enter:

```
(127) itsosrv1:/# wlmstat -l oltp.spray 10 6
      CLASS CPU MEM DKIO
oltp.spray  5  1  0
oltp.spray  5  2  0
oltp.spray  7  1  0
oltp.spray  6  2  0
oltp.spray  6  1  0
oltp.spray  5  1  0
(0) itsosrv1:/#
```

To get a detailed CPU report for all classes, enter the information shown in the next screen.

```
(0) itsosrv1:/# wlmstat -c -v
      CLASS tr i #pr CPU sha min smx hwx des rap urap pri
Unclassified 0 0 1 0 -1 0 100 100 100 0 97 10
  Unmanaged 0 0 0 0 -1 0 100 100 0 0 97 10
    Default 0 0 1 0 -1 0 100 100 0 0 97 97
Default.Default 0 0 1 0 1 0 100 100 100 100 48 48
Default.Shared 0 0 0 0 -1 0 100 100 0 0 96 96
  Shared 0 0 0 0 -1 0 100 100 0 0 97 97
Shared.Default 0 0 0 0 1 0 100 100 100 100 48 48
Shared.Shared 0 0 0 0 -1 0 100 100 0 0 96 96
  System 0 0 43 0 10 10 100 100 10 100 0 0
System.Default 0 0 43 0 1 0 100 100 100 100 0 0
System.Shared 0 0 0 0 -1 0 100 100 0 0 48 48
  oltp 0 0 101 77 35 0 100 100 38 -100 194 194
oltp.Default 0 0 -5 71 -1 0 100 100 100 0 144 144
oltp.Shared 0 0 0 0 -1 0 100 100 0 0 144 144
  oltp.spray 0 0 107 6 30 0 100 100 6 -90 187 187
    dss 0 0 3 10 20 0 100 100 22 100 0 0
dss.Default 0 0 2 10 1 0 100 100 100 100 0 0
dss.Shared 0 0 0 0 -1 0 100 100 0 0 48 48
  backup 0 0 2 11 35 0 100 100 38 100 0 0
backup.Default 0 0 3 11 1 0 100 100 100 100 0 0
backup.Shared 0 0 0 0 -1 0 100 100 0 0 48 48
(0) itsosrv1:/#
```

## 4.2 ps

The `ps` command writes the current status of active processes and associated kernel threads to standard output.

Syntax (X/Open Standards):

```
ps [-A] [-N] [-a] [-d] [-e] [-f] [-k] [-l] [-F format] [-o Format]
[-c Clist] [-G Glist] [-g Glist] [-m] [-n NameList] [-p Plist]
[-t Tlist] [-U Ulist] [-u Ulist]
```

In this book, we focus on using `ps` command to view the current status of processes in a single class or set of classes (either subclass or superclass).

Flags:

- a Writes information about all processes to standard output, except the session leaders and processes not associated with a terminal.
- e Writes information about all processes except kernel processes to standard output.
- c Clist Only displays information about processes assigned to the workload management classes listed in the `Clist` variable.

The `clist` variable is either a comma-separated list of class names or a list of class names enclosed in double quotation marks ( " ") and separated from one another by a comma, one or more spaces, or both.

-o `Format`

Displays information in the format specified by the `Format` variable. Multiple field specifiers can be specified for the `Format` variable. It is either a comma-separated list of field specifiers or a list of field specifiers enclosed within a set of " " (double-quotation marks) and separated from one another by a comma, one or more spaces, or both. Each field specifier has a default header. The default header can be overridden by appending an = (equal sign) followed by the user-defined text for the header. The fields are written in the order specified on the command line in column format. The field widths are specified by the system to be at least as wide as the default or user-defined header text. If the header text is null (for example, if `-o user=` is specified) the field width is at least as wide as the default header text. If all header fields are null, no header line is written.

The following field specifiers are recognized by the system and are relevant for use with WLM:

<code>pid</code>	Indicates the decimal value of the process ID. The default header for this field is <code>PID</code> .
<code>user</code>	Indicates the effective user ID of the process. The textual user ID is displayed. If the textual user ID cannot be obtained, a decimal representation is used. The default header for this field is <code>USER</code> .
<code>class</code>	Indicates the workload management class assigned to the process. The default header for this field is <code>CLASS</code> .
<code>pcpu</code>	Indicates the ratio of CPU time used to CPU time available, expressed as a percentage. The default header for this field is <code>%CPU</code> .
<code>tag</code>	Indicates the Workload Manager application tag. The default header for this field is <code>TAG</code> . The tag is a character string up to 30 characters long and may be truncated when displayed by <code>ps</code> . For processes that do not set their tag, this field displays as a hyphen (-).



<code>thcount</code>	Indicates the number of kernel threads owned by the process. The default header for this field is <code>THCNT</code> .
<code>vsz</code>	Indicates, as a decimal integer, the size in kilobytes of the process in virtual memory. The default header for this field is <code>VSZ</code> .
<code>wchan</code>	The event for which the process or kernel thread is waiting or sleeping. For a kernel thread, this field is blank if the kernel thread is running. For a process, the wait channel is defined as the wait channel of the sleeping kernel thread if only one kernel thread is sleeping; otherwise, a star is displayed. The default header for this field is <code>WCHAN</code> .
<code>args</code>	Indicates the full command name being executed. All command-line arguments are included, though truncation may occur. The default header for this field is <code>COMMAND</code> .

To get a detailed report of all classes, enter:

```
ps -ae -o pid,user,class,pcpu,tag,thcount,vsz,wchan,args
```

**Examples:**

To get a simple `ps` output for the superclass *backup* and the subclass *spray* of the superclass *oltp* (*oltp.spray*), enter the information in the following screen:

```
(0) itsosrv1:/# ps -c backup,oltp.spray
  PID   TTY   TIME CMD
 14086 pts/6  0:00 sh
 16490 pts/6  0:00 spray
 17234 pts/6  0:00 spray
 17698 pts/6  0:00 spray
 18928 pts/6  0:00 spray
 19868 pts/6  0:00 spray
 20878 pts/6  0:00 spray
 21108 pts/6  0:00 spray
 21718 pts/1  0:00 ksh
 24124 pts/6  0:00 spray
 25102 pts/6  0:00 spray
 25696 pts/6  0:00 spray
 26286 pts/6  0:00 spray
 26836 pts/6  0:00 spray
 27964 pts/1  0:01 backupserver
 28988 pts/6  0:00 spray
 31850 pts/6  0:00 spray
 32718 pts/6  0:00 spray
 33778 pts/1 110:09 backupserver
 36112 pts/6  0:00 spray
 36414 pts/6  0:00 spray
 38842 pts/6  0:00 spray
 40650   -    0:00
 41310 pts/6  0:00 spray
 41524 pts/6  0:00 spray
 42904 pts/6  0:00 spray
 43854 pts/6  0:00 spray
 45848 pts/6  0:00 spray
 46180 pts/6  0:00 sh
(0) itsosrv1:/#
```

---

### 4.3 topas

The `topas` command reports selected statistics about activity on the local system. It uses the curses library to display its output in a format suitable for viewing on an 80x24 character-based display or in a window of at least the same size in a graphical display.

The `topas` command requires the `perfagent.tools` fileset to be installed on the system.

#### Syntax:

```
topas [-d number_of_hot_disks] [-h show help information]
      [-i monitoring_interval_in_seconds] [-n number_of_hot_network_interfaces]
      [-p number_of_hot_processes] [-w number_of_hot_WLM_classes]
      [-c number_of_hot_CPUs]
```

If the `topas` command is invoked without flags, it runs with its following default flags:

```
topas -d5 -i2 -n2 -p12 -w2 -c1
```

`topas` extracts statistics from the system with an interval specified by the `monitoring_interval_in_seconds` argument.

The following flags can be used when *starting* `topas`.

- d Specifies the maximum number of disks shown. If this number exceeds the number of disks installed, the latter is used. If this argument is omitted, a default of five is assumed. If a value of zero is specified, no disk information is displayed.
- h Displays help information.
- i Sets the monitoring interval in seconds. The default is two seconds.
- n Specifies the maximum number of network interfaces shown. If this number exceeds the number of network interfaces installed, the latter is used. If this argument is omitted, a default of two is assumed. If a value of zero is specified, no network information is displayed.
- p Specifies the maximum number of processes shown. If this argument is omitted, a default of 12 is assumed. If a value of zero is specified, no process information is displayed. Retrieval of process information constitutes the majority of the `topas` overhead. If process information is not required, you should always use this option to specify that you don't want process information.
- w Specifies the maximum number of WLM classes to display. If this number exceeds the number of WLM classes installed, the latter is used. If this argument is omitted, a default of two is assumed. If a value of zero is specified, no WLM class information is displayed.
- c Specifies the maximum number of CPUs to display. If this number exceeds the number of CPUs available, the latter is used. If this argument is omitted, a default of one is assumed. If a value of zero is specified, no CPU information is displayed.

While `topas` is *running*, it accepts one-character subcommands. Each time the monitoring interval elapses, the program checks for one of the following subcommands and responds to the action requested.

- a Show all of the variable sections (network, disk, and process) if screen space allows.
- c Show CPU data. Pressing the `c` key the first time will list the CPUs. Pressing it again will show the totals, and pressing it a third time will turn off this section.
- d Show disk information. If the requested number of disks and the requested number of network interfaces will fit on a 24-line display, both are shown. If there is space left on a 24-line display to list at least three processes, as many processes as will fit are also displayed. Pressing the `d` key the first time will list the disks. Pressing it again will show the totals, and pressing it a third time will turn off this section.
- h Show the same help screen as displayed by the `-h` command line argument.
- n Show network interface information. If the requested number of disks and the requested number of network interfaces will fit on a 24-line display, both are shown. If there is space left on a 24-line display to list at least three processes, as many processes as will fit are also displayed. Pressing the `n` key the first time will list the network adapters. Pressing it again will show the totals, and pressing it a third time will turn off this section.
- w Display WLM classes. Pressing the `w` key will toggle this section on and off.
- W Replace the default display with a WLM classes only display. This display gives more detailed information about WLM classes running on the system than the WLM section of the main display. When the `W` key is pressed again, it toggles back to the default main display.
- p Show process information. If the requested number of processes leaves enough space on a 24-line display to also display the requested number of network interfaces, those are shown. If there is also space to show the requested number of disks, those are shown as well.

- P Replace the default display with a process only display. This display provides more detailed information about processes running on the system than the process section of the main display. When the P key is pressed again, it toggles back to the default main display.
- f Move the cursor over the WLM class and press *Focus* to show the top processes in the group.
- q Quit the program.

The output consists of two fixed parts and a variable section. The top two lines at the left of the display show the name of the system on which `topas` runs, the date and time of the last observation, and the monitoring interval.

The second fixed part fills the rightmost 25 positions of the display. It contains five subsections of statistics, as follows:

### **EVENTS/QUEUES**

Displays the per-second frequency of selected system-global events and the average size of the thread run- and wait queues over the monitoring interval:

Cswitch	The number of context switches
Syscalls	The total number of system calls
Reads	The number of read system calls
Writes	The number of write system calls
Forks	The number of fork system calls
Execs	The number of exec system calls
Runqueue	The average number of threads that were ready to run but were waiting for a processor to become available
Waitqueue	The average number of threads that were waiting for paging to complete

### **FILE/TTY**

Displays the per-second frequency of selected file and tty statistics over the monitoring interval.

Readch	The number of bytes read through the read system call
Writech	The number of bytes written through the write system call
Rawin	The number of raw bytes read from TTYs
Ttyout	The number of bytes written to TTYs

lgets	The number of calls to the inode lookup routines
Namei	The number of calls to the pathname lookup routines
Dirblk	The number of directory blocks scanned by the directory search routine

### **PAGING**

Displays the per-second frequency of paging statistics over the monitoring interval.

Faults	Total number of page faults taken. This includes page faults that do not cause paging activity.
Steals	Physical memory 4K frames stolen by the virtual memory manager.
PgspIn	Number of 4K pages read from paging space.
PgspOut	Number of 4K pages written to paging space.
PageIn	Number of 4K pages read. This includes paging activity associated with reading from file systems. By subtracting PgspIn from this value, you get the number of 4K pages read from file systems.
PageOut	Number of 4K pages written. This includes paging activity associated with writing to file systems. By subtracting PgspOut from this value, you get the number of 4K pages written to file systems.
Sios	The number of I/O requests issued by the virtual memory manager.

### **MEMORY**

Displays the real memory size and the distribution of memory in use.

Real,MB	The size of real memory in megabytes.
% Comp	The percentage of real memory currently allocated to computational page frames. Computational page frames are generally those that are backed by paging space.
% Noncomp	The percentage of real memory currently allocated to non-computational frames. Non-computational page frames are generally those that are backed by file space, either data files, executable files, or shared library files.
% Client	The percentage of real memory currently allocated to cache remotely mounted files.

### **PAGING SPACE**

Displays size and utilization of paging space.

Size,MB	The sum of all paging spaces on the system, in megabytes
% Used	The percentage of total paging space currently in use
% Free	The percentage of total paging space currently free

### **NFS**

Displays NFS status in calls/second:

- Server V2
- Client V2
- Server V3
- Client V3

The variable part of the topas display can have up to five subsections. If more than one appears, they are always shown in the following order:

- CPU
- Network Interfaces
- Physical Disks
- WorkLoad Management Classes
- Processes

### **CPU utilization**

By default, this display shows a bar chart with cumulative CPU usage. If more than one CPU is displayed, a list of CPUs are displayed followed by the cumulative totals across all CPUs on the system, not just what is displayed.

User	This shows the percent of CPU used by programs executing in user mode. (Default sorted by User%)
Kern	This shows the percent of CPU used by programs executing in kernel mode.
Wait	This shows the percent of time spent waiting for I/O.
Idle	This shows the percent of time the CPU(s) is idle.

### **Network Interfaces**

Lists the selected number of network interfaces. The interfaces are ordered after the activity over the monitoring interval. The interface that transferred most bytes (sum of bytes read and written) over the interval is listed first. Sorting is only valid for up to 16 network adapters. For each network interface, the following fields are displayed:

Network	The name of the network interface.
KBPS	The total throughput in megabytes per second over the monitoring interval. This field is the sum of kilobytes received and kilobytes sent per second.
I-Pack	The number of data packets received per second over the monitoring interval.
O-Pack	The number of data packets sent per second over the monitoring interval.
KB-In	The number of kilobytes received per second over the monitoring interval.
KB-Out	The number of kilobytes sent per second over the monitoring interval.

### ***Physical disks***

Lists the selected number of physical disks. The disks are ordered after the activity over the monitoring interval. The interface that was most busy over the interval is listed first. Sorting is only valid for up to 128 disks. For each disk, the following fields are displayed:

Disk	The name of the physical disk.
Busy%	Indicates the percentage of time the physical disk was active (bandwidth utilization for the drive).
KBPS	The number of kilobytes read and written per second over the monitoring interval. This field is the sum of KB-Read and KB-Writ.
TPS	The number of transfers per second that were issued to the physical disk. A transfer is an I/O request to the physical disk. Multiple logical requests can be combined into a single I/O request to the disk. A transfer is of indeterminate size.
KB-Read	The number of kilobytes read per second from the physical disk.
KB-Writ	The number of kilobytes written per second to the physical disk.



### ***WLM Classes***

Workload Management Classes displays the top [number] WLM Classes by default sorted by CPU%.

WLM-Class	The name of the class. The mode in which WLM is running (active or passive) is shown
CPU%	The average CPU utilization of the WLM class over the monitoring interval
Mem%	The average memory utilization of the WLM class over the monitoring interval
Disk-I/O%	The average percent of disk I/O of the WLM class over the monitoring interval

### ***Processes***

Lists the selected number of processes or as many as will fit on the display. The processes are ordered after their CPU usage over the monitoring interval. The process that consumed the most CPU over the interval is listed first. For each process, the following fields are displayed:

Name	The name of the executable program executing in the process. The name is stripped of any pathname and argument information and truncated to nine characters in length.
PID	The process ID of the process.
CPU%	The average CPU utilization of the process over the monitoring interval. The first time a process is shown, this value is the average CPU utilization over the lifetime of the process.
PgSp	The size of the paging space allocated to this process. This can be considered an expression of the footprint of the process but does not include the memory used to keep the executable program and any shared libraries on which it may depend.
Owner	The name of the user that owns the process (only when WLM section is off).
Class	The WLM class to which the process belongs (only when WLM section is on).

## Examples

To run the program with default options, type:

```
topas
```

The result is shown in Figure 48.

Topas Monitor for host: mothra						EVENTS/QUEUES		FILE/TTY							
Wed Sep 6 11:51:11 2000						Interval: 2		Cswitch	21	Readch	0				
								Syscall	25	Writch	50				
Kernel	0.0							Reads	0	Rawin	0				
User	0.2							Writes	0	Ttyout	0				
Wait	0.0							Forks	0	Igets	0				
Idle	99.7	#####				#####		Execs	0	Namei	0				
								Runqueue	0.0	Dirblk	0				
								Waitqueue	1.0						
Network						KBPS	I-Pack	O-Pack	KB-In	KB-Out					
tr0						0.0	0.9	0.4	0.0	0.0					
lo0						0.0	0.0	0.0	0.0	0.0					
								PAGING		MEMORY					
								Faults	0	Real,MB	511				
Disk						Busy%	KBPS	TPS	KB-Read	KB-Writ	Steals	0	% Comp	15.0	
hdisk0						0.0	0.0	0.0	0.0	0.0	PgspIn	0	% Noncomp	5.0	
								PgspOut	0	% Client	0.0				
WLM-Class (Active)						CPU%	Mem%	Disk-I/O%	PageIn		0				
Unmanaged						0	7	0	PageOut		0		PAGING SPACE		
Unclassified						0	8	0	Sios		0		Size,MB		
										0		% Used			
										0		0.9			
Name						PID	CPU%	PgSp	Class	NFS (calls/sec)		% Free		99.0	
topas						11902	0.2	0.8	System	ServerV2		0			
syncd						2706	0.0	0.1	System	ClientV2		0		Press:	
gil						1806	0.0	0.0	System	ServerV3		0		"h" for help	
init						1	0.0	0.8	System	ClientV3		0		"q" to quit	
snmpd						5942	0.0	0.7	System						

Figure 48. topas - example 1

To display five hot disks every five seconds and omit network interface and process information, type:

```
topas -i5 -d5 -n0 -p0
```

The result is shown in Figure 49 on page 133.

```

Topas Monitor for host:  itsosrv1          EVENTS/QUEUES  FILE/TTY
Thu Sep  7 12:29:38 2000  Interval:  5          Cswitch      24  Readch      422
                                                                Syscall      29  Writech      9
Kernel    0.0          |                               Reads        5  Rawin       0
User      0.0          |                               Writes       0  Ttyout      0
Wait     0.0          |                               Forks        0  Igets       0
Idle     99.9         |#####|                               Execs        0  Namei       1
                                                                Runqueue    0.0  Dirblk      0
                                                                Waitqueue   2.0

Disk   Busy%   KBPS   TPS  KB-Read  KB-Writ
hdisk2  0.0    0.0    0.0    0.0    0.0
hdisk0  0.0    0.0    0.0    0.0    0.0
hdisk3  0.0    0.0    0.0    0.0    0.0
hdisk6  0.0    0.0    0.0    0.0    0.0
hdisk1  0.0    0.0    0.0    0.0    0.0

WLM-Class (Active)   CPU%   Mem%   Disk-I/O%
Unclassified         0      0      0
Unmanaged            0      0      0

                                                                PAGING
                                                                Faults      0  Real,MB    1023
                                                                Steals      0  % Comp     10.0
                                                                PgspIn     0  % Noncomp  3.0
                                                                PgspOut    0  % Client   0.0
                                                                PageIn     0
                                                                PageOut    0  PAGING SPACE
                                                                Sios       0  Size,MB    0
                                                                % Used     0.6
                                                                NFS (calls/sec) % Free    99.3
                                                                ServerV2   0
                                                                ClientV2   0  Press:
                                                                ServerV3   0  "h" for help
                                                                ClientV3   0  "q" to quit

```

Figure 49. topas - example 2

To display the five most active processes and neither network nor disk information, type:

```
topas -p5 -n0 -d0
```

The result is shown in Figure 50 on page 134.

```

Topas Monitor for host:  mothra          EVENTS/QUEUES  FILE/TTY
Wed Sep  6 11:57:27 2000  Interval:  2          Cswitch      25  Readch      0
                               Syscall      107 Writech     1925
Kernel    0.2  |                               Reads         0  Rawin       0
User      0.0  |                               Writes        30  Ttyout      0
Wait      0.0  |                               Forks         0  Igets       0
Idle      99.7  |#####|                               Execs         0  Namei       8
                               Runqueue     0.0  Dirblk      0
WLM-Class (Active)  CPU%   Mem%  Disk-I/O%  Waitqueue  1.0
Unmanaged           0     7     0
Unclassified       0     8     0
PAGING             MEMORY
Name               PID CPU% PgSp Class  Faults  0  Real,MB  511
topas              11904 0.3  0.8 System Steals  0  % Comp   15.0
gil                1806 0.0  0.0 System Pgspln  0  % Noncomp 5.0
syncd              2706 0.0  0.1 System Pgspln  0  % Client  0.0
ksh                12144 0.0  0.2 System PageIn   0
telnetd            12448 0.0  0.5 System PageOut  0  PAGING SPACE
                               Sios       0  Size,MB  0
                               % Used    0.9
                               NFS (calls/sec) % Free  99.0
ServerV2           0
ClientV2           0  Press:
ServerV3           0  "h" for help
ClientV3           0  "q" to quit

```

Figure 50. topas - example 3

To see detailed information about the defined WLM classes running on the system, use the subcommand, W, when topas is running as shown in Figure 51 on page 135.

```

Topas Monitor for host:  mothra      Interval:  2   Wed Sep  6 11:55:21 2000
WLM-Class (Active)      CPU%      Mem%      Disk-I/O%
System                  0         4         0
Shared                  0         2         0
Default                  0         0         0
Unmanaged                0         7         0
Unclassified            0         8         0

[]

=====
          DATA TEXT PAGE          PGFAULTS
USER      PID PPID PRI NI  RES  RES SPACE  TIME CPU% I/O  OTH COMMAND
root     11902 12144 108 20  250   12  230  0:03 0.5  0   0 topas
root      1032   0  16 41    3 3775    3  0:00 0.0  0   0 lrud
root      1290   0  60 41    4 3775    4  0:00 0.0  0   0 xmgc
root      1548   0  36 41    4 3775    4  0:00 0.0  0   0 netm
root      1806   0  37 41   16 3775   16  0:02 0.0  0   0 gil
root      2064   0  16 41    4 3775    4  0:00 0.0  0   0 wlmsched
root      2706   1 108 20   42    1   37  0:01 0.0  0   0 syncd
root      3196  5680 108 20  210   11  202  0:00 0.0  0   0 portmap
root      3362   0 108 20    4 3775    4  0:00 0.0  0   0 lvmbb
root      3666   1 108 20  134   23  124  0:00 0.0  0   0 errdemon
root      3910  5680 108 20   96    8   87  0:00 0.0  0   0 syslogd

```

Figure 51. topas - example 4

#### 4.4 svmon

This tool generates snapshots of a system's virtual memory. It has been enhanced with usability, scalability, and speed improvements on the largest enterprise server systems. In addition, the `svmon` tool was enhanced to generate reports on users, commands, and WLM classes to support WLM functions.

The `svmon` command requires the `perfagent.tools` files set to be installed on the system.

The `svmon` command displays information about the current state of memory. The displayed information does not constitute a true snapshot of memory because the `svmon` command runs at the user level with interrupts enabled. The segment is the basic object used to report memory consumption. A segment is a set of pages, so the statistics reported by `svmon` are expressed in

terms of pages. A page is a 4K block of virtual memory while a frame is a 4K block of real memory. Unless otherwise noted, all statistics are in units of 4096-bytes of memory pages.

The memory consumption is reported using the `inuse`, `free`, `pin`, `virtual` and `paging space` counters.

- The **inuse** counter represents the number of used frames.
- The **free** counter represents the number of free frames from all memory pools.
- The **pin** counter represents the number of pinned frames, that is frames that cannot be swapped.
- The **virtual** counter represents the number of pages allocated in the system virtual space.
- The **paging space** counter represents the number of pages reserved or used on paging spaces.

A segment can be used by multiple processes. Each page from such a segment is accounted for in the `inuse`, `pin`, `virtual`, or `pgspace` fields for each process that uses the segment. Therefore, the total of the `inuse`, `pin`, `virtual`, and `pgspace` fields over all active processes may exceed the total number of pages in memory or on paging space.

VMM manages virtual page counters for statistical purpose only, which means they are not always up-to-date, and their values may be less than the corresponding `inuse` counters.

A segment belongs to one of the following types; persistent, working, client, mapping, and real memory mapping.

- **Persistent** segments are used to manipulate files and directories.
- **Working** segments are used to implement the data areas of processes and shared memory segments.
- **Client** segments are used to implement some virtual file systems, such as the Network File System (NFS) and the CD-ROM file system.
- **Mapping** segments are used to implement the mapping of files in memory.
- **Real memory mapping** segments are used to access the I/O space from the virtual address space.

The `svmon` command can create nine types of reports:

- Global
- User
- Command
- Class

- Tier
- Process
- Segment
- Detailed segment
- Frame

This book focus on describing only the workload management reports, *class* and *tier*. These reports are available when the workload manager is running. Otherwise, the message "WLM must be started" is displayed, and no statistics are reported. When the workload manager is running in passive mode, `svmon` will display the message, "WLM is running in passive mode", before displaying the statistics.

**Note**

WLM provides dynamic reclassification of processes and their segments. At each iteration, `svmon` uses a snapshot of the class configuration by using the `wlm_get_info` system call and accesses the related processes and segments. The `segstat_tbl`, `process_tbl`, and `wlm_tbl` are freed at the end of each iteration. Consequently, `svmon` is able to see the changes. Also, if a class disappears, `svmon` reports a message without any error.

Problems may appear when a segment or process is loaded in the `svmon` private data base with a given class ID associated to a given classname and the class ID, or the classname changes before the real analysis of the segment or process. Then `svmon` can report inaccurate statistics.

#### 4.4.1 Workload manager class report

There are two types of classes; *superclasses* and *subclasses*. Superclass names are up to 16 characters long and cannot contain a period. Subclass names start with their superclass name followed by a period and subclass part, which can be up to 16 characters long and cannot contain a period. The total number of superclasses that can be defined is limited to 27. The total number of subclasses that can be defined for a superclass is 10.

Superclasses and subclasses will be treated identically. When a superclass is passed as an argument, `svmon` reports all the segments belonging to all the subclasses of the superclass without giving subclass statistics.

The class report is printed when `-w` is specified.

## Syntax:

```
svmon -W [clnm1...clnmN] [-e] [-k] [-r] [-n | -s] [-w | -f | -c] [-tCount]
[-u | -p | -g | -v] [-iInterval [NumIntervals]] [-l] [-d] [-z] [-m]
```

The following flags can be specified:

- e Shows the statistics of the subclasses of the class and reports the segments statistics per subclass. In this case, the class parameter must be a superclass name.
- k When `-k` is specified, `svmon` reports statistics using a process point of view. There will be no change to this option except when `-e` is specified. Then the segments of each subclass will be split into three categories; system, exclusive, and shared.
- r If the `-r` flag is specified, each segment is followed by the range(s), within the segment, where pages have been allocated.
- n Indicates that only non-system segments are to be included in the statistics. By default, all segments are analyzed.
- s Indicates that only system segments are to be included in the statistics. By default, all segments are analyzed.
- w Indicates that only working segments are to be included in the statistics. By default, all segments are analyzed.
- f Indicates that only persistent segments (files) are to be included in the statistics. By default, all segments are analyzed.
- c Indicates that only client segments are to be included in the statistics. By default, all segments are analyzed.
- tCount Displays memory usage statistics for the top Count object to be printed.
- u Indicates that the objects to be printed are sorted in decreasing order by the total number of pages in real memory. It is the default sorting criteria if none of the following flags are present; `-p`, `-g`, or `-v`.
- p Indicates that the objects to be printed are sorted in decreasing order by the total number of pages pinned.
- g Indicates that the objects to be printed are sorted in decreasing order by the total number of pages reserved or



used on paging space. This flag, in conjunction with the segment, reports non-working segments at the end of the sorted list.

- v Indicates that the objects to be printed are sorted in decreasing order by the total number of pages in virtual space. This flag, in conjunction with the segment report, shifts the non-working segments to the end of the sorted list.
  
- iInterval  
[NumInterval] Instructs the `svmon` command to print statistics out repeatedly. Statistics are collected and printed every [Interval] seconds. NumIntervals is the number of repetitions; if not specified, `svmon` runs until user interruption (Ctrl-C).
  
- l Shows, for each displayed segment, the list of process identifiers that use the segment and, according to the type of report, the entity name (login, command, or class) to which the process belongs. For special segments, a label is displayed instead of the list of process identifiers.  
*System segment:*  
This label is displayed for segments that are flagged *system*.  
*Unused segment:*  
This label is displayed for segments that are not used by any existing processes.  
*Shared library text:*  
This label is displayed for segments that contain text in a shared library and that can be used by most of the processes (libc.a). This is to prevent the display of a long list of processes.
  
- d Displays, for a given entity, the memory statistics of the processes belonging to the entity.
  
- z Displays the maximum memory size dynamically allocated (malloc) by `svmon` during its execution.
  
- m Displays information about source segment rather than a mapping segment when a segment is mapping a source segment.

The column headings in a class report are:

Class or Superclass	Indicates the class or superclass name
---------------------	--

Inuse	Indicates the total number of pages in real memory from segments belonging to the class
Pin	Indicates the total number of pages pinned from segments belonging to the class
Pgsp	Indicates the total number of pages reserved or used on paging space by segments belonging to the class
Virtual	Indicates the total number of pages allocated in the virtual space of the class

After these statistics are displayed, `svmon` displays information about the segments belonging to the class.

**Examples:**

To print out the memory usage statistics for the superclass, *backup*, enter the information shown in the following screen:

```
((0)itsosrv1:/# svmon -W backup
```

```
=====
Superclass          Inuse  Pin   Pgspace  Virtual
backup              52833  10    0        50329

  Vsid  Esid  Type  Description          Inuse  Pin  Pgspace  Virtual
  6784  -    work                27989  0    0  28017
  1aa18  -    work                21887  0    0  21887
  14356  -    pers  /dev/lv_wlm1:17     1250   0    -    -
  173f5  -    pers  /dev/lv_wlm2:17     1250   0    -    -
  5347   -    work                103    2    0    101
  c34e   -    work                77     0    0    77
  1891a  -    work                77     0    0    77
  14636  -    work                46     0    0    37
  5327   -    work                28     0    0    20
  1d83f  -    work                16     0    0    18
  1e33c  -    work                16     0    0    13
  10772  -    work                15     0    0    13
  6a84   -    work                15     0    0    13
  15457  -    work                14     0    0    14
  38a1   -    work                8      0    0    8
  126f0  -    work                8      0    0    8
  11313  -    pers  /dev/hd1:26         6      0    -    -
  e50c   -    work                5      2    0    5
  b549   -    work                5      2    0    5
  12e3   -    work                3      2    0    3
  13351  -    work                3      0    0    3
  14a16  -    work                3      0    0    0
  12970  -    work                3      0    0    5
  6904   -    work                2      2    0    2
  a9c8   -    work                1      0    0    3
  2320   -    pers  /dev/hd1:32         1      0    -    -
  1d39f  -    pers  /dev/hd2:16870     1      0    -    -
  834a   -    pers  /dev/hd1:23         1      0    -    -
```

To print out the memory usage statistics for the subclass *spray*, enter the information shown in the following screen.

```
(0) itsosrv1:/# svmon -W oltp.spray
```

```
=====  
Class                               Inuse Pin  Pgspace Virtual  
oltp.spray                          852  20    0      944  
  
Vsid   Esid  Type  Description          Inuse  Pin  Pgspace  Virtual  
d96f   -    work                77     2    0        77  
c6ae   -    work                76     2    0        76  
da0f   -    work                76     2    0        76  
98eb   -    work                75     2    0        75  
d5cf   -    work                75     2    0        75  
12350  -    work                75     2    0        75  
1dalF  -    work                75     2    0        75  
e9ac   -    work                34     0    0        22  
1f9bd  -    work                34     0    0        22  
4786   -    work                34     0    0        22  
147d6  -    work                33     0    0        22  
23c0   -    work                33     0    0        22  
13771  -    work                33     0    0        22  
5e2    -    work                33     0    0        22  
44c6   -    work                29     0    0        21  
862a   -    work                20     2    0        80  
e9cc   -    work                20     2    0        80  
d8af   -    work                20     2    0        80=
```

To print out the memory usage for the superclass *oltp* with its subclasses, enter the information shown in the following screen.

```
(0) itsosrv1:/# svmon -W oltp -e
```

```
=====
Superclass                               Inuse   Pin    Pgps  Virtual
oltp                                       35941   26     0    35934
=====
Class                                     Inuse   Pin    Pgps  Virtual
oltp.Default                             35895   24     0    35899
=====
      Vsid   Esid Type Description                               Inuse   Pin Pgps Virtual
      a4c8   - work                                     33198   0   0  33206
      17995  - work                                     1493    0   0  1533
      33e1   - work                                     194     0   0   195
      782    - work                                     189     0   0   189
      92eb   - work                                     103     2   0   101
      1d81f  - work                                     88      0   0   88
      1e99c  - work                                     84      0   0   84
      e6ec   - work                                     74      0   0   74
      10712  - work                                     57      0   0   48
      c2ee   - work                                     53      0   0   53
      e2ec   - work                                     28      0   0   20
      98ab   - work                                     22      0   0   22
      3541   - work                                     18      0   0   20
      126b0  - work                                     17      0   0   17
      152f7  - work                                     13      0   0   11
      c38e   - work                                     8       0   0    8
      72e5   - pers /dev/hdl:25                          6       0   -    -
      e58c   - work                                     4       2   0    4
      1971b  - work                                     3       0   0    0
      b8a9   - work                                     2       2   0    2
      b2e9   - pers /dev/hdl:19                          1       0   -    -
      f2ed   - pers /dev/hdl:28                          1       0   -    -
      1e4bc  - work                                     1       0   0    3
=====
Class                                     Inuse   Pin    Pgps  Virtual
oltp.spray                                46      2     0    35
=====
      Vsid   Esid Type Description                               Inuse   Pin Pgps Virtual
      848a   - work                                     76      2   0    76
      1d8bf  - work                                     58      2   0    58
      1d75f  - work                                     33      0   0    22
      1d85f  - work                                     29      0   0    21
      17535  - work                                     20      2   0    80
      d7ef   - work                                     20      2   0    80
      157b7  - work                                     16      0   0    11
=====
Class                                     Inuse   Pin    Pgps  Virtual
oltp.Shared                                0       0     0     0
=====
```

To print out statistics using a process of view for each subclass of the superclass *oltp*, enter:

```
(0)itsostrvl:/# svmon -W oltp -e -k
```

```
=====
```

Superclass	Inuse	Pin	Pgsp	Virtual
oltp	21432	3670	1584	23238

```
=====
```

Class	Inuse	Pin	Pgsp	Virtual
oltp.Default	16340	1929	792	17146

```
.....
```

SYSTEM segments	Inuse	Pin	Pgsp	Virtual
	4209	1759	792	3151

```
.....
```

Vsid	Esid	Type	Description	Inuse	Pin	Pgsp	Virtual
0	0	work	kernel seg	3939	1735	792	2881
62e4	-	work		270	24	0	270

```
.....
```

EXCLUSIVE segments	Inuse	Pin	Pgsp	Virtual
	11108	170	0	10914

```
.....
```

Vsid	Esid	Type	Description	Inuse	Pin	Pgsp	Virtual
135f1	3	work	shmat/mmap	9	0	0	9
902	2	work	process private	4	2	0	4
eaec	f	work	shared library data	1	0	0	1

```
.....
```

SHARED segments	Inuse	Pin	Pgsp	Virtual
	1010	0	0	3070

```
.....
```

Vsid	Esid	Type	Description	Inuse	Pin	Pgsp	Virtual
16834	f	work	shared library data	29	0	0	21
14094	-	pers	/dev/hd2:16981	2	0	-	-

```
=====
```

Class	Inuse	Pin	Pgsp	Virtual
oltp.spray	5092	1741	792	6092

```
.....
```

SYSTEM segments	Inuse	Pin	Pgsp	Virtual
	3939	1735	792	2881

```
.....
```

Vsid	Esid	Type	Description	Inuse	Pin	Pgsp	Virtual
0	0	work	kernel seg	3939	1735	792	2881

```
.....
```

EXCLUSIVE segments	Inuse	Pin	Pgsp	Virtual
	167	4	0	156

```
.....
```

Vsid	Esid	Type	Description	Inuse	Pin	Pgsp	Virtual
6724	2	work	process private	58	2	0	58
6904	f	work	shared library data	33	0	0	22

```
.....
```

SHARED segments	Inuse	Pin	Pgsp	Virtual
	947	0	0	3027

```
.....
```

Vsid	Esid	Type	Description	Inuse	Pin	Pgsp	Virtual
c02c	d	work	shared library text	869	0	0	3027
e0ae	1	pers	code, /dev/hd2:4205	57	0	-	-

```
=====
```

Class	Inuse	Pin	Pgsp	Virtual
oltp.Shared	0	0	0	0

## 4.4.2 Workload manager tier report

The tier value for a superclass is the position of the class in the hierarchy of resource limitation desirability. The tier value for a subclass is the position of the subclass in the hierarchy of resource limitation desirability.

The tier report is printed when `-T` is specified.

Syntax:

```
svmon -T [tier1...tierN] [-a supclnm] [-x] [-e] [-r] [-u | -p | -g | -v]
[-n | -s] [-w | -f | -c] [-t Count] [-iInterval[NumIntervals]] [-l] [-z] [-m]
```

The following flags can be specified:

<code>-a</code>	Applies a tier to a superclass.
<code>-x</code>	Displays information about the segments belonging to each class.
<code>-e</code>	Reports the statistics of the subclasses of each superclass belonging to the tier.
<code>-r</code>	If the <code>-r</code> flag is specified, each segment is followed by the range(s), within the segment, where pages have been allocated.
<code>-l</code>	If the <code>-l</code> flag is specified, each segment is followed by the list of process identifiers that are using it. Besides the process identifier, the tier number and class that the process belongs to are also displayed.

### Note

`-e` is only allowed with `-T` and `-w`  
`-x` is only allowed with `-T`  
`-r` or `-l` is only allowed with `-T` if `-x` is specified

The column headings in a tier report are:

Tier	Indicates the tier number.
Superclass	Optional column heading. Indicates the superclass name when tier applies to a superclass (when the <code>-a</code> flag is used).
Inuse	Indicates the total number of pages in real memory from segments belonging to the tier.

Pin	Indicates the total number of pages pinned from segments belonging to the tier.
Pgsp	Indicates the total number of pages reserved or used on paging space by segments belonging to the tier.
Virtual	Indicates the total number of pages allocated in the virtual space of the tier.

After these statistics are displayed, `svmon` displays information about the classes belonging to the tier.

**Examples:**

To print out the memory usage for all defined tiers, enter the information shown in the following screen:

```
(0) itsosrv1:/# svmon -T
=====
Tier                Inuse      Pin      Pgsp  Virtual
  0                234012    10687    1498  195497
=====
Superclass          Inuse      Pin      Pgsp  Virtual
backup              67746      10        0    65721
dss                 64771       8        0    64799
oltp                42123      182       0    41726
Unclassified        31181       26        0     126
System              26744     10459     1498  19158
Shared              1207        0         0     3760
Default             240         2         0     207
Unmanaged           0           0         0         0
```

To print out the memory usage for the tier 0, enter the information shown in the following screen.



```
(0) itsosrv1:/# svmon -T
```

```
=====
Tier                                Inuse    Pin    Pgspace  Virtual
0                                    234012  10687  1498    195497
=====
Superclass                          Inuse    Pin    Pgspace  Virtual
backup                              67746   10     0        65721
dss                                  64771    8     0        64799
oltp                                 42123   182    0        41726
Unclassified                        31181   26     0         126
System                              26744  10459  1498    19158
Shared                               1207    0     0         3760
Default                              240     2     0         207
Unmanaged                            0       0     0          0
=====
```

To print out the memory usage for all tier subclasses of the superclass *oltp*, enter the following:

```
(0) itsosrv1:/# svmon -T -a oltp
```

```
=====
Tier                                Inuse    Pin    Pgspace  Virtual
0 oltp                              35677   18     0        35651
=====
Class                                Inuse    Pin    Pgspace  Virtual
oltp.Default                        35677   18     0        35651
oltp.Shared                          0       0     0          0
=====
Tier                                Inuse    Pin    Pgspace  Virtual
1 oltp                              524     22     0         656
=====
Class                                Inuse    Pin    Pgspace  Virtual
oltp.spray                          524     22     0         656
=====
```

To print out the memory usage for the tier 0, including the subclass statistics, enter the information shown in the following screen.

```
(0) itsosrv1:/# svmon -T 0 -e
```

```
=====
```

Tier	Inuse	Pin	Pgsp	Virtual
0	228018	10511	1372	189497

```
=====
```

Superclass	Inuse	Pin	Pgsp	Virtual
backup	68169	8	0	65673
dss	65995	6	0	66025

```
=====
```

Superclass	Inuse	Pin	Pgsp	Virtual
oltp	34587	20	0	34540

```
=====
```

Class	Inuse	Pin	Pgsp	Virtual
oltp.Default	34587	20	0	34540
oltp.Shared	0	0	0	0
oltp.spray	0	0	0	0
Unclassified	31181	26	0	116
System	26639	10449	1372	19176
Shared	1207	0	0	3760
Default	240	2	0	207
Unmanaged	0	0	0	0

```
=====
```

To print out the memory usage for the subclasses in tier 0 of the superclass *oltp*, including the segment statistics and the list of process identifiers, enter the information shown in the following screen.

```
(0) itsosrv1:/# svmon -T 0 -a oltp -x -l
```

```
=====
```

Tier	Inuse	Pin	Pgsp	Virtual
0 oltp	36063	28	0	36010

```
=====
```

Class	Inuse	Pin	Pgsp	Virtual
oltp.Default	36063	28	0	36010

```
=====
```

Vsid	Esid	Type	Description	Inuse	Pin	Pgsp	Virtual
44c6	f	work shared library data		13	0	0	11
			pid:23798 tier: 1 class:oltp.spray				
f4ad	-	work		13	0	0	11
			Unused segment				
e32c	f	work shared library data		13	0	0	11
			pid:36314 tier: 0 class:oltp.Default				
7e2	-	work		13	0	0	11
			Unused segment				
402	f	work shared library data		13	0	0	11
			pid:31514 tier: 0 class:oltp.Default				
1f4bd	-	work		13	0	0	11
			Unused segment				
13a31	f	work shared library data		13	0	0	11
			pid:29392 tier: 1 class:oltp.spray				
168f4	-	work		12	0	0	12
			Unused segment				
152f7	f	work shared library data		12	0	0	10
			pid:18794 tier: 0 class:oltp.Default				
1993b	2	work process private		11	2	0	11
			pid:33954 tier: 1 class:oltp.spray				
72e5	1	pers code, /dev/hdl:25		6	0	-	-
			pid:37556 tier: 0 class:oltp.Default				
			pid:31514 tier: 0 class:oltp.Default				
			pid:29392 tier: 1 class:oltp.spray				
			pid:23156 tier: 0 class:oltp.Default				
			pid:22092 tier: 1 class:oltp.spray				
			pid:18794 tier: 0 class:oltp.Default				
			pid:18496 tier: 1 class:oltp.spray				
c2ee	3	work shmat/mmap		6	0	0	6
D2ef	2	work process private		5	2	0	5
			pid:18794 tier: 0 class:oltp.Default				
15837	2	work process private		4	2	0	4
			pid:31514 tier: 0 class:oltp.Default				
ea8c	-	work		4	0	0	4
			Unused segment				
16754	-	work		4	0	0	4
			Unused segment				
178d5	2	work process private		4	2	0	4
			pid:29392 tier: 1 class:oltp.spray				
9aeb	2	work process private		4	2	0	4
			pid:37556 tier: 0 class:oltp.Default				

```
=====
```

Class	Inuse	Pin	Pgsp	Virtual
oltp.Shared	0	0	0	0

## 4.5 Web-based System Manager (WSM)

Apart from being a graphical user interface to configure WLM, Web-based System Manager (WSM) provides some monitoring tools to analyze and manipulate resource usage on a per-resource and per-class basis, and view the allocation of processes to classes. WSM filesets are shipped with the Base Operating System, and the tool is launched with the AIX command `wsm`.

The resource-based monitoring screens are accessible under the *Resources* view. When WLM is started, this option displays a view of the managed resources in the current configuration, and their current resource usage as shown in Figure 52.

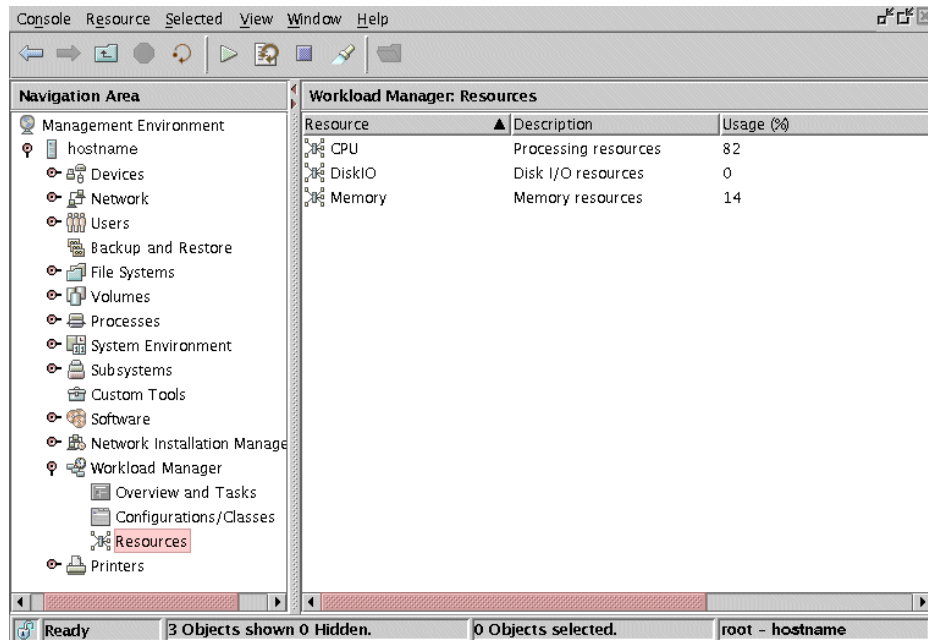


Figure 52. Resources screen in WSM

By double clicking any of the resources, its utilization on a per class basis is displayed. For instance, a sample output of memory usage by class could be the one shown in Figure 53 on page 151.

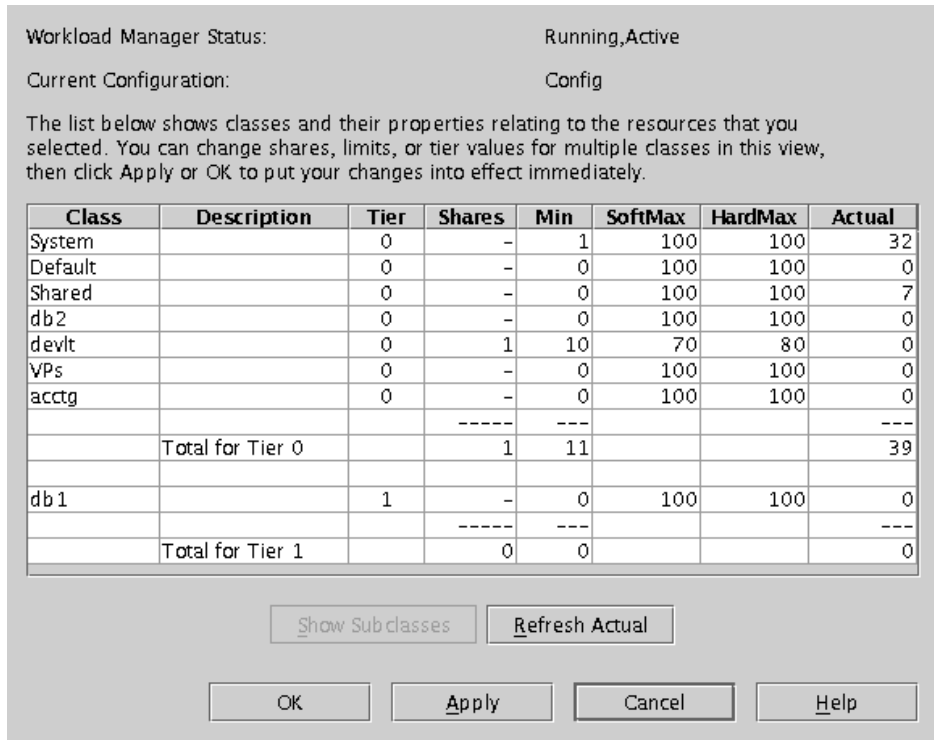


Figure 53. Memory usage by class

This screen also allows the system administrator to directly edit and modify the values in the different fields by clicking on the field whose value should be changed. After changing one or more of the values, the administrator clicks on *Apply*, waits a few minutes for the new settings to take effect, then clicks on *Refresh Actual* to see the updated actual usage. If the new usage numbers are not satisfactory, the administrator can repeat the process.

From this screen, the system administrator can also choose to monitor and manipulate resource utilization at the subclass level. For that purpose, the superclass whose subclasses are to be analyzed must be highlighted, and the *Show Subclasses* option must be chosen. The output is similar to that shown in Figure 53.

It is also possible in WSM to observe the processes classification on a per-class basis. In the *Configurations/Classes* view, by right-clicking the name of a class in a configuration tree, you get access to the classes options. One of them is *Show Processes*, which launches a view of the allocated

processes to the specified class. An example of the output of this option for a class with the /usr/bin/vi process in it can be seen in Figure 54.

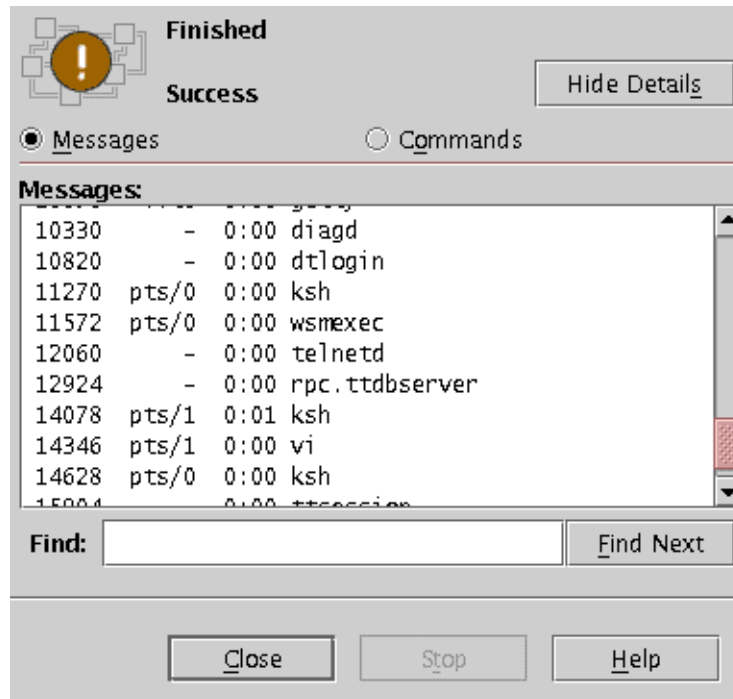


Figure 54. Show processes in WSM

---

## 4.6 Monitoring Workload Manager with PTX

Performance Toolbox (PTX) for AIX provides a high-level graphical user interface for monitoring a wide variety of system resources. It can be used to view and analyze AIX WLM information. These interfaces allow the user to monitor the behavior of a WLM configuration, analyze trends, and record activities.

A new parent context name, WLM, is added to the System Performance Measurement Interface (SPMI). In PTX, the SPMI is an application programming interface (API) that provides standardized access to local system resource statistics. By developing SPMI application programs, a user can retrieve information about system performance with minimum system overhead. For each WLM class, it includes metrics and associated properties (min, soft max, hard max, target, and actual usage). Any metric available via the SPMI can be processed by the PTX agents, recorded, filtered, and viewed

by local or remote PTX clients. There are no design limitations on the SPMI for two reasons:

- WLM already collects most of the data needed to provide performance monitoring support.
- An API exists to retrieve data.

#### 4.6.1 xmperf

xmperf is one of primary Performance Toolbox Manager user interfaces. This tool is used to monitor any metric on local or remote systems. The interface is composed of a set of instruments, with each instrument containing one or more metrics. Instruments can be displayed in a variety of styles (including lines, bars, pie charts, and speedometers). Each set of metrics can be displayed at sampling periods measuring from under one second to 30 minutes. The xmperf tool can also record and play back metric values for long-term analysis. Using xmperf has little impact on system performance because the SPMI utilizes existing system calls to access the WLM information.

**Note**

Superclass configurations are defined as percentages of total system resources. Subclass attributes, such as shares, min, and max, are defined as percentages of the parent superclass allocations. However, PTX reports all class resource usage as a percentage of the total system resource.

The standard xmperf menus allow users to select the metrics to be displayed. Figure 55 on page 154, Figure 56 on page 155, and Figure 57 on page 156 show the hierarchy of WLM-related metrics.

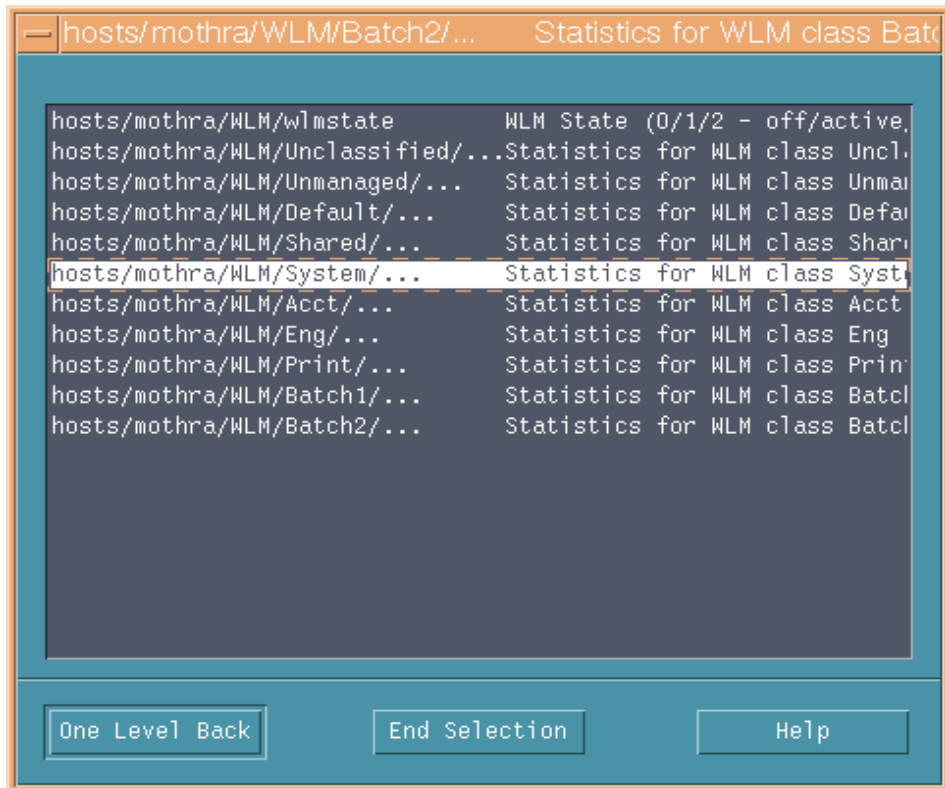


Figure 55. xperf selection list for available classes

Selecting *Statistics for WLM class System* takes you to a selection of the resources that are available to the selected class, *System*, as shown in Figure 56 on page 155.



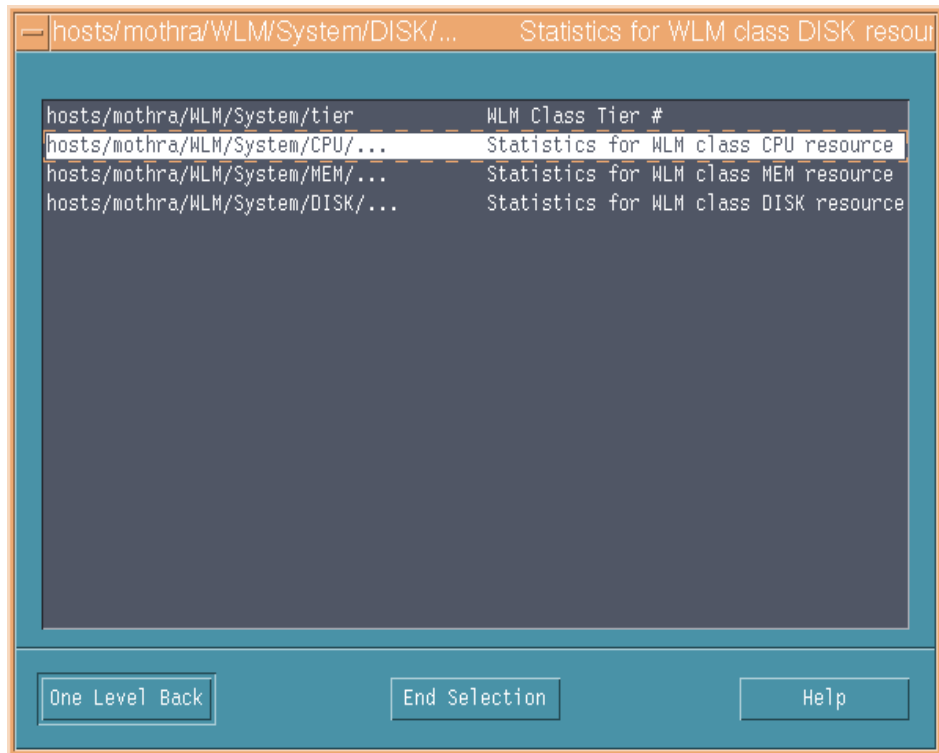


Figure 56. xperf selection list for the previously selected classes resources

Selecting *Statistics for WLM class CPU resource* takes you to a panel where you can select the resource attributes for the resource CPU for the class *System* as shown in Figure 57 on page 156.

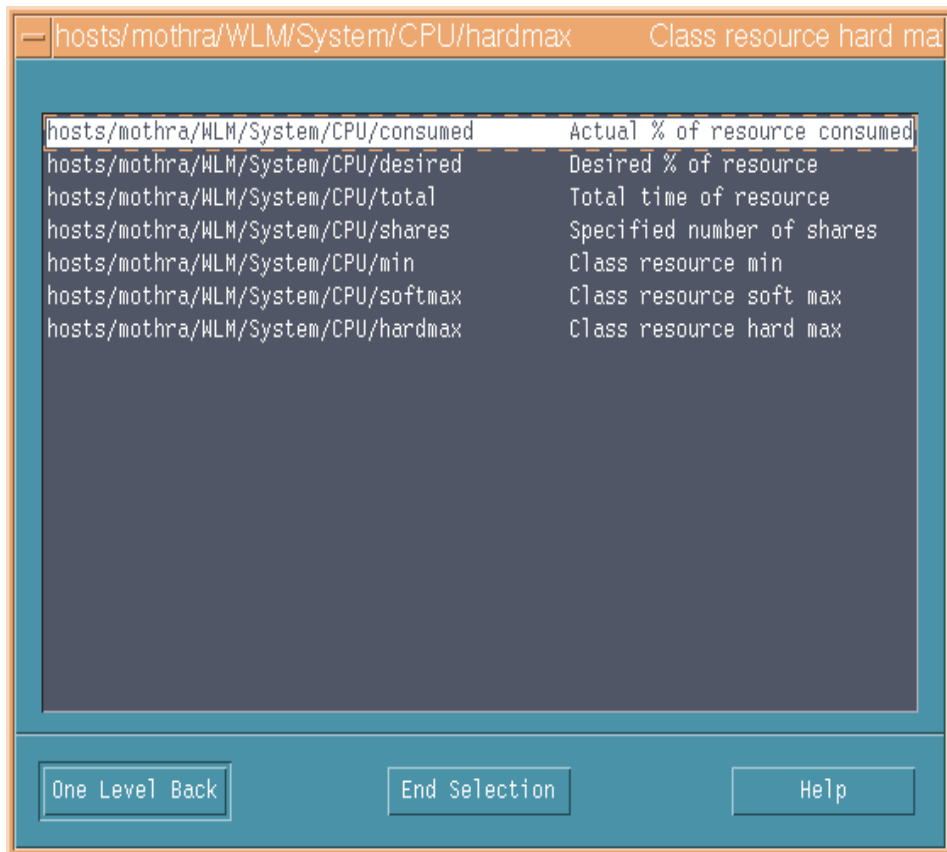


Figure 57. xperf selection list for the classes resource attributes

These xperf selections built PTX monitoring consoles.

The following are some examples of typical PTX monitoring consoles. The PTX console shown in Figure 58 on page 157 displays two instrument windows.

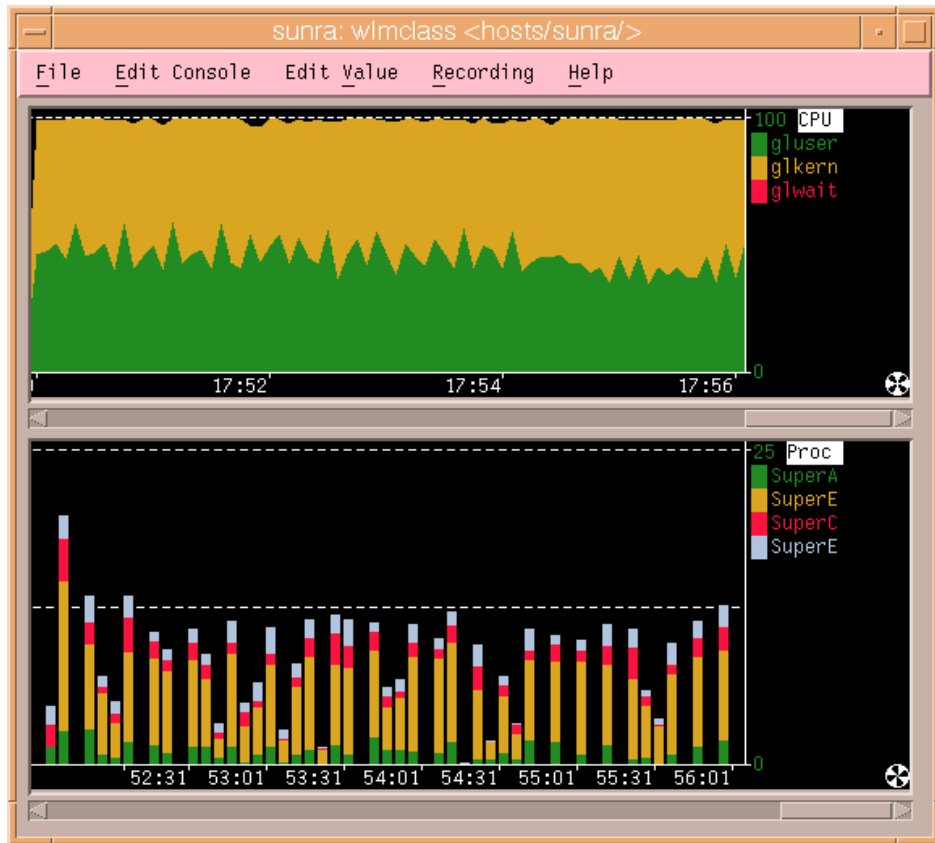


Figure 58. PTX console displaying CPU and class CPU metrics

The top instrument displays the CPU user, kernel, and wait metric values in a stacked area format. Stacked metrics are added together and displayed in separate colors. Here, user and kernel mode are each using about 50 percent of the system.

The lower instrument displays the load of four WLM superclasses on system CPU resource. The classes are stacked on top of each other in bar format. This format shows their relative sizes to each other. For display purposes, the upper scale is adjusted to 25 percent.

Both instruments are recording the data. In the PTX console, displayed in Figure 59 on page 158, the colors are used to associate the real time bars with the associated metric.

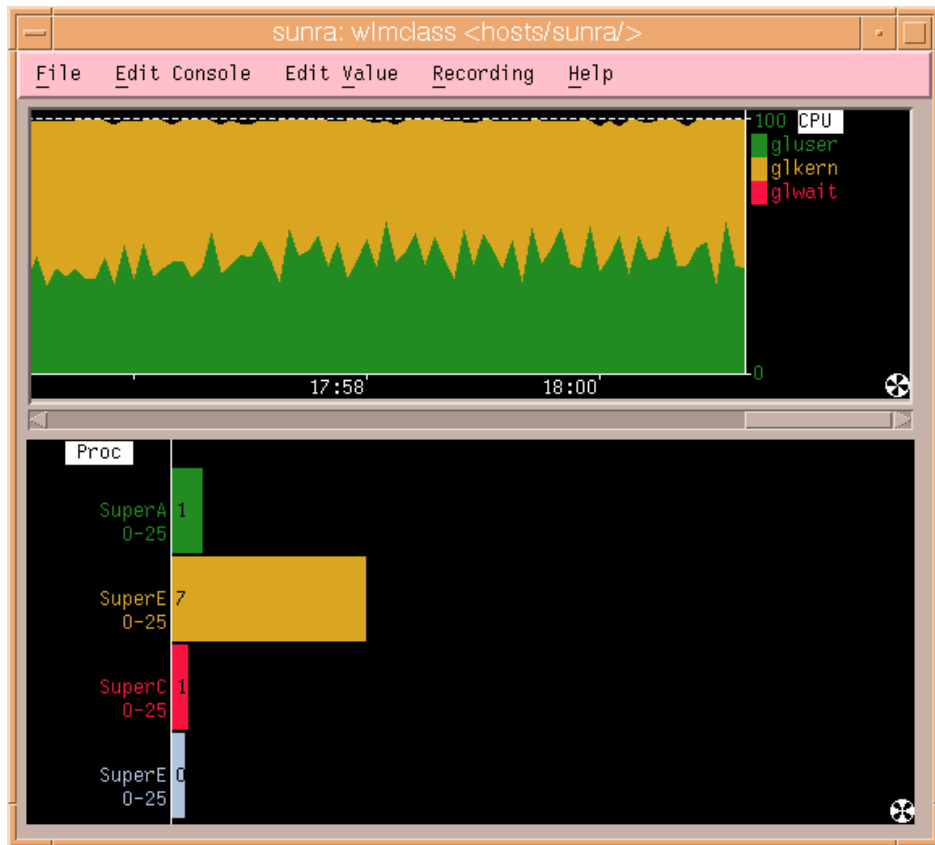


Figure 59. PTX console displaying WLM class metrics in bar format

Here again, the upper scale for the lower instrument is adjusted to 25 percent. The lower instrument of the PTX console, shown in Figure 60 on page 159, shows the WLM class metrics in a pie format.

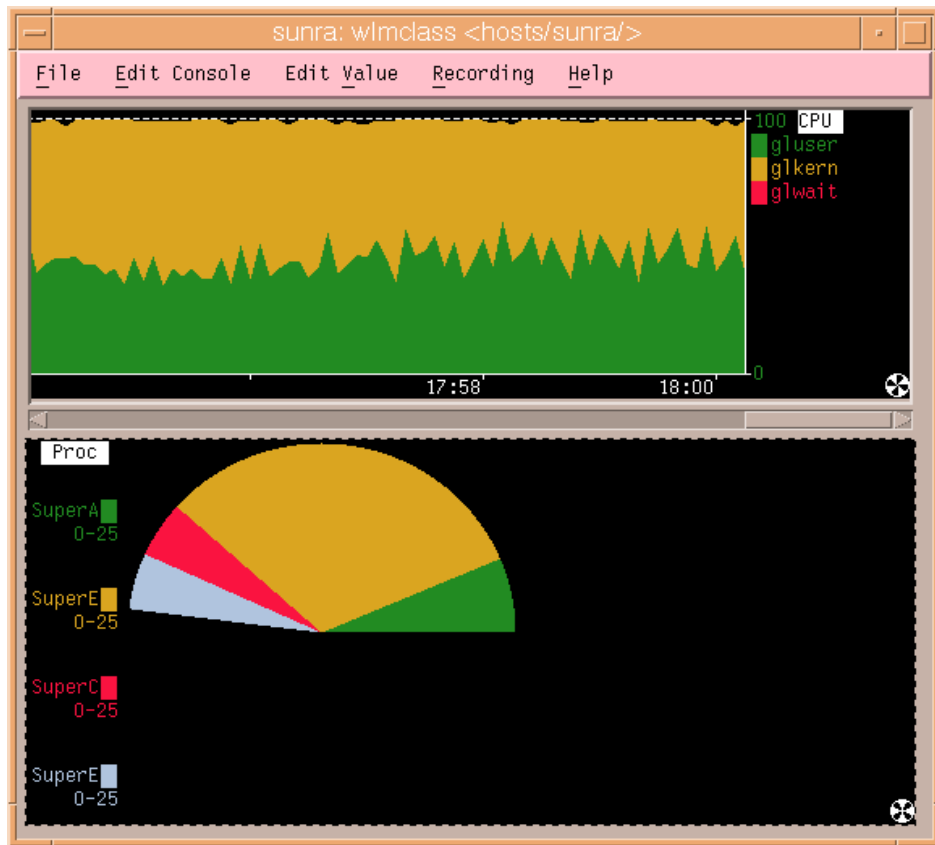


Figure 60. PTX console displaying WLM class metrics in pie format

#### 4.6.2 xmservd

The PTX Performance Aide consists of a set of agents and utilities for collecting, filtering, recording, and reporting performance metrics. The Performance Aide is required for the PTX Manager to view metrics remotely over the network.

The Performance Aide's primary agent is known as xmservd. This agent can also record metrics specified in a configuration file. The configuration file specifies the Metric name, start time, stop time, days to record, recording frequency, and other items. Refer to the *AIX Performance Toolbox User's Guide V1.2 and V2.1*, SC23-2625, for more details on using the Performance Aide to monitor a system. The HTML version of this guide ships with the base AIX media, along with other standard documentation. Performance Aide

recordings can be post-processed by the PTX Azizo and pxtab tools. The pxtab tool allows users to convert the recording into a comma- or tab-delimited spreadsheet format that can be imported into third-party spreadsheet applications.

### 4.6.3 Jazizo

Jazizo is a new tool for analyzing the long term performance characteristics of a system on PTX V3.0. It analyses recordings created by the `xmtrend` daemon, and provides customizable displays of the recorded data. Jazizo can be configured to show only the data of interest, in clear and concise graphical or tabular formats. Users can create, edit and save custom configurations. In addition, reports can be generated covering specific time periods, and data reduction options are provided to assist analysis.

The Performance Toolbox agents can collect hundreds of performance metrics available from the pool of resources available on a system. Jazizo was created to provide a simple user interface for analyzing recorded performance data over extended periods of time, primarily focused on aiding trend analysis. Specifically, jazizo can be used to graphically view resource usages over hours, days, weeks, or months to help determine if resources are, or will be, constrained. Recurring peak usage periods can also be identified and reviewed.

The syntax is:

```
jazizo [ -r RecordingFile [ -c ConfigurationFile ] ]
```

Where `-r RecordingFile` is a file created by `xmtrend` or a directory that contains `xmtrend` recording files, and `-c ConfigurationFile` is a file describing both metric and graph options.

### 4.6.4 wlmmon / wlmperf

The new `wlmmon` tool in AIX 5L Version 5.1 and `wlmperf` tool, available with PTX V3.0 for AIX 5L and AIX Version 4.3.3, provide graphical views of Workload Manager (WLM) resource activities by class. While the `wlmstat` command provides a per-second fidelity view of WLM activity, it is not suited for long-term analysis. The `wlmmon` and `wlmperf` tools were created to supplement `wlmstat`. These tools provide reports of WLM activity over much longer time periods. The `wlmmon` tool is a disabled version of the `wlmperf` tool. The primary difference between the two tools is the period of WLM activity that can be analyzed. The records of `wlmperf` are limited to one year, while `wlmmon` is limited to generating reports for the last 24 hour period. The records are generated by associated daemons that have minimal impact on

overall system performance. In wlmmon, this daemon is called xmwlm, and ships with the base AIX. For wlmperf, the xmtrend daemon is used to collect and record WLM. These daemons sample WLM and system statistics at a very high rate (measured in seconds), but only record supersampled values at a low rate (measured in minutes). These values represent the minimum, maximum, mean, and standard deviation values for each collected statistic over the recording period. To execute wlmmon and wlmperf, you can enter `wlmmon` or `wlmperf` without any options. This section explains the execution with `wlmperf`; any differences with `wlmmon` are pointed out in the according sections.

### **Daemon recording and configuration**

Both daemons mentioned above create recordings in the `/etc/perf/wlm` directory.

For `wlmperf`, the `xmtrend` daemon is used, and will utilize a configuration file for recording preferences. A sample of this configuration file for WLM related recordings is located at `/usr/lpp/perfagent/xmtrend_wlm.cf`. Recording customization, startup, and operation are briefly described in the section below. For more information, please refer to the *AIX Performance Toolbox User's Guide V1.2 and V2.1*, SC23-2625, which is available on the Performance Aide V3 media.

For `wlmmon`, the `xmwlm` daemon is used, and cannot be customized. For recordings to be created, adequate disk allocations must be made for the `/etc/perf/wlm` directory, allowing at least 10 MB of disk space. Additionally, the daemon should be started from an `/etc/inittab` entry so that recordings will automatically restart after system reboots. The daemon will operate whether the WLM subsystem is in active, passive, or disabled (off) mode. However, recording activity is limited when WLM is off.

### **xmtrend**

The `xmtrend` agent can be started from the command line or near the end of the `/etc/inittab` file. The general format of the command line is as follows:

```
xmtrend [-f infile] [-d recording_dir] [-n recording_name] [-t trace_level]
```

<code>-f</code>	Allows the user to specify a configuration file to use instead of the default. If <code>-f</code> is not used, <code>xmtrend</code> looks for and uses <code>/etc/perf/xmtrend.cf</code> as the configuration file. A configuration file must be available so <code>xmtrend</code> knows what to monitor.
-----------------	---

- d Specifies the output directory for the recording files. The default is to place the recording files in the /etc/perf directory.
- n Specifies a name for the recording file. By default, `xmtrend` creates recording files named `xmtrend.some date`. If `-n myrecording` is specified, the recording files will be named `myrecording.some date`.
- t Specifies a trace level. `xmtrend` prints various information to a log file in /etc/perf. The trace level can be set from 1 to 9. The higher the trace level, the more trace data is generated. This trace data is useful to determine `xmtrend` recording status and for debugging purposes. The log file name is either `xmtrend.log1` or `xmtrend.log2`. `xmtrend` will cycle between these two files after a file reaches the maximum size.

In order to start the recording, the daemons have to be active. To start the graphic monitoring tool, run the `wlmmon` command (base AIX) or the `wlmpref` command (PTX).

Upon startup, a default Report Display is shown. To view recordings, use the *WLM\_Console Menu* as described in the next section.

### **The WLM\_Console Menu**

The tab down menu `WLM_Console`, shown in Figure 61 on page 163, displays the following selections:

- |          |  |
|----------|--|
| Open log | Allows you to browse to and view recordings                                  |
| Reports  | Allows you to open, copy, and delete reports (for <code>wlmpref</code> only) |
| Print    | Allows you to print the current report                                       |
| Exit     | Exits the <code>wlmmon</code> tool   |



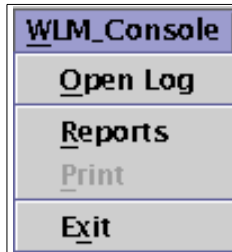


Figure 61. Tab down menu WLM\_Console

### **WLM Report Browser**

When selecting the **Open Log** menu, the Report Browser is displayed as shown in Figure 62. The browser allows you to browse through the different directories and displays a list of reports.

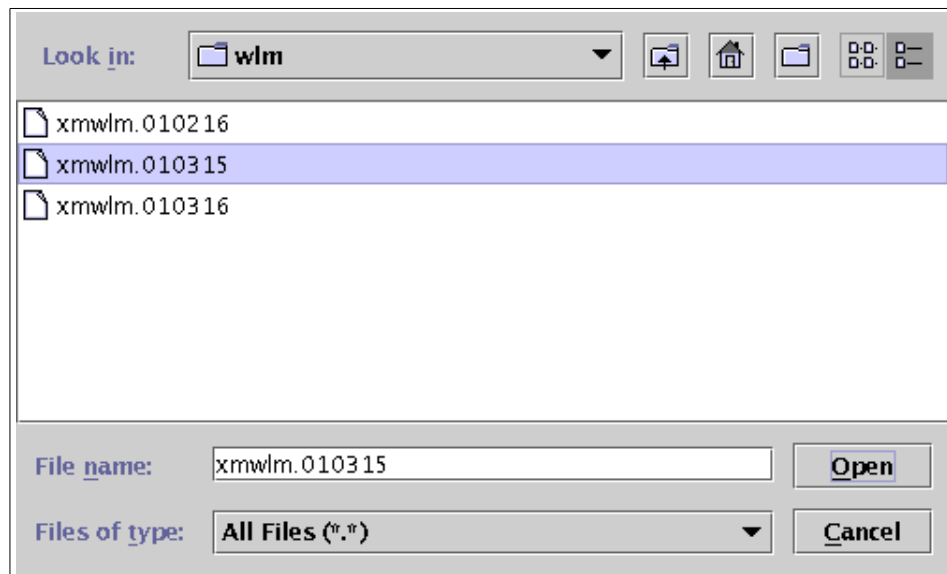


Figure 62. Report browser

### **Report Displays**

There are three types of report displays; snapshot display, bar display, and tabulation display. The bar display is opened by default.

These three displays have the following common elements:

WLM Console	Tab down menu that allows you to open recordings (log file), open reports (wlmperv only), print reports, and exit the tool.
Selected	Tab down menu that allows you to select the Report Properties.
Tier Column	Displays the tier number associated with a class.
Class Column	Displays the class name.
Resource Columns	Displays the resource information (CPU, memory, disk I/O) based on the type of graphical report selection chosen.
Status area	Displays a set of global system performance metrics that are also recorded to aid in analysis. The set displayed may vary between AIX releases, but will include metrics such as run, queue, swap queue, and CPU busy.
Host	Displays the hostname of the system on which the recording was made.
WLM State	Displays the state of WLM. This can be Active or Passive.
Time Period	Displays the time period defined in the <i>Times</i> menu of the Report Properties Panel. For trend reports comparing two time periods, two time displays are shown.

- **Bar Display**

As shown in Figure 63 on page 165, the resource columns are displayed in bar-graph style, along with the percentage of measured resource activity over the time period specified. The percentage is calculated based on the total system resources defined by the WLM subsystem. If the Bar display is trended, the later (second) measurement is shown above the earlier (first) measurement interval. Refer to Section “Times Menu” on page 168 for more trend information.

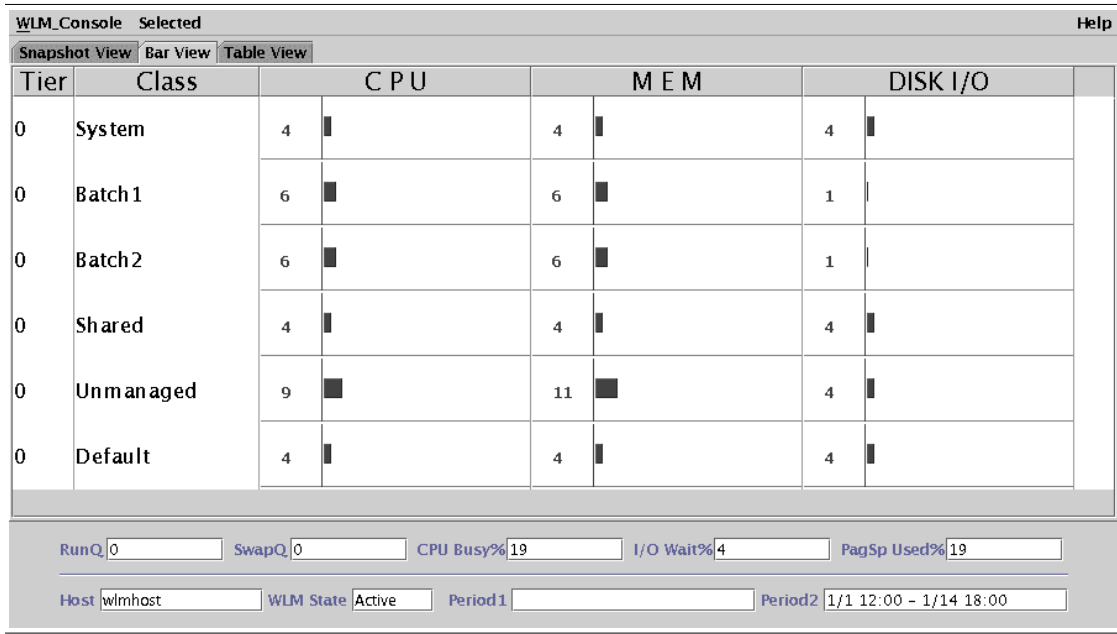


Figure 63. Bar view

- **Snapshot Display**

Figure 64 on page 166 shows the snapshot display showing class resource relationships based on user-specified variation from the defined target shares. To select or adjust the variation parameters for this display, utilize the Report Properties Panel *Advanced* menu, as shown in Figure 72 on page 173. If the snapshot display is trended, the earlier (first) analysis period is shown by an arrow pointing from the earlier measurement to the later (second) measurement. If there has been no change between the periods, no arrow is shown. Refer to Section “Times Menu” on page 168.

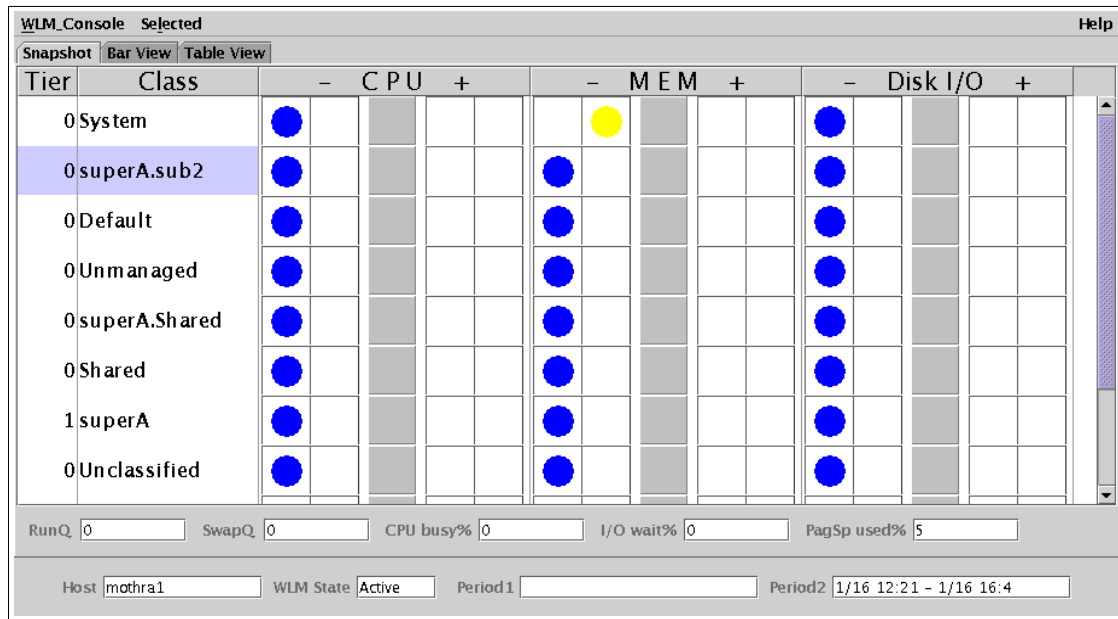


Figure 64. Snapshot view

- **Tabulation Display**

The third type of display report is shown in Figure 65 on page 167. In this report, the following fields are provided:

Shares	Defined shares in WLM configuration.
Target	Computed share value target by WLM in percent. If the share is undefined, the target displays 100.
Min	Class minimum defined in WLM limits.
SMax	Class soft maximum defined in WLM limits.
HMax	Class hard maximum defined in WLM limits.
Actual	Calculated average over the sample period.
Low	Actual observed min across time period.
High	Actual observed max across time period.
Standard Deviation	Computed standard deviation of Actual, High, and Low. Indicates the variability of the Actual values during the recording period. Higher standard deviation means more variability, lower standard deviation means less variability.

Samples

Number of recorded samples for this period.

Tier	Class	Shares	Target	Min	SMax	HMax	Actual	Low	High	StDev	Samples
0	System	4	4	50	48	48	4	0	9	0	637
0	Batch1	49	6	49	48	49	6	0	14	0	637
0	Batch2	48	6	47	48	49	6	0	14	0	637
0	Shared	49	4	49	49	49	4	0	9	0	637
0	Unmanaged	50	9	48	47	49	9	0	19	0	637
0	Default	49	4	48	48	48	4	0	9	0	637

RunQ: 0    SwapQ: 0    CPU Busy%: 19    I/O Wait%: 4    PagSp Used%: 19

Host: wlmhost    WLM State: Active    Period1:    Period2: 1/1 12:00 - 1/14 18:00

Figure 65. Table view

If the Tabulation Display is trended, the earlier (first) analysis is shown by the first number between the brackets, and the later (second) analysis is shown by the second number between the brackets. Refer to Section “Times Menu” on page 168 for more information about the trend display.

### Report Properties

The *Report Properties Panel* allows the user to define the attributes that control the actual graphical representation of the WLM data. The Report Properties are displayed by selecting **Selected** at the top of the Report Display, as shown in Figure 66.

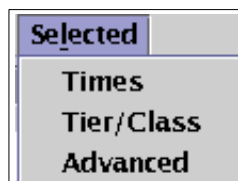


Figure 66. Report Properties

- **Times Menu**

The first tabbed panel is displayed in Figure 67 on page 169. It allows the user to edit the time properties of a display.

**Note**

wlmmom does not allow selection of days, weeks, and months.

Trend Box	Indicates that a trend report of the selected type will be generated. Trend reports allow the comparison of two different time periods on the same display. Selecting this box enables the <i>End of first Period</i> field for editing.
Width of Interval	Represents the period of time covered by any display type, measuring from user-input time selections. <i>Interval widths</i> are selected from this pull down menu. The selections available vary depending on the tool being used. While wlmmom only has selections for minutes and hours, wlmperf has selections for minutes, hours, days, weeks, and months.
End of First Period	Represents the end time of a period of interest for generating a trend report. The first period always represents a time frame ending earlier than the last period. This field can only be edited if the <i>Trend Box</i> is selected.
End of Last Period	Represents the end time of a period of interest for trend and non-trend reports.

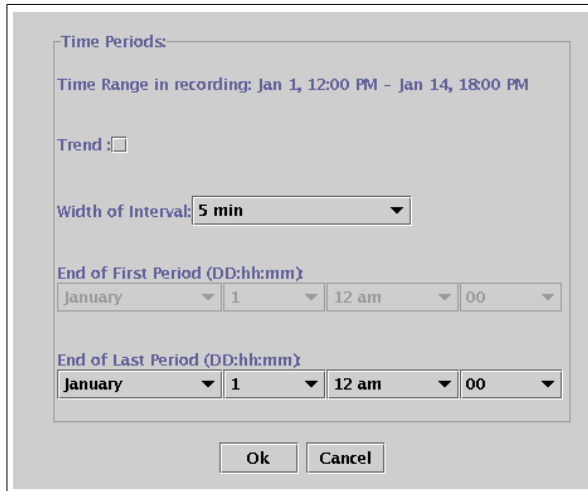


Figure 67. Times menu

Figure 68 is an example of a trend selection. The display shows different usage of resources between the two time periods. The time periods are displayed in the fields called Period 1 and Period 2. The bars on top mark the later recording period and the bars on the bottom mark the earlier recording period.

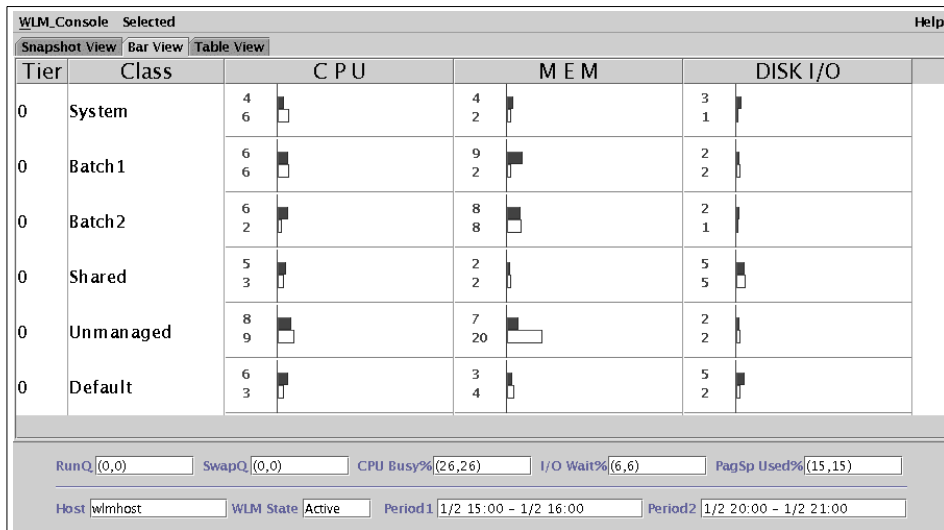


Figure 68. Example of trend display, bar view

Figure 69 also shows an example of a snapshot display using the trend option. The locations of the arrows mark the status during the earlier recording (Period 1) and the direction in which the resource usage of the class was moving. The colored dots mark the status during the later recording (Period 2).

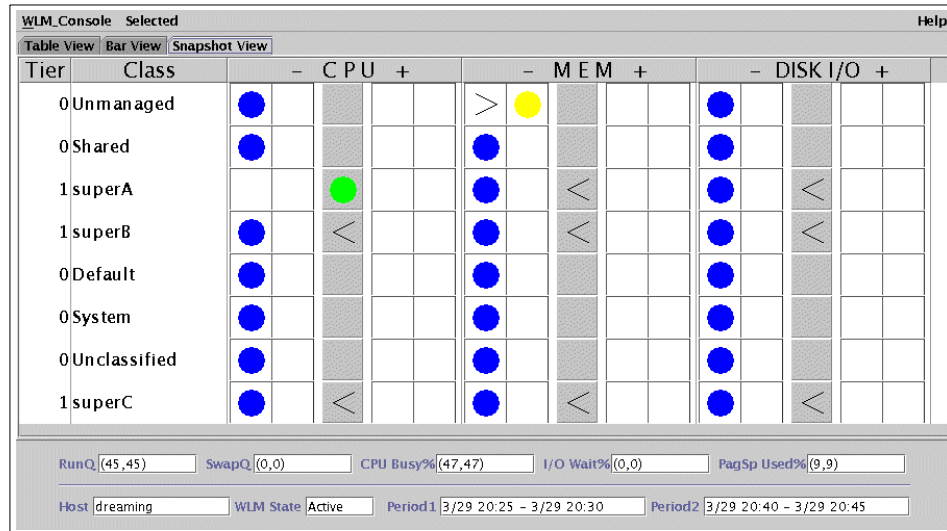


Figure 69. Example of trend display, snapshot view

Figure 70 on page 171 shows an example of a tabulation display using the trend option. The first number marks the later recording (Period 2) and the second number marks the earlier recording (Period 1).



WLM Console Selected											Help	
Table View											Bar View	Snapshot View
CPU												
Tier	Class	Shares	Target	Min	SMax	HMax	Actual	Low	High	StDev	Samples	
0	Unmanaged	(0,0)	(100,100)	(0,0)	(100,100)	(100,100)	(0,0)	(0,0)	(0,0)	(0,0)	(5,5)	
0	Shared	(0,0)	(100,100)	(0,0)	(100,100)	(100,100)	(0,0)	(0,0)	(0,0)	(0,0)	(5,5)	
1	superA	(33,33)	(33,33)	(0,0)	(50,50)	(50,50)	(35,46)	(17,29)	(66,65)	(12,9)	(5,5)	
1	superB	(33,33)	(33,33)	(0,0)	(50,50)	(50,50)	(26,0)	(0,0)	(43,0)	(11,0)	(5,5)	
0	Default	(0,0)	(100,100)	(0,0)	(50,50)	(50,50)	(0,0)	(0,0)	(0,0)	(0,0)	(5,5)	
0	System	(0,0)	(68,96)	(10,10)	(50,50)	(50,50)	(0,0)	(0,0)	(1,1)	(0,0)	(5,5)	
0	Unclassified	(0,0)	(100,100)	(0,0)	(100,100)	(100,100)	(0,0)	(0,0)	(0,0)	(0,0)	(5,5)	

RunQ	(45,45)	SwapQ	(0,0)	CPU Busy%	(47,47)	I/O Wait%	(0,0)	PagSp Used%	(9,9)
Host	dreaming	WLM State	Active	Period1	3/29 20:25 - 3/29 20:30	Period2	3/29 20:40 - 3/29 20:45		

Figure 70. Example of trend display, table view

- **Tier/Class Menu**

The second tabbed pane is displayed in Figure 71 on page 172. It allows users to define the set of WLM tiers or classes to be included in a report. The pull down menu at the top allows the user to select whether superclasses or Tiers are to be included or excluded in the Report Display. The list on the bottom then allows the user to select specific tiers or superclasses.

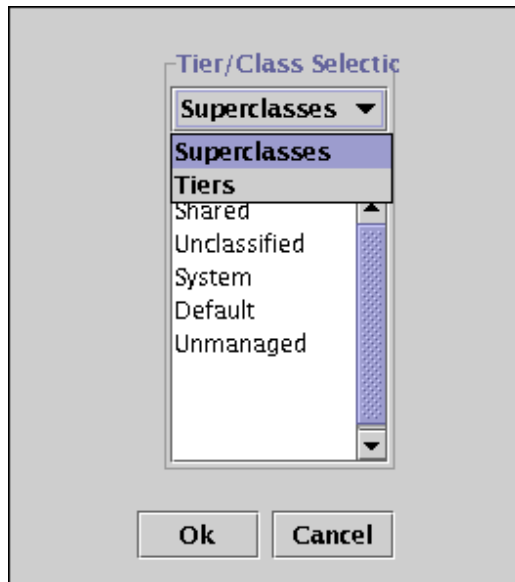


Figure 71. Tier / Class menu

- **Advanced Menu (Snapshot Option Panel)**

The third tabbed pane of the *Report Properties* panel is displayed as shown in Figure 72 on page 173. It provides advanced options for the snapshot display. For snapshots, exclusive methods for coloring the display are provided for user selection. *Option 1* ignores the minimum and maximum settings defined in the configuration of the WLM environment while *Option 2* utilizes the minimum and maximum settings.

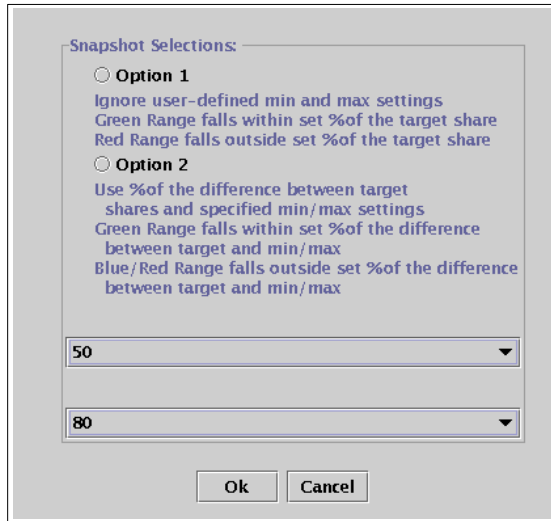


Figure 72. Advanced menu

### Example of the Advanced Menu

Figure 73 on page 174 shows an example that describes the functions of the Advanced Menu.

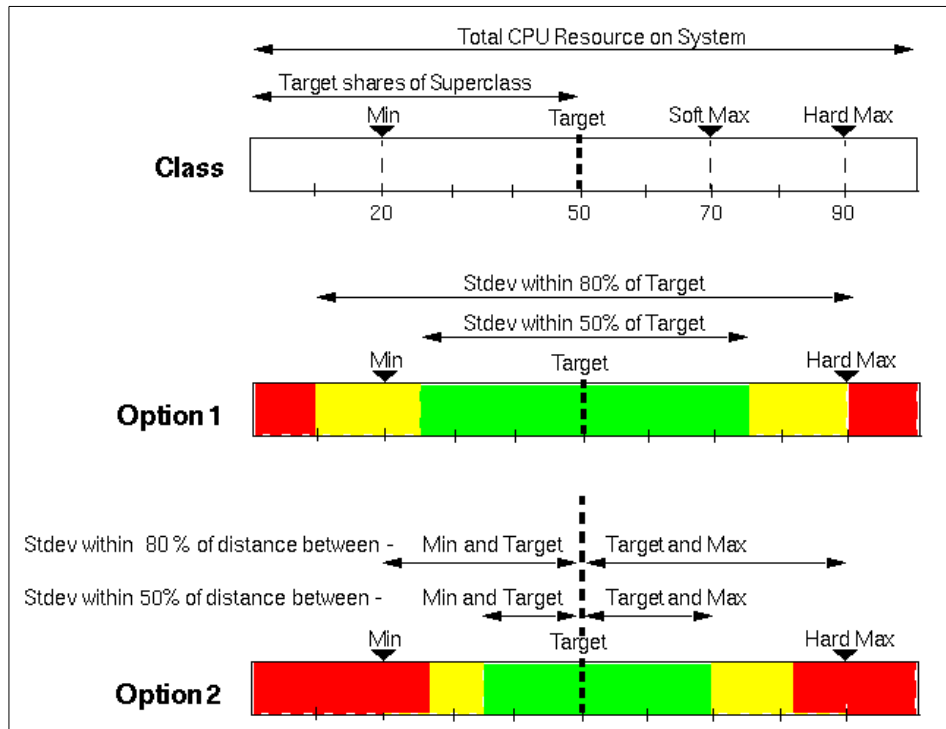


Figure 73. Example of the Advanced Menu

Figure 73 shows a class definition with its soft and hard minimum and maximum. The class has as a target (share value) 50 percent, a minimum limit (MIN) of 20 percent and maximum limit (MAX) of 90 percent. The following describes the functions of the two advanced options.

**Option 1** ignores the user-defined min and max settings. In this example, we selected option 1 with 50 percent as the green range percentage (green%) and 80 percent as the red range percentage (red%), as shown in Figure 72 on page 173.

To define the green range, the following formula is used:

$$\text{Low green range} = \text{Target} - (\text{Target} \times \text{green}\%) = 50 - (50 \times 50\%) = 25$$

$$\text{High green range} = \text{Target} + (\text{Target} \times \text{green}\%) = 50 + (50 \times 50\%) = 75$$

Figure 73 shows the green range from 25 percent to 75 percent on the scale of 0 to 100 percent.

The red range is calculated with the same formula but with the red range percentage.

Low red range = Target - (Target x red%) = 50 - (50 x 80%) = 10

High red range = Target + (Target x red%) = 50 + (50 x 80%) = 90

The red range is shown in Figure 73, option 1, 0 to 10 percent and from 90 to 100 percent. The area between the red and green range is yellow.

**Option 2** takes in account the predefined minimum limit and maximum limit settings. If we use the same advanced options as in Figure 72 on page 173, the red and green range are interpreted between the target and the hard minimum and hard maximum definitions (here 20 and 90 percent).

Low green range = Target - ((Target - MIN) x green%)  
= 50 - ((50 - 20) x 50%) = 35 percent on the scale from 0 to 100 percent.

High green range = Target + ((MAX - Target) x green%)  
= 50 + ((90 - 50) x 50%) = 70 percent on the scale from 0 to 100 percent.

Low red range = Target - ((Target - MIN) x red%)  
= 50 - ((50 - 20) x 80%) = 26 percent on the scale from 0 to 100 percent.

High red range = Target + ((MAX - Target) x red%)  
= 50 + ((90 - 50) x 80%) = 82 percent on the scale from 0 to 100 percent.

## Files

/usr/bin/wlmmmon	Base AIX, located in perfagent.tools.
/usr/bin/xmwlm	Base AIX, located in perfagent.tools.
/usr/bin/wlmparf	Performance Toolbox, located in perfmgr.analysis
/usr/lpp/perfagent/xmtrend_wlm.cf	Performance Aide, located in perfagent.server

## Prerequisite filesets

The following filesets are prerequisites for wlmmmon:

- Java130.rte.bin
- Java130.rte.lib
- perfagent.tools (at least level 2.2.33.50 for AIX Version 4.3.3)
- perfagent.tools (at least level 5.1.0.0 for AIX 5L)

The following filesets are prerequisites for wlmparf:

- perfagent.server
- perfagent.common

- perfmgr.network
- perfmgr.analysis

All of the above filesets are part of the AIX Performance Toolbox Version 3 level 3.0.0.0.

---

## Chapter 5. Manual assignment

The automatic assignment, used by WLM throughout its whole execution, is based on five attributes. These attributes are the process' characteristics used as classification criteria; user name, group name, application pathname, process type, and application tag. Refer to Section 2.5.1, "Automatic assignment" on page 22 for more information on how automatic assignment works. With these attributes as classification criteria, it is practically impossible for WLM to automatically classify two instances of the same application differently. Unless the application itself uses WLM's API routine, `wlm_set_tag`, to tag all its occurrences differently, all these attributes will, most of the time, be equal throughout all instances of a process. For example, different Oracle database instances in a system are normally launched by the same user (therefore, having the same group), have the same executable, and, of course, are of the same type. If application tagging is not being used, WLM cannot place the database instances in different classes, but, depending on the importance to the business that these instances might have, the system administrator might want to assign the resources throughout these processes differently. That is when manual assignment joins the party.

Manual assignment is a feature introduced in AIX 5L. It allows system administrators and applications to, at any time, override the traditional WLM automatic assignment (processes' automatic classification based on class assignment rules) and force a process to be classified in a specific class. The following sections focus on the description of manual assignment and on some sample scripts that can be used to manually assign different instances of some database products.

---

### 5.1 Description

The manual assignment can be made or canceled separately at the superclass level, the subclass level, or both. In order to manually assign processes to a class or cancel an existing manual assignment, a user must have the proper level of privilege (that is, they must be the root user, `adminuser/adminingroup` for the superclass, or `authuser/authgroup` for the superclass or subclass). A process can be manually assigned to a superclass only, a subclass only, or to a superclass and a subclass of the superclass. In the latter case, the dual assignment can be done simultaneously (with a single command or API call) or at different times, possibly by different users.

A manual assignment will remain in effect (and a process will remain in its manually-assigned class) until:

- The process terminates.
- WLM is stopped. When WLM is restarted, the manual assignments in effect when WLM was stopped are lost.
- The class the process has been assigned to is deleted.
- A new manual assignment overrides a prior one.
- The manual assignment for the process is canceled.

In order to assign a process to a class or cancel a prior manual assignment, the user must have authority both on the process and on the target class. These constraints translate into the following:

- The root user can assign any process to any class.
- A user with administration privileges on the subclasses of a given superclass (that is, the user or group name matches the attributes, `adminuser` or `admingroup`, of the superclass) can manually reassign any process from one of the subclasses of this superclass to another subclass of the superclass.
- An user can manually assign his/her own processes (same real or effective user ID) to a superclass and/or a subclass for which he or she has manual assignment privileges (that is, the user or group name matches the attributes, `authuser`, or `authgroup` of the superclass or subclass).

This defines three levels of privilege among the persons who can manually assign processes to classes, root, of course, being the highest. In order for a user to modify or cancel a manual assignment, he or she must be at the same level of privilege as the person who issued the last manual assignment or higher.

### 5.1.1 First assignment

In this section, the first time assignment is described with a few examples.

The system administrator manually assigns process *P1* from the superclass *superA* to the superclass *superB*. The automatic assignment rules for the subclasses of the superclass, *superB*, will be used by WLM to determine which subclass of the *superB* superclass the process is ultimately assigned to. *P1* will end up, for instance, in the subclass *superB.subA*, and is flagged as having a *superclass only* assignment.



A user with the right privileges assigns a process, *P2*, from its current class, *superA.subA*, to a new subclass of the same superclass, *superA.subB*. *P2* is assigned to its new subclass and flagged as having a *subclass only* assignment.

The WLM administrator of the subclasses of the superclass *superB*, can decide to manually reassign the process *P1* to another subclass of *superB*, for instance, *subC*. *P1* will be reclassified into *superB.subC* and will be now flagged as having *both superclass* and *subclass* level assignment.

### 5.1.2 Reassignment and cancellation

In this section, the reassignment and assignment cancellation are explained with a few examples.

Suppose that the system administrator thinks that *P2* should really be in a superclass with more resources, and decides to manually assign *P2* to the superclass, *superC*. Previously, *P2* was manually assigned to the subclass, *subB*, of the superclass, *superA*, with a *subclass only* assignment. Because *P2* is assigned to a different superclass, the previous manual assignment becomes meaningless and is canceled. *P2* now has a *superclass only* manual assignment to the superclass, *superC*, and is assigned to a subclass of *superC* using the automatic assignment rules.

Now the system administrator decides to terminate the manual assignment from *P1* to the superclass, *superB*, set up earlier. *P1*'s *superclass level* manual assignment is canceled, and *P1* is assigned a superclass using the top level automatic assignment rules:

- If the rules have not changed, *P1* will be assigned to the superclass, *superA* (its original class), and its *subclass level* manual assignment to *superB.subC* above becomes meaningless and is canceled.
- If for some reason the top level rules assign *P1* in superclass *superB*, then the subclass level assignment to *superB.subC* is still valid and remains in effect. *P1* now has a *subclass only* manual assignment.

The reassignment/cancellation of a manual assignment at the subclass level is simpler and just affects the subclass level assignment.

### 5.1.3 Interaction with inheritance

When a process is manually assigned to a superclass and/or subclass with the inheritance attribute set to *yes*, if the process is a process group leader, WLM will attempt to reclassify all the processes in the process group.

So, the class inheritance attribute has two interpretations depending on if we are dealing with automatic or manual assignment. See Figure 74.

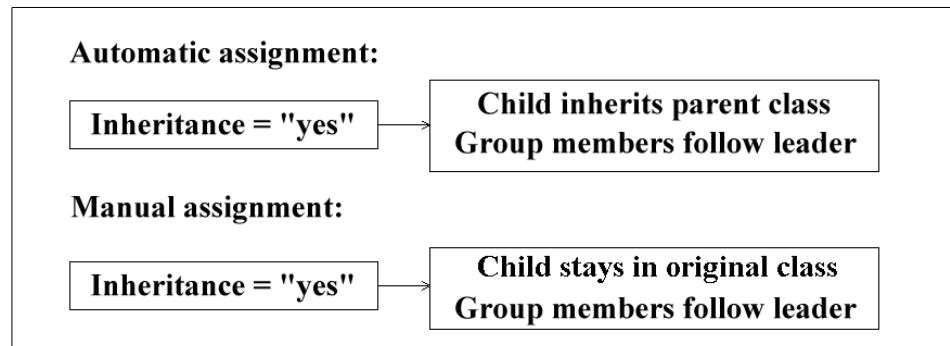


Figure 74. Inheritance in automatic and manual assignments

Let us describe how all this works together with a few examples.

Refer to Figure 75 on page 181 for an illustration of the first example:

1. Process *A*, classified into *class1*, which has inheritance set to *yes*, launches the child processes *A1* and *A2*.
2. *A1* and *A2* get classified into *class1* as well.
3. The system administrator manually assigns process *A* into *class2*, which also has inheritance set to *yes*. *A1* and *A2* stay in *class1*.
4. Process *A* launches a new child process, *A3*, which gets classified in *class2*.
5. The manual assignment of process *A* is cancelled. *A* goes back to *class1*, and *A3* stays in *class2*.

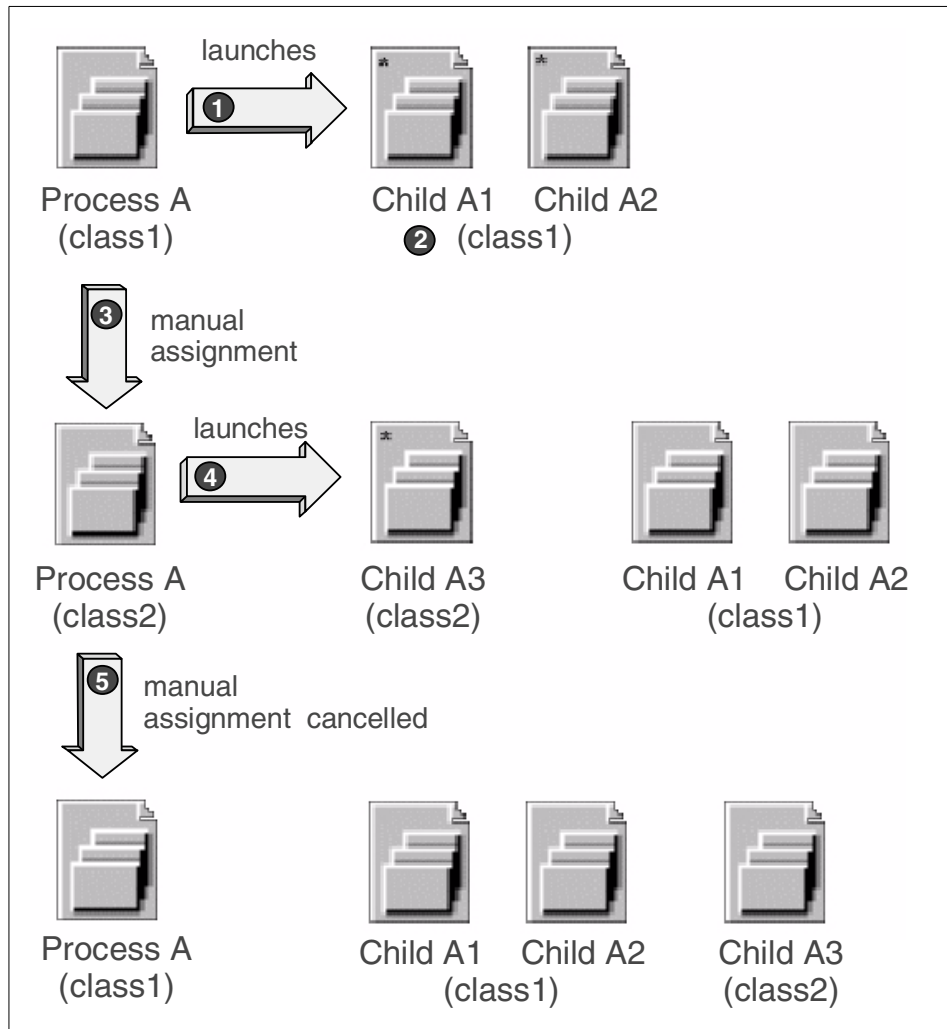


Figure 75. First example of manual assignment and inheritance interaction

Refer to Figure 76 on page 182 for an illustration of the second example:

1. Process *B* is the leader of a process group (*PGID1*) of which processes *C* and *D* are members. Processes *B*, *C*, and *D* are automatically classified in *class1*.
2. The system administrator manually assigns process *B* to *class2*, which, as we know, has inheritance set to *yes*.
3. Processes *C* and *D* follow the process group leader *B* into *class2*.

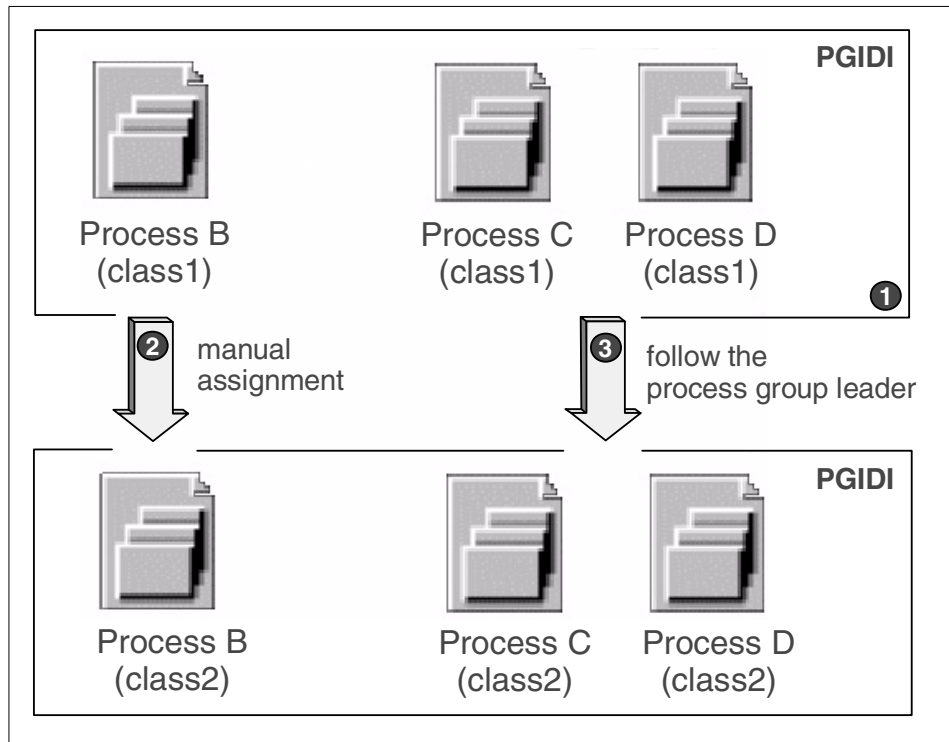


Figure 76. Second example of manual assignment and inheritance interaction

There are cases where some of the processes in the process group will not be reclassified to the new class of the group leader. For instance, if some of the processes themselves have been manually assigned to their current class, they will remain in their class.

## 5.2 Manual assignment methods

A process or a group of processes can be manually assigned to a superclass and/or subclass using the WLM administration interfaces Web-based System Manager (WSM) and SMIT, the command `wlmassign`, or an application using the WLM API function `wlm_assign`.

The `wlmassign` command and inheritance attribute are provided in AIX Version 4.3.3 maintenance level 8 to give AIX 4.3.3 users early access to some of the features available in Workload Manager with AIX 5L. See Section 5.2.1, “AIX Version 4.3.3 maintenance level 8 manual assignment” on page 188.

### **Command line - *wlmassign***

The command used in WLM to perform manual assignments and unassignments is `wlmassign`. The syntax of the command is:

```
wlmassign [ -s | -S ] [ -u | class ] [ pid_list ] [ -g pgid_list ]
```

The options to this command are:

- u Cancel any manual assignment in effect for the processes in the *pid\_list* or the *pgid\_list*. If none of the `-s` or `-S` flags are used, this cancels the manual assignments for both the superclass and the subclass level.
- S To specify a superclass-only level assignment or unassignment when used with a subclass name of the form, *supername.subclass*.
- s To specify a subclass-only level assignment or unassignment, when used with a subclass name of the form, *supername.subname*.
- g *pgid\_list* To indicate that the following is a list of pgids (and not pids, which would be what the command would interpret by default).

The `wlmassign` command is used to:

- Assign a set of processes specified by a list of process identifiers (pids) and/or process group identifiers (pgids) to a specified superclass or subclass, thus overriding the automatic class assignment or a prior manual assignment
- Cancel a previous manual assignment for the processes specified in *pid\_list* and/or *pgid\_list*, allowing the processes to be subjected to the automatic assignment rules again.

The `wlmassign` command allows you to specify processes using a list of pids, a list of pgids, or both. The format of these lists is `pid[,pid[,pid[...]]]` or `pgid[,pgid[,pgid[...]]]`, that is comma (,) separated lists of pids and pgids.

The name of a valid superclass or subclass must be specified to manually assign the target processes to a class. The assignment can be done or canceled at the superclass level, the subclass level, or both. The processes can be assigned to the superclass only by specifying the `-s` option or the subclass only by specifying the `-S` option. For a manual assignment, if the class name is the name of a superclass, the processes in the list will be assigned to the superclass. The subclass will then be determined using the assignment rules for the subclasses of the superclass. If the class name is a subclass name, *supername.subname*, the processes will, by default, be

assigned to both the superclass and the subclass. The following are examples from Table 2 on page 26:

1. To assign a process with pid 9846 to superclass *VPs*, enter:

```
# wlmassign -S VPs 9846
```

or:

```
# wlmassign VPs 9846
```

This is a superclass-only assignment. The assignment rules for superclass *VPs* select a subclass for the process (for instance, *Default*).

2. To assign at the subclass level the process with pid 9846 from *VPs.Default* to *VPs.editors*, enter:

```
# wlmassign -s VPs.editors 9846
```

or:

```
# wlmassign VPs.editors 9846
```

This would become a superclass and subclass assignment.

3. To cancel the subclass level assignment of a process with pid 9846 (the process still has the superclass level assignment staying; thus, in superclass *VPs* and being submitted to the superclass assignment rules), enter:

```
# wlmassign -u -s 9846
```

4. Finally, to cancel the superclass level assignment of a process with pid 9846 (making it be submitted to the general configuration assignment rules):

```
# wlmassign -u -S 9846
```

### **SMIT**

A process can be manually assigned in SMIT by accessing the *Assign/Unassign processes to a class/subclass* screen or using the following fastpath

```
# smitty wlmassign
```

For instance, to manually assign a process with pid 9846 to superclass *VPs*, (from the example in Table 2 on page 26) and subclass *editors*, see Figure 77 on page 185.

```

Assign/Unassign processes to a class/subclass

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                     [Entry Fields]
Assign/Unassign to/from Superclass/Subclass/Both  Assign Superclass      +
Class name (for assignment)                      [VPs.editors]           +
List of PIDs                                     [9846]                  +
List of PGIDs                                    []                       +

F1=Help      F2=Refresh      F3=Cancel      F4=List
F5=Reset     F6=Command     F7=Edit       F8=Image
F9=Shell     F10=Exit       Enter=Do

```

Figure 77. Manual assignment in SMIT

To manually unassign the same process from the subclass *editors* of *VPs* superclass, the administrator can assign it to another class or cancel its subclass level assignment as shown in Figure 78.

```

Assign/Unassign processes to a class/subclass

Type or select values in entry fields.
Press Enter AFTER making all desired changes.

                                     [Entry Fields]
Assign/Unassign                                unassign subclass      +
Class name                                     []                       +
List of PIDs                                   [9846]                 +
List of PGIDs                                    []                       +

F1=Help      F2=Refresh      F3=Cancel      F4=List
F5=Reset     F6=Command     F7=Edit       F8=Image
F9=Shell     F10=Exit       Enter=Do

```

Figure 78. Subclass level unassignment in SMIT

Finally, to unassign the same process from the superclass *VPs* altogether, the system administrator can assign it to another class or cancel its superclass level assignment, as shown in Figure 79.

```

Assign/Unassign processes to a class/subclass
Type or select values in entry fields.
Press Enter AFTER making all desired changes.

Assign/Unassign
Class name      [Entry Fields]
List of PIDs   unassign class      +
List of PGIDs  [VPs ]              +
               [9846]            +
               [ ]              +

F1=Help        F2=Refresh      F3=Cancel      F4=List
F5=Reset       F6=Command      F7=Edit        F8=Image
F9=Shell       F10=Exit        Enter=Do

```

Figure 79. Superclass level manual unassignment in SMIT

The PGID's could have been used as well to perform the assignments and unassignments.

### **WSM**

In WSM, manual assignment and unassignment is done by right clicking the configuration name to work with in the *Configurations/Classes* screen and by choosing the *Add or Remove Processes* option as shown in Figure 80 on page 187.



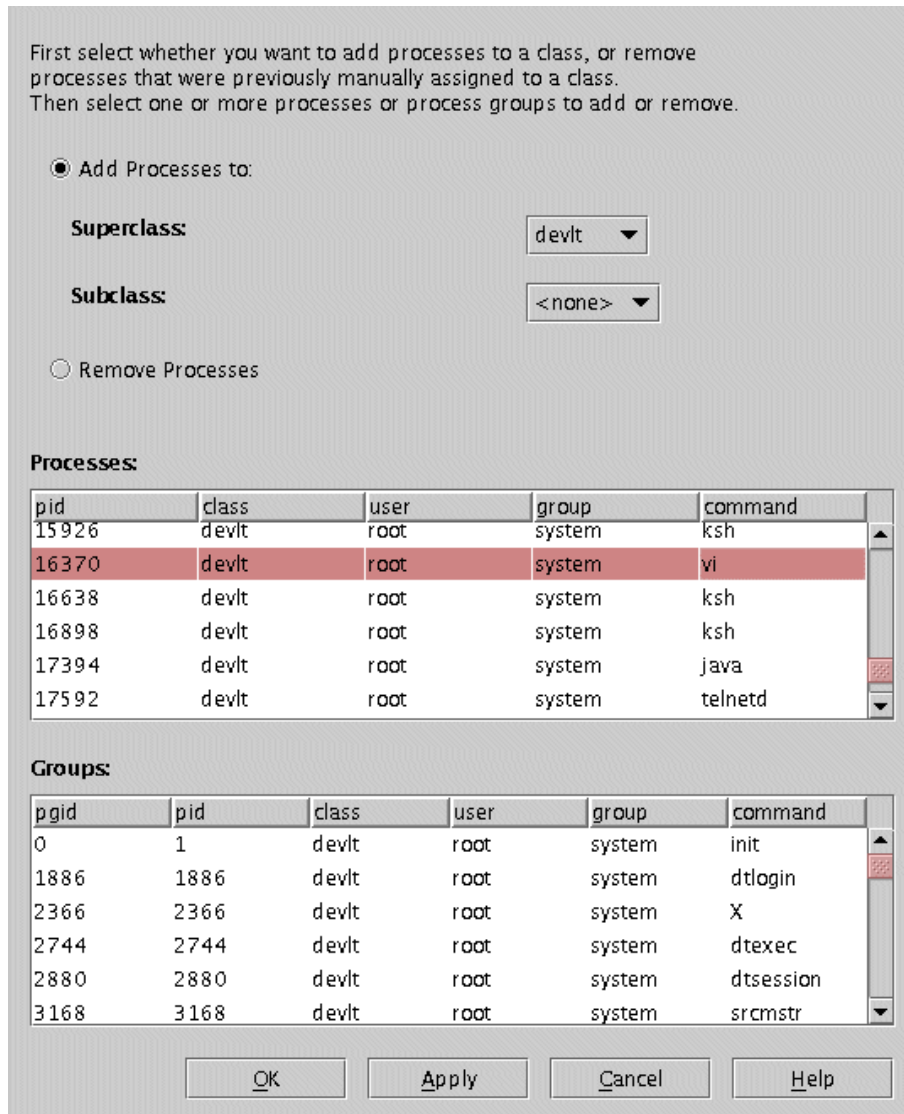


Figure 80. Manual Assignment in WSM

### Applications - `wlm_assign`

An application can perform its own manual assignments and unassignments, using, for that purpose, one of the WLM API routines; `wlm_assign`. For more information about manual assignment in the API, see also Chapter 6, “WLM Application Programming Interface (API)” on page 195, and Section A.6, “WLM management” on page 280.

### 5.2.1 AIX Version 4.3.3 maintenance level 8 manual assignment

AIX Version 4.3.3 maintenance level 8 has been enhanced to support manual assignment on a per process ID basis. The command used to perform manual assignments is `wlmassign`. The `wlmassign` command assigns the process designated by its process ID (`pid`) to the class named `classname`. This overrides the automatic assignment. The syntax of the command is:

```
wlmassign classname pid
```

The options to this command are:

<code>classname</code>	The name of the class the process is to be assigned to.
<code>pid</code>	The process ID of the process to be assigned to the new class.

The `wlmassign` command is restricted to the root user.

#### Note

Sometimes processes are manually assigned to a class that has the *localshm* attribute set to *yes* in order to prevent commonly accessed memory segments from being accounted to the Shared super- or subclass.

That means that up until these processes are manually assigned to this particular class, their memory segments may be accounted to the Shared class if they are accessed by a process in another class, and will remain there as long as they are being used by both processes, even after the processes get manually assigned to the correct class.

For this reason it is recommended to manually assign processes right after their creation or to re-assign them when the application is not actively executing. It is also recommended to set the *inheritance* attribute of the class the processes get manually assigned to, to *yes*.

---

### 5.3 Oracle database example

The examples described in this section focus on the Oracle database, which is one of the most commonly-known databases capable of running more than one instance at the same time in the same system. Some databases, including an Oracle instance, change their instances to show the instance name in them (for example *ora\_dbwr\_wlmdb*). Using this knowledge, we can differentiate Oracle's instances by the name of their processes in the process table. The scripts supplied are general enough (or easily modifiable) to meet

any application whose behavior is similar to the examples. For example, if we have an Oracle instance named *wlmdb* and another named *acct* running on the same machine, the output of `ps -ef | grep ora` for their processes would be something like the following:

```
# ps -ef | grep ora
oracle 35614 1 0 23:20:49 - 0:00 ora_dbwr_wlmdb
oracle 35872 1 0 23:20:49 - 0:00 ora_reco_wlmdb
oracle 36130 1 0 23:20:49 - 0:00 ora_pmon_wlmdb
oracle 36388 1 0 23:20:49 - 0:00 ora_smon_wlmdb
oracle 36654 1 0 23:20:49 - 0:00 ora_lgwr_wlmdb
oracle 63186 1 0 23:20:50 - 0:00 ora_d000_wlmdb
oracle 94736 1 0 23:20:49 - 0:00 ora_s000_wlmdb
oracle 75614 1 0 23:20:49 - 0:00 ora_dbwr_acct
oracle 75872 1 0 23:20:49 - 0:00 ora_reco_acct
oracle 76130 1 0 23:20:49 - 0:00 ora_pmon_acct
oracle 76388 1 0 23:20:49 - 0:00 ora_smon_acct
oracle 76654 1 0 23:20:49 - 0:00 ora_lgwr_acct
oracle 73186 1 0 23:20:50 - 0:00 ora_d000_acct
oracle 94737 1 0 23:20:49 - 0:00 ora_s000_acct
root 64040 85492 7 23:56:12 pts/21 0:00 grep ora
```

From this knowledge, it is possible to create a Korn shell script set to run at every WLM start to classify these instances differently using manual assignment. This script should verify that all the instances' processes are up and running before executing the manual assignment. If the Korn shell script is set as an entry in the `/etc/inittab`, a position close to the end is recommended.

A sample script for this situation is provided in Section C.1, "Oracle example script" on page 307. Its functionality is described here.

### **Configuration file**

The script uses a configuration file, which, for the sake of the example, is `/etc/wlm/ma.conf`. The format of this configuration file is one line per required manual assignment. These lines have the following format:

```
<Instance name> <Class> <Inheritance>
```

where:

- *Instance name* is the Oracle (or other application in a similar situation) instance name.
- *Class* is the name of the classes to manually assign the processes to. This name is *supername* for the superclasses and *supername.subname* for the subclasses.

- *Inheritance* is a flag that is set to *yes* if the processes belonging to a process group (whose leader is a process being manually assigned) should be manually assigned as well. If the group members should stay in the original class, this flag must be set to *no*.

#### **Data structure**

The script uses as data structure an array of three positions, named `MANUAL`, where:

- Position 0 takes the instance name.
- Position 1 takes the class name.
- Position 2 takes the inheritance flag value.

#### **Function**

The script has one single function, named `getpids()`, which receives, as a parameter, the instance name whose processes are to be manually assigned. The function gets the processes' IDs related to that instance (the ones that have the instance's name as part of their *own* name in the process table) and returns them in the format of a comma (,) separated list.

#### **Script process**

For each line read from the configuration file, the script does the following:

- Sets the `MANUAL` array with the values read from the configuration file (instance name, class name, and inheritance flag) and works with this data structure.
- Saves the inheritance attribute value of the target class and sets it to its new value.
- Invokes `getpids()` to get a comma (,) separated list of the PIDs to be manually assigned, that is, the ones related to the instance in question.
- Manually assigns the list of processes to the target class.
- Reverts the inheritance attribute value of the target class to the saved one.

---

## **5.4 DB2 UDB**

Spawned DB2 processes always belong to the AIX user-ID that created the DB2 instance, such as the user-ID *db2inst1*. Thus, each DB2 instance created on the server is owned by a unique AIX user-ID. If WLM is to be configured to control more than one DB2 instance, the applied configuration of WLM must simply have automatic assignment rules that match the user-ID of each DB2 instance so no special scripts have to be developed to differentiate DB2's several instances.

However, there is a more sophisticated use of WLM in combination with DB2 that requires some basic understanding about the way DB2 works.

#### 5.4.1 DB2 process model

DB2 has several groups of processes that perform different, specialized tasks. Two very important kinds are the *agents* and the *io servers* (or *prefetchers*). Briefly, the agents carry out the logic of the queries and the prefetchers are in charge of reading data from disk and copying it to the *bufferpools*, which are shared memory segments where the agents go to find the data they want to work with (filter, sort, compare, transform, etc.). In some cases, agents are also allowed to read the data files directly from disk, but we will assume that the tuning of the database parameters will prevent this from being relevant.

Agents “live” in a *pool* if they are idle. The database administrator configures what the pool must look like (for instance, how many idle agents are allowed to be prepared to serve a query or how many of them must be present when the engine starts). When a query enters the system, the DB2 optimizer can decide to start more than one agent to fulfill the query; that is, the engine picks some idle agents from the pool (if they are available) or spawns as many processes as are needed to match the *query degree* (how many agents work for a particular query). As soon as the query finishes, the agents go back to the pool and are ready to serve another query. Obviously, we do not want an infinite number of processes doing nothing. So if the pool becomes too big (there are too many idle agents according to the database parameters), some processes will be terminated. There is a possibility of having no pool of agents at all; simply set a database parameter to zero. In that case, no idle agent is allowed and the processes are removed when the query finishes.

In contrast, there is no pool of prefetchers. The database administrator fixes a number of them and they are permanently started. If they have no work, they simply stay there waiting for requests. If all prefetchers are busy and an agent needs to read some data, it can read the data by itself. Although we do not ever expect prefetchers to hog the CPU, it might happen (in case of very aggressive read-ahead parametrization) that the prefetchers consume a noticeable amount of CPU resources. In fact, reading a lot and reading fast requires considerable CPU resources.

Another essential difference between prefetchers and agents has to do with the way in which DB2 allocates processes to queries. It is possible to know exactly which agents are working for a given running query. On the other hand, there is no way to know which query a prefetcher is working on; they

see only a queue of requests to read data blocks rather than a list of queries. Prefetchers are intended to serve all queries and all agents.

### 5.4.2 Using AIX WLM with DB2 UDB

In a complex query environment, keeping in mind the concepts introduced in Section 5.4.1, “DB2 process model” on page 191, the following concerns arise:

- Could we control the resource consumption of individual queries?
- Could we dynamically determine that some queries should be favoured and some others penalized in terms of CPU resources in order to guarantee a certain execution time for important queries?
- How can WLM help us to achieve it?

DB2 has a feature called “*snapshot monitoring*” that allows us to take a photo of the engine status at any time. We can know how many queries are running on the system, which users started which queries, which tables a query accesses, how much memory a query needs for sorting data, if a query is doing a *hash-join* or not, how the prefetching mechanism is working, and so on. We can also obtain performance data, both aggregated and particularized for a certain query, that we can use to know up to the finest detail what the running queries are really doing.

If we take such a database snapshot, look for some decision criteria, get the process ids of the agents, and, by using the manual assignment feature of WLM, send them to different classes configured according to our needs, we can develop a very sophisticated strategy of workload balancing. The different classes can be configured to prevent some queries from hogging the CPU or, on the other hand, allow a heavy query to complete within certain time margins.

A simple, but complete test scenario was developed to illustrate this procedure. Due to its length and specialization, we decided not to include it in this chapter, but it is available with the additional material delivered with this redbook. See Appendix F, “Using the additional material” on page 313 for details.

---

## 5.5 Conclusion

Manual assignment is a very useful enhancement to WLM’s automatic assignment functionality. In WLM’s first release, all the instances of a database were classified in the same manner, disregarding the importance

each one of them could have to the business. Manual assignment allows additional classification options (providing more flexibility of control over some important applications) essential to successful server consolidation.

**Note**

Keep in mind that all manual assignments are cancelled if WLM or the applications are stopped. If, for any reason, root or the privileged system administrator needs to stop and restart WLM or any of the manually-assigned applications, the manual assignments need to be redone.





---

## Chapter 6. WLM Application Programming Interface (API)

The AIX Workload Manager Application Programming Interface (API) is comprised of a set of routines in the */usr/lib/libwlm.a* library. These routines provide applications with the capability to perform all the tasks a WLM administrator can carry out using the WLM commands; that is, create, change, and remove classes; manually assign processes to specific classes; and get WLM statistics. In addition, a routine, `wlm_set_tag`, allows an application to set up a process tag and specify whether this tag should be inherited by child processes at `fork` and/or `exec` times. With AIX 5L Version 5.1 the WLM API has also been enhanced to support per class accounting using the routines `wlm_initkey`, `wlm_class2key`, `wlm_key2class`, and `wlm_endkey`. The library provides support for multi-threaded 32 or 64 bit applications. Refer to Appendix A.1, "The Include file - sys/wlm.h" on page 261 for a technical description of the `sys/wlm.h` header file.

The API routines have the additional ability (over WLM commands' regular functionality) to make changes only to the currently-running configuration (in-core) data in the kernel, not saving them into the property files (thus, not making them available after restarting WLM). These changes can only be seen in the directory that holds the image of the running configuration, */etc/wlm/running*.

The application programmer must be aware that there are some initialization routines in the API that must be run before any others. Refer to Appendix A.3, "Initialization routines" on page 271 for the technical description of the initialization routines.

---

### 6.1 Application tag

The application tag interface, `wlm_set_tag`, is a technique provided to the applications that want to have some level of control over how their various instances are classified, such as databases. The *tag* is a string of characters that is used as one of the classification criteria for the automatic classification of processes (using the rules file). This, basically, provides a process with an additional classification condition to add to the already defined ones, such as user, group, application pathname, and process type. Refer to Appendix A.4, "Application tag" on page 273 for a technical description of the `wlm_set_tag` routine.

### 6.1.1 Description

When an application process sets its tag, it is immediately reclassified using the superclass and subclass rules in effect for the currently-active WLM configuration. WLM goes through the assignment rules looking for a match using all the process attributes, including the new tag. In order to be effective, this tag must appear in one or more of the assignment rules. This means that the format and the use of the various tags each application might create must be clearly specified in the application's administration documentation. This way, WLM administrators get to know all the choices of values a specific application tag might take and can use them in their assignment rules to distinguish between different instances of the same application.

Different system administrators might have different requirements depending on what set of application process characteristics they want to use to classify them. It is recommended that the application provide a set of configuration or runtime attributes that could be used to build the tag. This would provide the application administrator with the ability to specify the format of this tag to the application. The attributes that can be used for the tag and the syntax to be used to specify the format of the WLM tag are application dependent, and are the responsibility of the application provider.

### 6.1.2 An application tag situation

Let us suppose that an instance of a database server is able to determine which database is working on *db\_name* and through which TCP port, *port\_num*, a given user is connected. Some WLM administrators may want to create different classes for processes accessing different databases and give each class different resource entitlements. Others might want to separate the processes serving remote requests by different origins by using the port number as a classification attribute. Others might want both and create one superclass for each database and subclasses per port numbers in each superclass. A way of accommodating these different needs would be to specify the content and format of the tag. We can imagine, for the sake of the example, that this could be passed to the application in a configuration file or runtime parameter, such as:

```
WLM_TAG=<$db_name> or WLM_TAG=<$port_num>
```

or

```
WLM_TAG=<$db_name>_<$port_num>
```

When setting its tag, an application can specify whether or not it will be inherited by its children so that all the processes spawned by a specific

instance of an application can be classified in the same class. Setting the tag inheritance is probably how the application tag will be used most of the time.

Taking the example of a database, here is how application tags can be used:

Consider Table 2 on page 26, where the provider of a database server application could have specified that the tag would be the database name. In that case two instances of the server working on two different databases would set up two different tags, for instance, *\_db1* and *\_db2*. A system administrator could create two different classes, *db1* and *db2* and classify the two database servers (and all their children if tag inheritance is used) in these classes using the tags. It would then be possible to give each class a different resource entitlement according to specific business goals.

The corresponding assignment rules could look like:

```
* class resvdusergroupapplicationtypetag
*
db1- - - /usr/oracle/bin/db*-_db1
db2- - - /usr/oracle/bin/db*-_db2
```

### 6.1.3 Example of an application tag program

A simple program to launch an application with a specified tag is provided in Appendix D, "Sample program for application tag" on page 309. Let us say the program is called *settag*, and its syntax is:

```
# settag tag_name program_name
```

where:

*tag\_name* is the string we want to tag the application with.

*program\_name* is the application to be tagged.

Basically, the program procedure is:

- Run *wlm\_initialize*, which is required before using any other API routine (refer to Appendix A.3, "Initialization routines" on page 271 for a technical description of the initialization routines).
- Run *wlm\_set\_tag* to set the application tag. The *flags* argument of this routine is set in such a way that child processes of *settag* inherit the tag at *exec* and *fork* times.
- Launch the application, which inherits the tag from its parent, *settag*.

With this program, a system administrator can launch any application explicitly tagged and let WLM automatically classify it using, for that purpose, the rules that should have been previously created to handle the application tags.

As a usage example of this program, let us consider a Korn shell script, *test*, that simply issues a `sleep` command. The following rule was created to classify this process in the class *myclass* when issued with the `_mytag` tag:

```
myclass - root - test - _mytag
```

The next screen exhibits the test performed and the output obtained:

- First, the `settag` program was run to launch and tag the *test* process with `_mytag`.
- The `ps` command was used to check the classification and tagging process.
- Second, the `settag` program was run again to launch and tag the *test* process with `_notag`.
- The `ps` command was used to check the classification and tagging process.

Note that, in the first `settag` run, the process, *test*, and the child process, *sleep*, were classified correctly in the *myclass* superclass. The second time `settag` was run, both processes were classified in the System class because there is no rule for tag `_notag`, and root was being used in the tests. This demonstrates how an application can provide differentiation between its various instances using application tagging:

```
# settag _mytag test &
# ps -ae -o class,pid,ppid,tag,args |grep tag |grep -v grep |grep -v ps
myclass.Default  2270 7324  _mytag  sleep 100
myclass.Default  7324 12192 _mytag  sh -- test
# settag _notag test &
# ps -ae -o class,pid,ppid,tag,args |grep tag |grep -v grep |grep -v ps
myclass.Default  2270 7324  _mytag  sleep 100
myclass.Default  7324 12192 _mytag  sh -- test
System          9214 17192 _notag  sleep 100
System          17192 12192 _notag  sh -- test
```

---

## 6.2 Class management

The WLM API provides applications with the ability to:

- Query the names and characteristics of the existing classes of a given WLM configuration (`wlm_read_classes`)
- Create a new class for a given WLM configuration and define the values of the various attributes of the class (tier, inheritance, adminuser, admingroup, rset, authuser, and authgroup) and the shares and limits for the resources managed by WLM, such as CPU, physical memory, and disk I/O (`wlm_create_class`)
- Change the characteristics of an existing class of a given WLM configuration, including the class attributes and resource shares and limits (`wlm_change_class`)
- Delete an existing class of a given configuration (`wlm_delete_class`).

The changes will be applied only to the property files of the specified WLM configuration. Optionally, by specifying an empty string as the configuration name, it is possible to apply the change only to the currently-running classes, resulting in an immediate update of the state of the active configuration.

The API calls require the same level of privilege from the caller that would be required for the command line, SMIT, or WSM interfaces:

- Any user can read the class names and characteristics.
- Only root can create/modify/delete superclasses.
- Only root or designated superclass administrators (superclass attributes, adminuser, and admingroup) can create/modify/delete subclasses of a given superclass.

In cases where WLM administration is done both through the command line and administration tools by WLM administrators, and by applications through the API, some caution must be applied. Both interfaces share the same name space for the superclass/subclass names and the total number of superclasses and subclasses. In addition, when the API directly modifies the currently-running (in-core) WLM data (create new classes, for instance), the WLM administrators are not aware of this until they see classes they did not create appear on the output of commands, such as `wlmstat`. In order to avoid conflicts that would confuse the applications using this API, the classes created through the API that are not defined in the WLM property files are not automatically removed from the in-core data if the system administrator updates WLM. They remain in effect until explicitly removed through the

wlm\_delete\_class routine or through an invocation of the `rmclass` command (invoked directly or through SMIT or WSM by the system administrator).

Refer to Appendix A.5, "Class management" on page 274 for technical descriptions of the class management routines.

---

### 6.3 WLM management

The WLM API also provides applications with the ability to:

- Query/change the mode of operation of WLM using the `wlm_set` function.
  - Query the current status of WLM.
  - Stop WLM.
  - Switch from active to passive mode, and vice-versa.
  - Turn the rset binding on and off.
- Start/update WLM, using the current (or an alternate) configuration, with the `wlm_load` routine.
- Assign a process or a group of processes to a class using the `wlm_assign` routine.

Here again, the API requires the same levels of privilege as the corresponding command line interfaces, `wlmcntrl` and `wlmassign`:

- Any user can query the state of WLM.
- Only root can change the mode of operation of WLM.
- Only root can update/refresh a whole configuration.
- Only root or an authorized superclass administrator (adminuser/admingroup) can update WLM for the subclasses of a given superclass.
- Only root, an authorized user (specified by authuser/authgroup), or an authorized superclass administrator (adminuser/admingroup) can assign processes to a superclass and/or subclass.

Refer to Appendix A.6, "WLM management" on page 280 for technical descriptions of WLM management routines.

---

## 6.4 WLM statistics

The WLM API routines `wlm_get_info` and `wlm_get_bio_stat` provide applications with access to the WLM statistics displayed by the `wlmstat` command.

Refer to Appendix A.7, "WLM statistics" on page 285 for technical descriptions of WLM statistics routines.

---

## 6.5 WLM classification

The API routine `wlm_check` allows you to check the class definitions and the assignment rules for a given WLM configuration.

The API routine `wlm_classify` allows an application to find out which class a process with a specified set of attributes would be classified to.

Refer to Appendix A.8, "WLM classification" on page 290 for technical descriptions of WLM classification routines.

---

## 6.6 WLM accounting

- The API routine `wlm_initkey` allocates and initializes the classes to keys translation table.
- The API routine `wlm_class2key` performs class name to key translation.
- The API routine `wlm_key2class` retrieves a class name from a key.
- The API routine `wlm_endkey` frees the classes to keys translation table.

Refer to Appendix A.9, "WLM accounting" on page 293 for technical descriptions of WLM accounting routines.

---

## 6.7 Binary compatibility

In order to provide binary compatibility in the future if there are any changes in the data structures, each API call receives a version number as one of the parameters. This will allow the library to determine which version of the data structures the application has been built with, and read and/or write the correct data.

---

## 6.8 Integration with Tivoli products

By itself, WLM does not allow a system administrator to monitor the performance of an application. It can only work with system resources' usage and monitor if that usage is above or below the defined targets. However, an integration of the WLM API with Tivoli Application Performance Management (TAPM) can bring the best of the two worlds together; monitoring an application's availability and response time and its behavior at the system level (resource usage).

### 6.8.1 TAPM overview

TAPM focuses on two different approaches to measure applications availability and response time; application instrumentation and transaction simulation. Both methods consist of using the TAPM Application Response Measurement (ARM) API routines.

#### 6.8.1.1 Application instrumentation

The application instrumentation approach focuses on changing the application code to include ARM API function calls. This method has the advantage of giving the application control over what is monitored and when, but has the obvious drawback of the unavailability of the application's source code to many customers. An example of application instrumentation would be to measure the end-user's response time, which could be defined as the time between the user submitting the transaction and the screen refreshing with the result. In order to measure the end-user's response time, the ARM API calls that start and stop the TAPM agent timer have to be placed in the application code around the user transaction. In other words, an `arm_start` call must be made when the user clicks on the submit button, and an `arm_stop` call must be made when the screen refreshes.

The time the server component of the transaction takes could be measured in the same way.

#### 6.8.1.2 Transaction simulation

In the second approach, meant for when the application's source code is not available, typical end-user transactions are collected in a script for simulation purposes. This script is edited to include the ARM API function calls, just like the application instrumentation approach. The script is then set to run periodically from a dedicated client, simulating the chosen transactions. The measurements it provides are good approximations of real end-user experience.



## 6.8.2 TAPM and WLM

In both approaches described in the previous section, the WLM API can work together with the ARM API to gather statistics of both system resource usage and response times on an AIX application environment. This can help to determine if an application performance bottleneck resides in the application itself or in a less appropriate configuration of resource targets for the application class. The WLM API calls to gather these statistics are `wlm_get_info` and `wlm_get_bio_stat`. Refer to Appendix A.1, "The Include file - `sys/wlm.h`" on page 261 for a description of the routines' data structures, and to Appendix A.7, "WLM statistics" on page 285 for a technical description of the routines.

## 6.8.3 Monitoring an application in a WLM and Tivoli environment

In this section, the steps of the process of monitoring an application and using WLM and Tivoli products together are described.

The first step is to determine what to monitor and when:

- Which transactions within the application are to be studied?
- Which approach is to be used?
- At what time of day should the monitoring process run?
- What sort of system resource statistics are to be collected?

After the planning is done, the chosen method is applied. Applications are instrumented or scripts are written using the WLM and ARM API calls to collect the chosen statistics and performance measurements.

At this time, the instrumented applications or scripts need to be registered within the Tivoli environment and added to TAPM profiles before distributing them to any specific endpoints. The subsequent profile distribution will make the scripts or instrumented applications generate data. This data is stored in an external database, and, with the use of Tivoli Decision Support (TDS), reports can be generated from it.

The Distributed Monitoring agent provided with TAPM also enables you to detect and act upon any exceptions that might occur. These events can be forwarded to the Tivoli Enterprise Console (TEC). Examples of events needing immediate action would be a critical application getting resources way below its target (thus, presenting really low performance) or another process starving all other applications of a particular resource.

This process is briefly represented in Figure 81 on page 204.

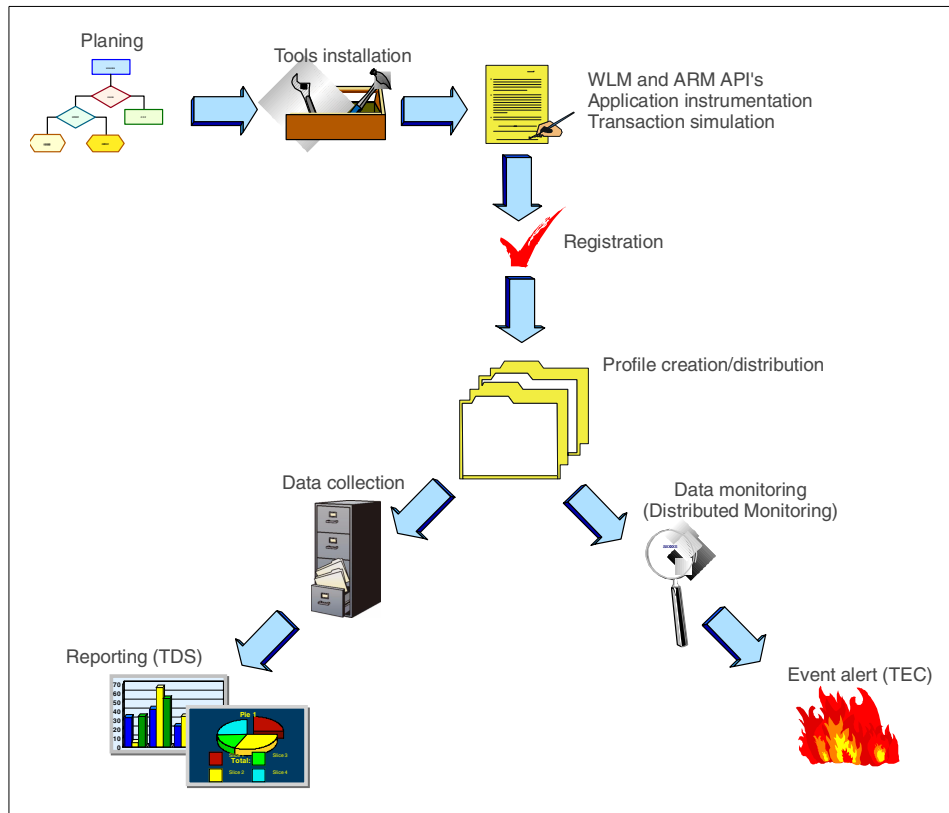


Figure 81. WLM and Tivoli interaction

## 6.9 Summary

The WLM API provides applications with the ability to:

- Perform regular WLM and class administration tasks
- Tag processes to extend the range of classification criteria
- Gather resource usage statistics
- Initialize per class accounting

With Tivoli product interaction, WLM's monitoring functionality can be extended to an application performance oriented one.

---

## Chapter 7. Sizing recommendations for Workload Manager

The introduction of WorkLoad Manager (WLM) has greatly enhanced the functionality of AIX, and helps to more efficiently use the capacity of IBM @server pSeries servers and RS/6000 SP systems. WLM provides the means to use otherwise wasted “overcapacity” without impairing the performance requirements of the primary workload(s). However, only after proper sizing and control of the nature and behavior of the workload mix will the expected improvement in overall system usage be achieved.

This chapter suggests some recommendations for system capacity sizing of stand-alone systems and server consolidation systems using AIX WLM. It does not deal with sizing theories for individual applications.

---

### 7.1 Typical UNIX system capacity sizing

Few production UNIX systems have an average utilization of more than 70 percent (often more than 80 percent is considered resource constrained). Moreover, it is not surprising to find that the average utilization of most UNIX systems is below 40 percent. This is chiefly due to the following reasons:

- System sizing should be based on the highest expected peak load, not on the average workload.
- Generally, system sizing is conservative and the sizing often results in a generous amount of buffering capacity, more than 20 percent, in addition to the top peak load.
- The duration of peak load time is, usually, not long.
- In most cases, a UNIX server is dedicated to only one application service, thus producing a single pattern of peak loads.

The typical UNIX system resource utilization, therefore, is similar to that shown in Figure 82 on page 206. Actually, a substantial percentage of the total system resource is wasted in most UNIX systems in preparation for peak loads that do not last long.

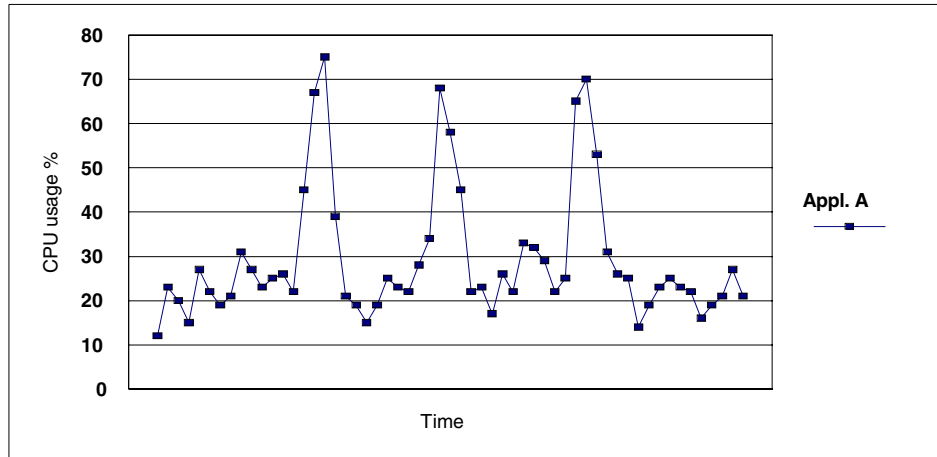


Figure 82. Typical CPU usage when running single application service

These peak loads cannot simply be ignored. When there is an unexpected peak of heavy workload whose resource consumption exceeds the system capacity, users often experience a duration of system hang-up until the load is over. This is one of the system administrator's nightmares. So, even if system resource utilization is quite low, a system large enough to survive such peak workloads without a hang-up has to be prepared.

---

## 7.2 Server consolidation considerations

The key to correctly sizing a UNIX system is to eliminate that wasted capacity. It would not be practical to try to change the behavior of the application itself. Nor would it be acceptable to force the service users not to produce those peak loads.

One of the more reasonable solutions to this problem is to combine multiple application services with different system resource utilization patterns into a single server, also known as server consolidation. By doing that, multiple patterns of peak loads can be combined to produce a greater average system usage.

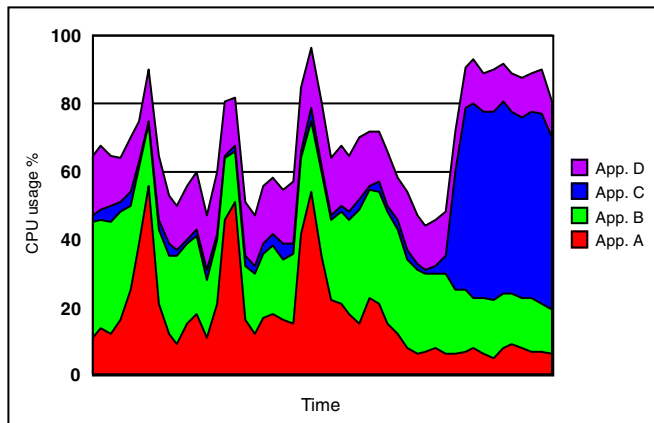


Figure 83. A typical CPU usage in a server consolidation environment

Integrating multiple applications that run on separate, single systems into one system of larger capacity is part of a server consolidation solution. Running multiple applications on one server of larger capacity has many pros and cons.

The pros are:

- Only one instance of the OS is required, thus saving the resources needed for multiple OS instances, such as memory and disk space.
- More flexible utilization of system resources.
- The total cost of ownership is decreased (that is, less maintenance cost and less manpower).
- Even though there is more complexity in the system being administered, there are fewer systems to be maintained (for operating system updates, for instance).
- Simpler architecture than that of distributed server systems.

The cons are:

- Running more than one application service in one system can lead to resource contention among the applications, thus degrading the performance of critical services or workloads.
- It is not always possible to limit the resource usage of some applications that are not mission-critical or tend to take up all the available system resources.

- If the system fails due to OS or other application errors, all other services are lost.
- If one application crashes or goes out of control, the other applications may be brought down as well.

Many of these problems can be overcome with the modern UNIX technologies. The availability problems can be addressed by UNIX clustering technologies, such as HACMP for AIX. The resource contention problems can be solved by using a workload management solution.

The main reason for performance degradation when running multiple applications in a single system is the resource contention between applications. AIX WLM can effectively isolate applications by controlling the resource allocation algorithm of the UNIX scheduler, virtual memory manager (VMM), and the I/O-bandwidth of disk devices so that applications of more importance can be configured to receive preferential allocation of resources compared to less important ones.

WLM provides for integrating multiple applications on single systems. Refer to Chapter 2, “AIX Workload Manager functionality” on page 9 for details.

---

### 7.3 System capacity sizing for Workload Management

Workload Management can be very useful in terms of system capacity usage in two ways:

- WLM can help, by integrating multiple applications on a single server, to utilize the unused portion of system resource that would be wasted in preparation for the peak loads if the applications ran on separate individual systems.
- WLM automates the process of (re-)scheduling system resources allocated to lower priority workloads back to high priority (critical) workloads whenever these enter their *peak load* period. This reallocation process can be so extreme that low priority jobs seem to be stopped. Therefore, the system should be sized sufficiently to handle the combined peak loads of critical workloads. Although some buffering (that is, extra resources) may still be desired to meet increasing resource requirements by critical applications, the amount of consolidated buffer space can be less than the combined buffers of individual systems.

### 7.3.1 System capacity sizing steps for server consolidation

One method of estimating the required system capacity for server consolidation is explained here. It should be noticed that this is just one of many methods of system sizing, and that the method explained here may not be applicable to all cases. Basically, this method is based on the highest peak load of the monitored application. It is assumed that each existing application is running on its dedicated system, and WLM is not active.

#### 7.3.1.1 Step 1 - Monitor resource usage

First, monitor for a sufficiently long period, using standard AIX performance monitoring tools such as `vmstat`, to get a distribution of workload load levels. The maximum load is an important statistic. A second important statistic is the average load exclusive of peak loads (for instance, 0-5 percent or 20 percent versus 80 percent peak load). Each of these levels has to be described according to their period and distribution over the day, week, and month.

Wherever possible, identify patterns related to the business cycle (Monday, Friday, weekend, end of month, end of quarter, end of business year). For example, in the banking business there can be some days in a month on which the systems are used much more than on others.

The existing systems may be underutilized or overutilized. If the system is overutilized (the application requires more resource than is available in the current system), you cannot obtain the exact value of the highest peak load for that application. In that case, a test system with a larger capacity may be used, or the theoretical peak load has to be extrapolated using the monitored data.

As a result, a resource usage data table, such as the one in Appendix E, "Sample for CPU resource usage calculation" on page 311, can be obtained.

It is recommended that you draw a graph, such as the one shown in Figure 84 on page 210, for each application using the resource usage data.

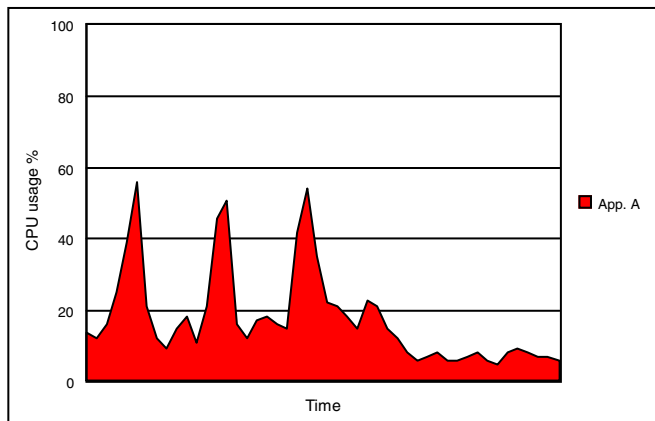


Figure 84. Peak load single application

### 7.3.1.2 Step 2 - Estimate the requirements for each application

The calculations to be done for such an estimation are:

- Minimum required system capacity (AR)
- Resource Utilization Percentage (RUP)
- Average resource utilization percentage (ARUP)

#### ***Minimum required system capacity***

For a consolidated system, first build a table without regard to buffering.

The system sizing buffer is an estimate of the additional resources needed to handle:

1. Concurrent critical applications *growth*
2. Concurrent (though lower priority) resources for other workloads during critical application *peak* load requirements.

The minimum required system capacity for each application is calculated by adding the estimated buffer to the highest peak load observed.

The *minimum required system capacity*, which is used in this example as the *total available system resource*, is calculated with the following formula:

$$AR = \frac{HP \times (100 + BF)}{100}$$



AR = Minimum Required System Capacity; is used as the Total Available System Resource.

HP = The highest peak load.

BF = The buffering factor as a percentage of the total capacity need.

Using the data from the table in Appendix E, "Sample for CPU resource usage calculation" on page 311, Application A produces the following peak load:

$$AR = \frac{5600 \times (100 + 20)}{100} = 6720$$

Calculating the highest peak loads of all applications:

- Application A: 5,600
- Application B: 3,400
- Application C: 5,700
- Application D: 1,900

Assume the capacity of the system on which these individual applications are running is 10,000 tpm (transactions per minute). Because the system capacity is 10,000 tpm, each percentage value in the graphs is easily converted, by multiplying the actual tpm value that was consumed by each application at the moment of measurement by 100.

The minimum required system capacity for each of the applications, based on the highest peak loads with a moderate buffering factor of 20 percent, would be:

- Application A:  $5600 \times 1.2 = 6,700$  tpm
- Application B:  $3400 \times 1.2 = 4,100$  tpm
- Application C:  $5700 \times 1.2 = 6,800$  tpm
- Application D:  $1900 \times 1.2 = 2,300$  tpm

The values below one hundred are rounded.

If these four applications are run on four individual servers dedicated to each application, the total CPU power needed for these four applications will add up to 19,900 tpm (Figure 85 on page 212).

Total TPM consumption = the sum of TPM consumption of individual systems  
 $= 6,700 + 4,100 + 6,800 + 2,300 = 19,900$

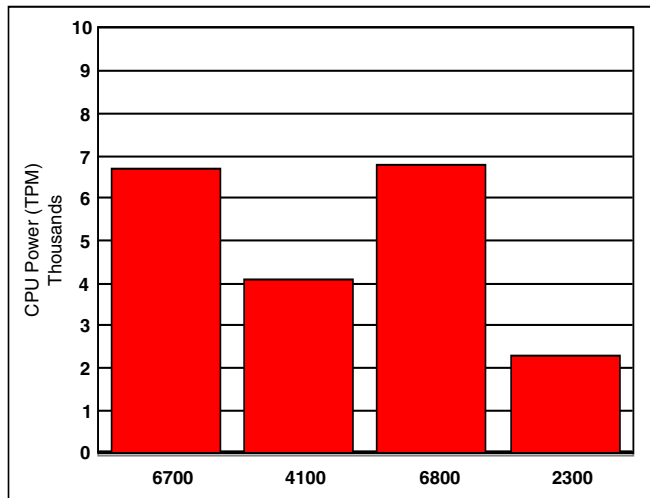


Figure 85. Total TPM consumption

### **Resource Utilization Percentage**

The *Resource Utilization Percentage (RUP)* can be calculated using the following formula:

$$RUP = \frac{(UR \times LTU)}{(AR \times TU \times LTU)} \times 100$$

UR = Actually used resource during the period (colored area under the usage curve of the example graph). UR can be calculated by adding the values of the resource usage measured at each measuring point.

AR = Total Available System Resource calculated earlier as the Minimum Required System Capacity (total area of the example graph).

TU = Number of time units during the monitoring period.

LTU = Length of Time Unit in seconds. If the monitoring interval is, for instance, set to ten seconds, the Length of Time Unit (LTU) is 10.

Using the resource usage graph displayed in Figure 84 on page 210 the *Resource Utilization Percentage (RUP)* can be calculated as follows:

$$RUP = \frac{(86800 \times 10)}{(6700 \times 500 \times 10)} \times 100 = 26$$

The overall CPU utilization percentages of each application that runs on its dedicated individual system has the minimum required system capacity calculated above are calculated as follows:

Resource utilization percentage =  $(UR / (AR \times TU)) \times 100$

- Application A:  $(86800/(6700 \times 50)) \times 100 = 26$  percent
- Application B:  $(111600/(4100 \times 50)) \times 100 = 54$  percent
- Application C:  $(73600/(6800 \times 50)) \times 100 = 22$  percent
- Application D:  $(74500/(2300 \times 50)) \times 100 = 65$  percent

Notice that the less variance the CPU resource utilization pattern shows along with time, the higher overall resource utilization percentage we get.

### **Average resource utilization percentage**

The overall average of the resource utilization percentage of the multiple systems can be calculated using the following formula:

$$ARUP = \frac{SUR}{(SAR \times TU)} \times 100$$

SUR = Sum of actually-used resources per system during the measuring period accommodating all the applications on one system.

This value is obtained by adding up the values of each system's Total Actually Used Resource (UR), and is the sum of the colored areas under the usage curves of the graphs in the example (Figure 84 on page 210).

SAR = Sum of total available resources of all the systems, or the sum of the total required system capacity for accommodating all the applications on one system. This value is obtained by adding up the values of each system's Minimum Required System Capacity (AR), and is the sum of total areas of the graph boxes in the example (Figure 85 on page 212).

TU = The number of time units during the monitoring period.

The average resource utilization percentages of the four systems are calculated as follows:

$$ARUP = \frac{86800 + 111600 + 73600 + 74500}{(6700 + 4100 + 6800 + 2300) \times 50} \times 100 = 35$$

### **7.3.1.3 Estimate the capacity for integrated applications**

In this step, the minimum required capacity of a single system required for integrated applications is estimated.

Taking the sum of individual resource usage values of all the applications at one of the measurement points gives the expected resource usage value of the applications integrated into one system at the same measurement point. Repeating this at all measurement points produces a table of the expected resource usage data when the applications are integrated into one system, such as the one that is obtained for each separate application by actual monitoring in Section 7.3.1.1, “Step 1 - Monitor resource usage” on page 209.

An expected resource usage graph, such as the one shown in Figure 92 on page 219, can be obtained from this.

The minimum required capacity and the resource utilization percentage for integrated applications are calculated as described in Section 7.3.1.1, “Step 1 - Monitor resource usage” on page 209.

## **7.3.2 Examples**

The following examples give a good illustration of the capacity usage benefit using the WLM solution.

The resource usage data table used in these examples is available in Appendix E, “Sample for CPU resource usage calculation” on page 311. The time unit used in the table is 10 minutes, and the number of this time unit monitored here is 50. Thus, the total monitoring duration is 500 minutes. It should be noticed that the minimum monitoring period has to be at least 24 hours in actual cases. The length of 500 minutes is used here just for simplicity of the example.

The examples here are CPU resource only. Considerations for memory and disk I/O bandwidth are discussed in Section 7.3.3, “Considerations for memory and disk I/O bandwidth” on page 221.

### **7.3.2.1 Base line - Applications running on separate systems**

For example, assume that there are four different applications that have the CPU usage patterns shown in the following four figures.

Application A, shown in Figure 86 on page 215, exhibits short, pronounced peak loads.

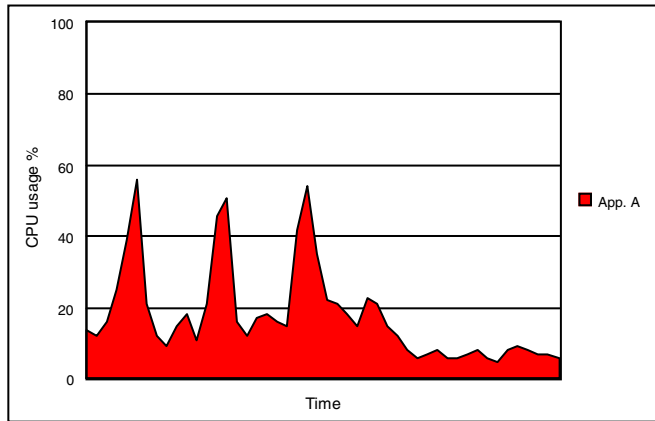


Figure 86. CPU peak load of Application A

Application B, shown in Figure 87, shows workload increasing and decreasing gradually over time.

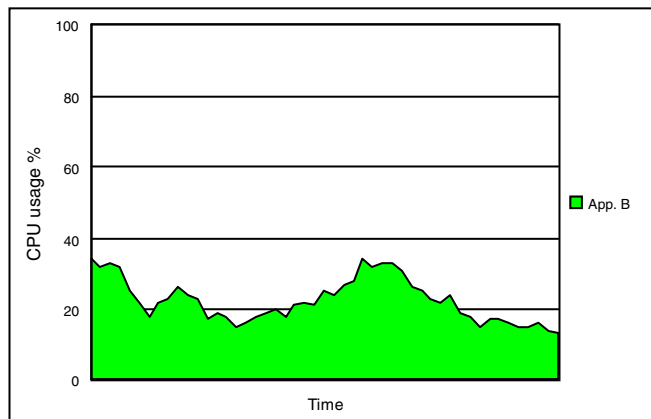


Figure 87. CPU usage pattern of Application B

Application C, shown in Figure 88 on page 216, is a good example of a nightly batch job.

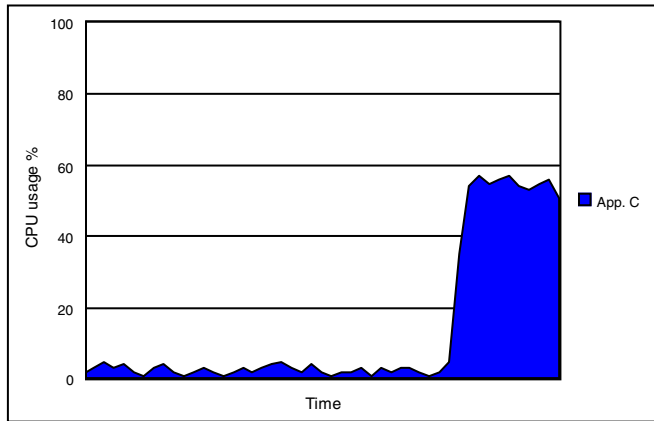


Figure 88. CPU usage pattern of Application C

Application D, shown in Figure 89, has a comparatively flat, constant resource usage pattern.

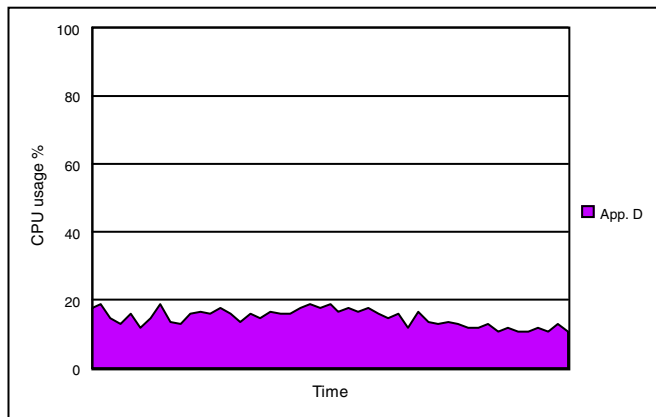


Figure 89. CPU usage pattern of Application D

### 7.3.2.2 Approach 1 - All applications are mission-critical

Now, consider using WLM to integrate the four applications on a single server. It is assumed that WLM can address all the obstacles against the application integration on a single system. Then, the usage pattern shown in Figure 90 on page 217 is obtained.

In this case, the minimum required system capacity for the integrated applications based on the highest peak load, with the same buffering factor of 20 percent as before, is estimated as follows:

- The highest peak load in Figure 90 is 9700.
- The minimum required capacity = 9700 X 1.2 = 11,600 tpm

The overall CPU usage percentage on the server of this capacity during the given time span would be:

$$\text{Resource utilization percentage} = (\text{UR} / (\text{AR} \times \text{TU})) \times 100$$

See Section 7.3.1.2, “Step 2 - Estimate the requirements for each application” on page 210 for detailed information about this calculation.

Resource utilization percentage

$$= (86800 + 111600 + 73600 + 74500) / (11600 \times 50) \times 100 = 60 \text{ percent}$$

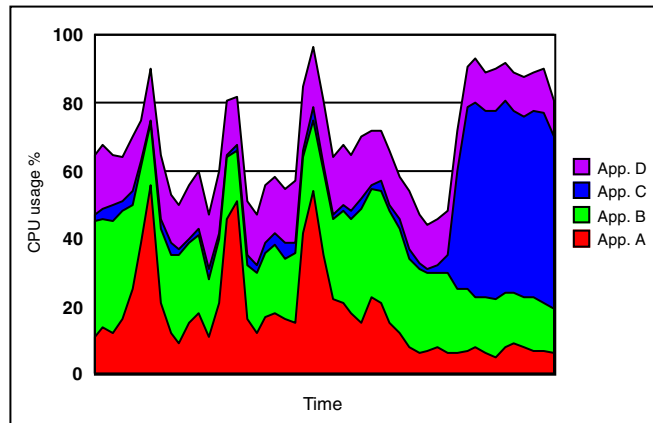


Figure 90. CPU usage pattern of applications integrated on a single serve

### 7.3.2.3 Approach 2 - Only some applications are mission-critical

The capacity usage benefit of WLM becomes manifest when some of the integrated applications are not mission-critical. If WLM is not used, the system does not offer any practical method to give the higher priority to the more important applications. As a consequence, if system resources are running short, all applications will contend for them, thus hurting the performance of all applications (Figure 91 on page 218). To guarantee the performance of some mission-critical applications, the required system capacity has to be estimated based on the top peak load, usually with some

percentage of buffer capacity in case of unexpected heavy workloads, even if their duration is short.

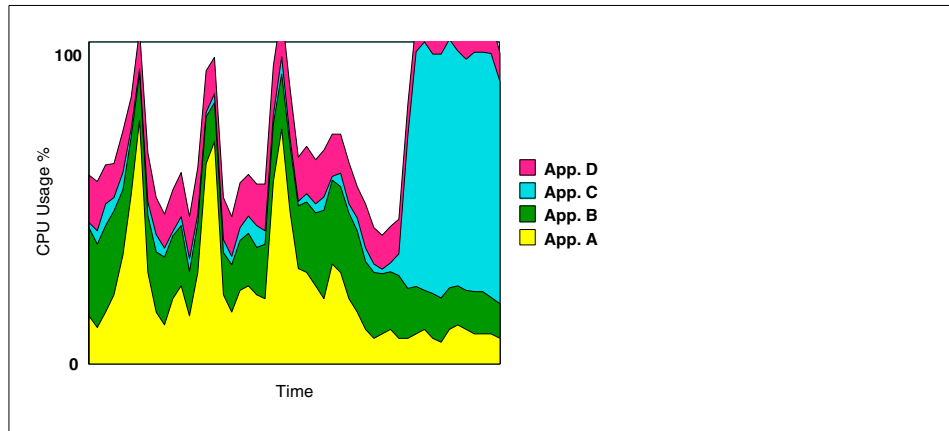


Figure 91. Server consolidation - system resources running short

The required system capacity can be reduced using WLM if the performance of some of the integrated applications is not important. WLM can effectively control the resource allocation to each application, with its shares, limits, and tiers, to guarantee the performance of mission-critical applications. Of course, this makes sense only if the performance degradation of the other applications is acceptable to the business.

For example, assume that Application B and Application D (Figure 92 on page 219) do not require prompt response or output and that only the response time of Application A and the processing time of Application C are important (Figure 93 on page 219). Then the required capacity is estimated (with a generous buffering factor of 20 percent) as follows:

The required capacity

$$= (\text{the top peak of (Application A + Application C)}) \times 1.2$$

$$= 5,600 \times 1.2 = 6,700$$



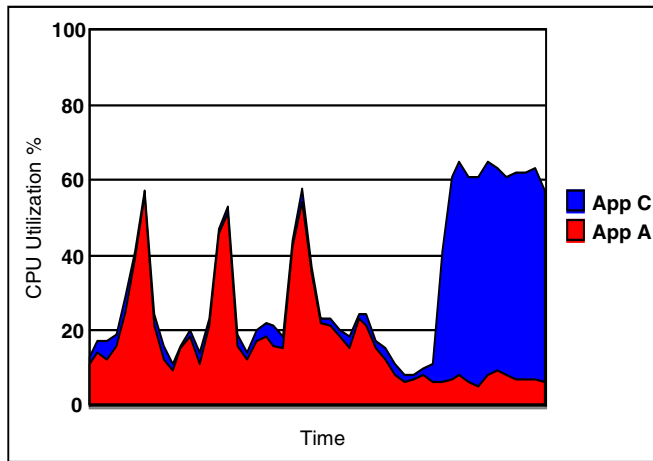


Figure 92. Consolidation of Application A and Application C

The required capacity

$$= (\text{the top peak of (Application B + Application D)}) \times 1.2$$

$$= 5,700 \times 1.2 = 6,800 \text{ tpm}$$

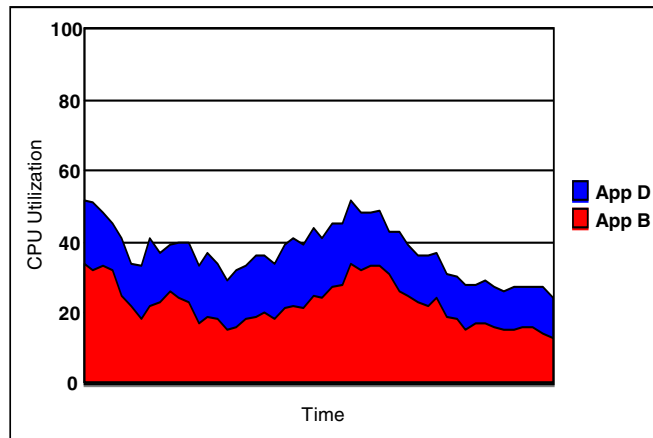


Figure 93. Consolidation of Application B and Application D

Because there are several points at which the total required CPU resource exceeds this value without WLM, all the applications will be slowed down. However, by using WLM and placing Application A and Application C in a higher tier than the others, we can isolate the important applications from the

others. At those points where resource is running short, only Application B and Application D are slowed down, which is acceptable to the overall business operation.

In this case, the overall resource utilization percentage is calculated as follows:

$$\text{Resource utilization percentage} = (\text{UR} / (\text{AR} \times \text{TU})) \times 100$$

See Section 7.3.1.2, “Step 2 - Estimate the requirements for each application” on page 210 for detailed information about this calculation.

$$\text{Resource utilization percentage} = ((86800 + 111600 + 73600 + 74500) / (9100 \times 50)) \times 100 = 76 \text{ percent}$$

#### 7.3.2.4 Comparison of the cases

You can clearly see the capacity usage benefit of server consolidation using WLM in Table 4 on page 220.

If you use four individual systems for your applications, you have to pay for four systems with the total capacity of 19,900 tpm, and you will be using only 35 percent of the total available resource. However, if you decide to integrate the applications into one system using WLM, you will need a system of 11,600 tpm, and the overall utilization will be up to 60 percent. Granted that only the performance of Application A and Application C is important, you can cut the estimate down to 9,100 tpm, even with a generous buffering factor of 40 percent. The overall utilization will be as high as 76 percent.

Table 4. Comparison of individual application systems and one integrated system

	Required capacity (tpm)	Overall utilization (percent)	Remarks
Application A	6,700	26	Pronounced, short peaks in resource usage pattern
Application B	4,100	54	Moderate peaks
Application C	6,800	22	Nightly batch
Application D	2,300	65	The most even resource usage pattern
Sum of A,B,C, and D	19,900	35	Total, and average of the four systems

	Required capacity (tpm)	Overall utilization (percent)	Remarks
Integrated applications	11,600	60	All four applications integrated on one server, allowing enough space for each application.
Applications B and D are considered non-critical	9,100	76	There are some points where Applications B and D are slowed down

There are several points that you have to consider before estimating the required system capacity when using AIX WLM:

1. AIX WLM can help improve the overall resource utilization percentage, thus reducing the required system capacity.
2. AIX WLM can be helpful in improving system capacity usage, especially when the resource usage patterns of the applications are quite different from one another.
3. It is recommended that you integrate mission-critical applications with non-critical ones on one system to get the maximum benefit from using WLM.
4. If the overall resource utilization percentages of the individual application servers are already good, for example, more than 70 percent, and you want to guarantee the performance of all the applications to be integrated into one system, there would be only a little gained in system capacity by using AIX WLM.

Thus, it is very important to have a well-designed plan on the grouping and deployment of different applications to get the expected improvement. For example, it would be a better idea to integrate Application A and Application C (Figure 92 on page 219), which have different peak time and behavior on one system than to integrate Application B with Application D (Figure 93 on page 219), both of which have rather constant, even resource utilization patterns. Often, it is more important to make a right selection of applications to be integrated than to make good property files for WLM configurations.

### 7.3.3 Considerations for memory and disk I/O bandwidth

Basically, the same methodology can be used to estimate the capacity of memory and disk I/O bandwidth resources as that used to estimate CPU

resource. However, special care should be taken when estimating the required capacity of memory because this is, by nature, not a *renewable* resource, as opposed to CPU, meaning that AIX might first have to take actions in order to provide the application with memory (for instance, freeing up memory pages by paging out the pages that another application is using).

The performance of mission-critical classes can be protected from memory swapping to or from paging spaces by setting generous minimum limits for them and/or placing those classes in a higher tier than the others. The system-defined classes, such as *Shared* and *System*, should be given enough minimum limits to ensure overall constant performance. However, the overall system performance might be degraded when some processes in one class begin to swap to or from paging spaces. It is recommended that you use a more conservative estimation for memory capacity sizing than for CPU capacity sizing.

It certainly helps to guarantee the performance of mission-critical applications by entitling more disk I/O bandwidth to them than to non-critical ones. However, in most situations, it is difficult to trace which process is using which disk for which logical volume. Thus, it is not easy to estimate the capacity usage benefit by using WLM.

**Note**

CPU time and disk I/O bandwidth are considered renewable system resources.

---

## 7.4 Conclusion

AIX WLM can reduce the required minimum system capacity for applications by enhancing the overall system resource utilization. However, there is no committed capacity gain from using AIX WLM. Only by selecting the right set of applications to be integrated on a single system and by correct planning of the WLM configuration can you benefit from WLM in terms of system capacity usage. It is recommended that you set up the consolidation plan after monitoring the resource utilization pattern of each application.

---

## Chapter 8. Practical experience

This chapter reflects some practical experiences with WLM. The ISV case studies were based on AIX Version 4.3.3 Maintenance levels 2 and 6; therefore, the latest features of WLM with AIX 5L could not be tested. Readers should be aware of this fact and conduct their own experiments after studying the results of this chapter.

Section 8.2, “Customer experience - WLM and a compute server for research” on page 252 reflects some experiences with WLM in a production environment at the Forschungszentrum Jülich GmbH (Research Center Jülich) in Germany, from the perspective of a system administrator.

---

### 8.1 ISV case studies

The case studies have been set up in the PeopleSoft ISV Lab in Austin, Texas USA, and in the IBM SAP International Competence Center (ISICC) in Walldorf, Germany.

The goal of the case studies was to see the effect of various WLM configurations on the different scenarios described in the following sections.

Be aware that the case studies did not focus on tuning the results to optimal performance.

#### 8.1.1 PeopleSoft

The idea of this case study was to run four concurrent PeopleSoft benchmark kits in different combinations and different WLM configurations:

- PeopleSoft General Ledger (GL)
- PeopleSoft Payroll (PAYROLL)
- PeopleSoft Financial (FI)
- PeopleSoft Human Resources (HR)

GL and PAYROLL are batch benchmarks.

FI and HR are online transaction processing (OLTP) benchmarks.

The primary concern was to demonstrate that one class, such as batch, with high CPU requirements, does not dominate response time for interactive/OLTP workloads.

Because 32bit Oracle was run, it was decided to create four independent databases with four Oracle listener processes to improve performance. By doing this, the total System Global Area (SGA) size for all four benchmarks was about 10 GB, whereas, with a single database, the limit is about 2.5 GB.

#### **8.1.1.1 Case study description**

The OLTP benchmarks were run in a logical three tier configuration. With this setup, the number of users in the load was reduced. This means that the database server and application server were installed on the same host.

In the OLTP benchmarks, average retrieve and update response times were measured for an individual client with 1250 FI and 6000 HR concurrent users.

Mercury Interactive's LoadRunner was used to simulate concurrent users. For the FI benchmark, it submitted a business transaction at an average rate of five transactions per second. For the HR benchmark, it submitted a business transaction at an average rate of 13 transactions per second.

SQA Robot was used to automatically submit transactions and record the benchmark measurements on the client PCs. Measurements were recorded when the user load was attained and the environment reached the steady state.

Batch processes are background processes requiring no operator intervention or interactivity. Results of these processes are automatically logged in the database. The runtimes are posted to the Process Request database table. Both batch benchmark processes were initiated at the client workstations. For these benchmarks, all jobs were started from MicroFocus COBOL 4.1 script files executed on the RS/6000 S80 server (see Table 5 on page 227).

In PeopleSoft General Ledger (GL), the batch performance of 40 Journal Edit processes were measured. The eight Journal Post processes were not measured.

The Journal Edit process validates journal entries including items, such as ChartField values, control totals, and debit/credit balancing.

The Journal Post process summarizes detail line activity and either inserts a new row or updates an existing row in the ledger. There is one ledger row for each unique combination of ChartField values, accounting period, and fiscal year. In this benchmark, the Post step updated only existing ledger rows. This is typical for companies that perform the edit and post functions on a frequent

basis. The database model represented an extra large organization that processes 3,000,000 journal transactions per run.

The PeopleSoft Payroll benchmark commits 32 jobs. Each of the jobs has three phases; creation, calculation, and confirmation.

The Paysheet Creation process generates payroll data worksheets for employees consisting of standard payroll information for each employee for the given pay cycle. This process ran separately from the other two tasks and was not measured.

The Payroll Calculation process looks at paysheets and calculates checks for those employees. Payroll Calculation can be run any number of times throughout the pay period. The first run does most of the processing while each successive run updates only the calculated totals of changed items. This interactive design minimizes the time required to calculate a payroll as well as the processing resources required. In this benchmark, Payroll Calculation was run only once as though it was the end of a pay period.

The Payroll Confirmation takes the information generated by Payroll Calculation and updates the employees' balances with the calculated amounts. The system assigns check numbers at this time and creates direct deposit records. Confirm can only be run once and, therefore, must be run at the end of the pay period. Only the last two phases were measured. The database model represented a large organization with 72,000 employees.

#### **8.1.1.2 Case study method**

First, each of the benchmarks was run individually on a six-way and 24-way RS/6000 S80 to establish the baseline. In this step, WLM was inactive.

##### **Note**

The results of the six-way baseline are not listed for all the tests described in this chapter.

Then six WLM control files were set up to get a baseline running WLM in passive mode (see Section 8.1.1.3, "WLM configuration" on page 228). Running WLM in passive mode allows you to observe class resource allocations without actually incurring any WLM adjustment. The observed results were used as guidelines for setting up shares and limits in the WLM control files.

After getting a baseline running WLM in passive mode, the benchmarks were started with WLM in active mode. The goal of these runs was to make the two

OLTP benchmarks work better in the consolidated server without much regard for the two batch benchmarks.

Both OLTP benchmarks ran with the GL batch and also with the PAYROLL batch (see Section 8.1.1.4, “One batch - Two OLTP benchmarks: PAYROLL-FI-HR” on page 232, and Section 8.1.1.5, “One batch - Two OLTP benchmarks: GL-FI-HR” on page 234).

Both batch benchmarks ran with two different WLM configuration files with no OLTP benchmark (see Section 8.1.1.6, “Two batch benchmarks: GL-PAYROLL” on page 235).

Finally, all four benchmarks ran with four different WLM configurations (see Section 8.1.1.7, “Two batch - Two OLTP benchmarks: PAYROLL-GL-FI-HR” on page 235). Figure 94 shows the HR OLTP benchmark environment.

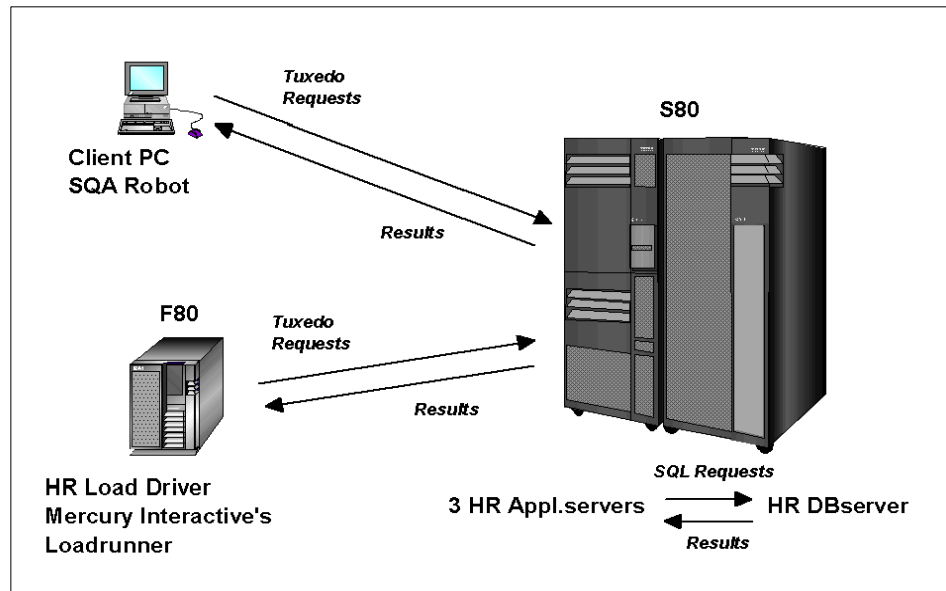


Figure 94. HR OLTP benchmark

The HR OLTP Mercury scripts were started on an RS/6000 F80. Three application server domains, each with 2000 users, ran on the RS/6000 S80. Figure 95 on page 227 shows the FI OLTP benchmark environment.



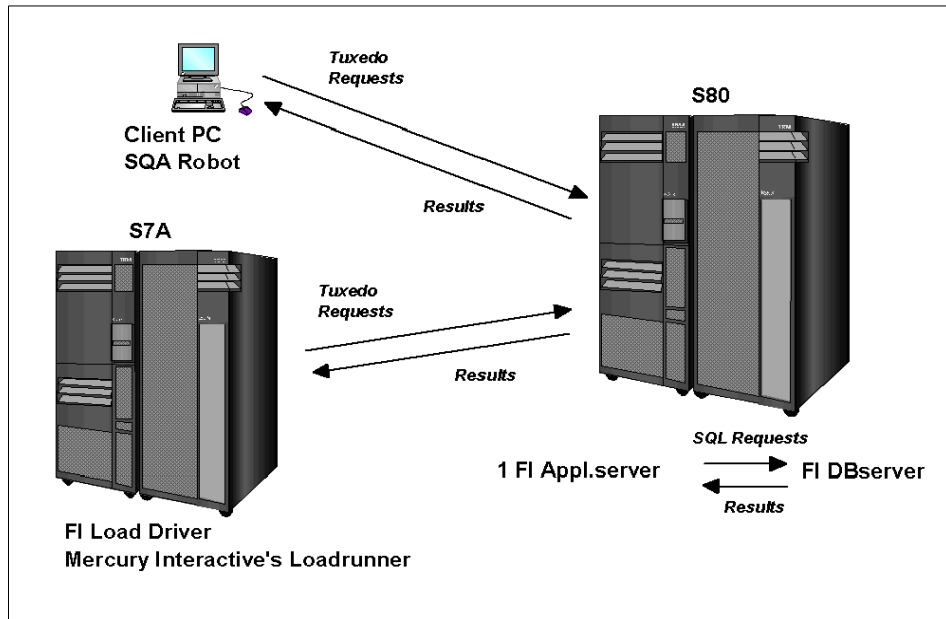


Figure 95. FI OLTP benchmark

The FI OLTP Mercury scripts were started on an RS/6000 S7A. One application server domain with 1250 users ran on the RS/6000 S80.

Table 5 gives a list of the hardware configuration used for this case study.

Table 5. PeopleSoft case study HW configuration

Function	Model	CPU	Memory
DB and AP Server (logical 3-tier)	RS/6000 S80	24 Way	32 GB
HR Load Driver	RS/6000 F80	6 Way	16 GB
FI Load Driver	RS/6000 S7A	12 Way	16 GB
2 x Display Server	RS/6000 B50	1 Way	1 GB

Additionally, four PC clients were used as shown in Table 6.

Table 6. PeopleSoft case study PC HW configuration

Function	Clock speed	CPU	Memory
FS Client	400 MHz	1 Way	64 MB
HR Client	166 MHz	1 Way	64 MB
PAY Client	180 MHz	1 Way	112 MB
GL Client	180 MHz	1 Way	80 MB

The following is a list of the software used for this case study:

- AIX 4.3.3 Maintenance level 2
- PeopleSoft Financials 7.52
- PeopleSoft Payroll 7.50
- PeopleSoft General Ledger 7.50
- PeopleSoft HRMS 7
- PeopleTools 7.55
- Oracle 8.0.5.1
- BEA TUXEDO 6.4 and 6.5
- Micro Focus COBOL 4.1
- SQR 4.3.2
- Mercury Interactive's LoadRunner 5.02
- Microsoft Windows NT 4.0
- SQA Suite Robot 6.1.0.42
- PAY Client: PT 7.58
- GL Client: PT 7.54.1
- FI Client: PT 7.55
- HR Client: PT 7.54.1

### 8.1.1.3 WLM configuration

Several WLM configurations were tested with various share and limit combinations.

Eight classes were active. Four classes were supplied by WLM (Unclassified, Shared, System, and Default). Four classes were configured by the benchmark team (pay, gl, fs, and hr).

For the two pseudo-classes (Unclassified and Shared) no classification rules, resource limits, or resource shares can be specified. These classes are outside WLM control and, therefore, fall under default AIX resource allocation control.

A unique user ID was created for each of the four databases. Each user belongs to the DBA group.

The WLM configuration is described in the following tables.

The WLM configuration, p\_conf\_1 (Table 7):

Table 7. p\_conf\_1

Tier	Class	User	CPU	Memory
1	pay	pay750	min=0 max=100 share=1	min=0 max=100 share=1
1	gl	gl75	min=0 max=100 share=1	min=0 max=100 share=1
0	fs	fs75	min=0 max=100 share=1	min=0 max=100 share=1
0	hr	hr75	min=0 max=100 share=1	min=0 max=100 share=1
0	System	root	min=0 max=100 share=1	min=0 max=100 share=1
0	Default	-	min=0 max=100 share=1	min=0 max=100 share=1

WLM configuration, p\_conf\_2 (Table 8):

Table 8. p\_conf\_2

Tier	Class	User	CPU	Memory
1	pay	pay750	min = 0 max = 25 share = 25	min = 0 max = 100 share = 1
1	gl	gl75	min = 0 max = 25 share = 25	min = 0 max = 100 share = 1
1	fs	fs75	min = 0 max = 100 share = 25	min = 0 max = 100 share = 1
1	hr	hr75	min = 0 max = 100 share = 25	min = 0 max = 100 share = 1
0	System	root	min = 0 max = 100 share = 1	min = 1 max = 100 share = 1
0	Default	-	min = 0 max = 100 share = 1	min = 0 max = 100 share = 1

WLM configuration p\_conf\_3 (Table 9):

Table 9. p\_conf\_3

Tier	Class	User	CPU	Memory
7	pay	pay750	min = 0 max = 100 share = 1	min = 0 max = 100 share = 1
7	gl	gl75	min = 0 max = 100 share = 1	min = 0 max = 100 share = 1
1	fs	fs75	min = 0 max = 100 share = 1	min = 0 max = 100 share = 1
1	hr	hr75	min = 0 max = 100 share = 1	min = 0 max = 100 share = 1

Tier	Class	User	CPU	Memory
0	System	root	min = 0 max = 100 share = 1	min = 0 max = 100 share = 1
0	Default	-	min = 0 max = 100 share = 1	min = 0 max = 100 share = 1

WLM configuration, p\_conf\_4 (Table 10):

Table 10. p\_conf\_4

Tier	Class	User	CPU	Memory
1	pay	pay750	min = 0 max = 100 share = 50	min = 0 max = 100 share = 1
1	gl	gl75	min = 0 max = 100 share = 50	min = 0 max = 100 share = 1
0	System	root	min = 0 max = 100 share = 1	min = 0 max = 100 share = 1
0	Default	-	min = 0 max = 100 share = 1	min = 0 max = 100 share = 1

WLM configuration p\_conf\_5 (Table 11):

Table 11. p\_conf\_5

Tier	Class	User	CPU	Memory
1	pay	pay750	min = 0 max = 100 share = 32	min = 0 max = 100 share = 1
1	gl	gl75	min = 0 max = 100 share = 40	min = 0 max = 100 share = 1
0	System	root	min = 0 max = 100 share = 1	min = 0 max = 100 share = 1

Tier	Class	User	CPU	Memory
0	Default	-	min = 0 max = 100 share = 1	min = 0 max = 100 share = 1

WLM configuration p\_conf\_6 (see Table 12):

Table 12. p\_conf\_6

Tier	Class	User	CPU	Memory
1	pay	pay750	min = 0 max = 100 share = 10	min = 0 max = 100 share = 1
1	gl	gl75	min = 0 max = 100 share = 10	min = 0 max = 100 share = 1
1	fs	fs75	min = 0 max = 100 share = 40	min = 0 max = 100 share = 1
1	hr	hr75	min = 0 max = 100 share = 40	min = 0 max = 100 share = 1
0	System	root	min = 0 max = 100 share = 1	min = 1 max = 100 share = 1
0	Default	-	min = 0 max = 100 share = 1	min = 0 max = 100 share = 1

#### 8.1.1.4 One batch - Two OLTP benchmarks: PAYROLL-FI-HR

Table 13 presents the process results of the two OLTP and one batch benchmark.

Table 13. PAYROLL-FI-HR

Application	Measured data	24-way baseline	WLM passive	WLM active with p_conf_3
PAYROLL (batch)	Calc+Cnfrm/Hr	407,932	227,488	241,530
	Percent CPU	74	41	42

Application	Measured data	24-way baseline	WLM passive	WLM active with p_conf_3
FI (OLTP)	Average Ret	0.530	0.593	0.586
	Average Updte	0.755	1.002	0.868
	Overall Avrge	0.579	0.682	0.647
	Percent CPU	31	27	26
	TPM	257	262	264
HR (OLTP)	Average Ret	0.870	1.254	0.949
	Average Updte	0.672	0.779	0.736
	Overall Avrge	0.791	0.917	0.864
	Percent CPU	26	26	24
	TPM	922	911	897

For the batch benchmark, it shows the number of employees processed per hour for the Calculation and Confirmation phases, and the CPU utilization in percent.

For the OLTP benchmarks, it displays the average retrieval time in seconds, the average update time in seconds, the overall average time in seconds, the CPU utilization in percent, and the number of transactions per minute (TPM).

**Observations:**

- The performance of FI and HR OLTP benchmarks improves when activating WLM.
- The performance of PAYROLL batch benchmark also improves when activating WLM.

### 8.1.1.5 One batch - Two OLTP benchmarks: GL-FI-HR

Table 14 presents the process results of the two OLTP and one batch benchmark.

Table 14. GL-FI-HR

Application	Measured data	24-way baseline	WLM passive	WLM active with p_conf_3
GL (batch)	Edit	13,088,434	7,680,491	8,125,457
	Percent CPU	82	48	50
FI (OLTP)	Average Ret	0.530	0.703	0.618
	Average Updte	0.755	1.285	0.994
	Overall Avrge	0.579	0.829	0.700
	Percent CPU	31	25	24
	TPM	257	261	261
HR (OLTP)	Average Ret	0.870	1.085	0.972
	Average Updte	0.672	0.867	0.740
	Overall Avrge	0.791	0.998	0.879
	Percent CPU	26	25	24
	TPM	922	911	909

For the batch benchmark, it shows the number of journal lines processed per hour in the Edit phase and the CPU utilization in percent.

For the OLTP benchmarks, it displays the average retrieval time in seconds, average update time in seconds, overall average time in seconds, CPU utilization in percent, and the transactions per minute (TPM).

#### Observations:

- The performance of FI and HR OLTP benchmarks improves when activating WLM.
- The performance of GL batch benchmark also improves when activating WLM.



### 8.1.1.6 Two batch benchmarks: GL-PAYROLL

Table 15 presents the process results of the two batch benchmarks with different WLM configurations.

Table 15. GL-PAYROLL

Application	Measured data	24-way baseline	WLM passive	WLM active with p_conf_4	WLM active with p_conf_5
GL (batch)	Edit	13,088,434	9,237,306	9,285,051	9,021,199
	Percent CPU	82	58	58	58
PAYROLL (batch)	Calc+Cnfrm/Hr	407,932	237,938	263,833	245,399
	Percent CPU	74	39	42	39

For the GL benchmark, it presents the number of journal lines processed per hour in the Edit phase and the CPU utilization in percent.

For the PAYROLL benchmark, it presents the number of employees processed per hour for the Calculation and Confirmation phases, and the CPU utilization in percent.

#### Observations:

The performance of GL batch benchmark with 40-32 shares (p\_conf\_5) has a worse result than the passive or equal shares (p\_conf\_4) run.

### 8.1.1.7 Two batch - Two OLTP benchmarks: PAYROLL-GL-FI-HR

Table 16 shows the two batch and two OLTP process results with different WLM configurations.

Table 16. PAYROLL-GL-FI-HR

Application	Measured data	24-way baseline	WLM passive	WLM active with p_conf_3	WLM active with p_conf_6
PAYROLL (batch)	Calc+Cnfrm/Hr	407,932	155,072	145,425	148,423
	Percent CPU	74	24	22	23
GL (batch)	Edit	13,088,434	5,244,846	5,027,567	5,285,877
	Percent CPU	82	29	30	30

Application	Measured data	24-way baseline	WLM passive	WLM active with p_conf_3	WLM active with p_conf_6
FI (OLTP)	Average Ret	0.530	0.821	0.625	(not measured)
	Average Updte	0.755	1.933	1.075	(not measured)
	Overall Avrge	0.579	1.062	0.723	0.738
	Percent CPU	31	24	24	24
	TPM	257	261	262	262
HR (OLTP)	Average Ret	0.870	1.249	1.080	(not measured)
	Average Updte	0.672	1.013	0.821	(not measured)
	Overall Avrge	0.791	1.154	0.976	0.973
	Percent CPU	26	23	23	23
	TPM	922	901	905	904

For the GL benchmark, it shows the number of journal lines processed per hour in the Edit phase and the CPU utilization in percent.

For the PAYROLL benchmark, it displays the number of employees processed per hour for the Calculation and Confirmation phases, and the CPU utilization in percent.

For the OLTP benchmarks, it displays the average retrieval time in seconds, the average update time in seconds, the overall average time in seconds, the CPU utilization in percent, and the transactions per minute (TPM).

**Observations:**

- FI OLTP benchmark performance is best with p\_conf\_3.
- HR OLTP benchmark performance is best with p\_conf\_6.
- GL batch benchmark performance is best with p\_conf\_6.
- Payroll batch benchmark performance is best when running WLM in passive mode.

The following tables display the best results in the top row and the worst results in the bottom row.

*p\_conf\_3*

Default and System classes were in tier 0; HR and FI classes were in tier 1 to

provide better fulfillment of their resource requirements, and payroll and GL classes were in tier 7 (see Table 9 on page 230). Table 17 contains an overview of the results for p\_conf\_3.

Table 17. Overview of the results for p\_conf\_3

Payroll	GL	HR	FI
24-way baseline 407,932 emp/hr	24-way baseline 13,088,434 lines/hr	24-way baseline 0.791 sec	24-way baseline 0.579 sec
6-way baseline 158,486 emp/hr	WLM passive 5,244,846 lines/hr	WLM active 0.976 sec	WLM active 0.723 sec
WLM passive 155,072 emp/hr	WLM active 5,027,567 lines/hr	6-way baseline 0.990 sec	6-way baseline 0.756 sec
WLM active 145,425 emp/hr	6-way baseline 3,794,586 lines/hr	WLM passive 1.154 sec	WLM passive 1.062 sec

#### p\_conf\_6

Default and System classes were in tier 0, all four benchmark classes were in tier 1, and the shares were adjusted in the shares file (see Table 12 on page 232). Table 18 contains an overview of the results for p\_conf\_6.

Table 18. Overview of the results for p\_conf\_6

Payroll (batch)	GL (batch)	HR (OLTP)	FI (OLTP)
24-way baseline 407,932 emp/hr	24-way baseline 13,088,434 lines/hr	24-way baseline 0.791 sec	24-way baseline 0.579 sec
6-way baseline 158,486 emp/hr	WLM active 5,285,877 lines/hr	WLM active 0.973 sec	WLM active 0.738 sec
WLM passive 155,072 emp/hr	WLM passive 5,244,846 lines/hr	6-way baseline 0.990 sec	6-way baseline 0.756 sec
WLM active 148,423 emp/hr	6-way baseline 3,794,586 lines/hr	WLM passive 1.154 sec	WLM passive 1.062 sec

#### 8.1.1.8 Summary

Without the involvement of WLM, three out of the four workloads suffered from server consolidation, that is, running WLM in passive mode.

The CPU-intensive batch jobs dominated the usage of the CPU resource.

- Both OLTP benchmarks suffered from the server consolidation.
- The Payroll benchmark suffered from the server consolidation.
- The General Ledger benchmark benefited from the server consolidation.

Setting WLM active improves all three benchmarks that suffered before:

- The two OLTP benchmarks no longer suffer from the server consolidation. In fact, some performance gains were observed.
- The performance of the General Ledger benchmark improved.
- The Payroll benchmark's performance decreased even more.

Because the WLM configuration was only targeted to bring up the performance of online benchmarks, these results were expected.

The goals were accomplished with appropriate resource share allocation. Finer control can be further accomplished by observing the resource allocation of each class and making more adjustments.

Another attempt with OLTP benchmarks in tier 0 and batch benchmarks in tier 1 (p\_conf\_1, see Table 7 on page 229) did not accomplish the goals.

### **8.1.2 SAP R/3 Case Study**

This section is a modified excerpt from a document related to a joint project between IBM, BULL, and SAP that tested the potential of WLM based on AIX Version 4.3.3 Maintenance Level 6 in an SAP R/3 server consolidation environment.

The project was carried out by the IBM SAP International Competence Center (ISICC) in Walldorf, Germany.

This project identified a number of solution patterns that could be useful in controlling the behavior of multiple SAP R/3 systems consolidated on one server. WLM for AIX can control the CPU, memory resources, and the I/O subsystem. This study was concerned with CPU resource allocation only. In a server consolidation environment, it is crucial that a proper sizing for memory and CPU capacity has been done for each SAP R/3 system that will be housed on the machine, and that the total resources are adequate.

This chapter describes a number of successful solution patterns and describes how they were achieved. It also attempts to analyze various architectural aspects of both the SAP R/3 kernel and AIX that affect the control potential and scalability of these patterns.

Within the scope of this project, the following products were tested together with WLM:

- DB2 UDB 6.1
- Oracle 8.1.6 64bit

- SAP R/3 version 46B
- SAP R/3 version 46C

#### **8.1.2.1 SAP standard benchmark tool**

A generic standardized benchmark tool for Sales and Distribution (SD), provided by SAP, used by all SAP platform providers for platform positioning and competitive benchmarking within the SAP world.

The tool is highly OLTP, simulating many online users with quick transaction scenarios and much user context switching.

It was selected as the best tool in this scenario, as the granularity for adjustment is very fine (add or subtract a number of users), it is well known in the SAP world, and it produces a system load that is as near as possible to “exactly reproducible.” This being the case, it is easy to calibrate the expectations of a given load on a dedicated system, and then monitor the change of behavior as the system becomes more loaded with competing work.

Most of the data used in the tests to compare performance behavior comes from the SD test suite.

#### **8.1.2.2 WLM classes versus OS processes**

Under the AIX scheduler, every process is an independent schedulable entity unrelated and in competition with every other process for CPU resources. WLM introduces a means of grouping processes into resource classes. These classes or different SAP R/3 systems can be classified according to priority for resource consumption or for consumption monitoring purposes. The ability to monitor and control the total resource consumption for groups of processes makes WLM an interesting tool in a SAP R/3 server consolidation environment.

#### ***AIX CPU allocation***

The AIX kernel maintains a scheduling priority value for each thread. The priority value is a positive integer within the range from 0-127, and varies inversely with the importance of the associated thread. That is, a smaller priority value indicates a more important thread. When the scheduler is looking for a thread to dispatch, it chooses the dispatchable thread with the smallest priority value.

The CPU penalty is an integer that is calculated from the recent CPU usage of a thread. The recent CPU usage increases by one each time the thread is in control of the CPU at the end of a 10 ms clock tick, up to a maximum value

of 120. Once per second, the recent CPU usage values for all threads are reduced. The priority of a nonfixed-priority thread decreases as its recent CPU usage increases and vice versa. This implies that, on average, the more time slices a thread has been allocated recently, the less likely it is that the thread will be allocated the next time slice.

### ***WLM CPU allocation***

WLM performs CPU allocation by manipulating the priorities of the competing processes on the basis of class resource consumption rather than individual process consumption. When a class exceeds its resource allocation, the processes comprising the class are degraded in priority by a value determined by WLM according to the class' relative resource allocation. That is, if there is no concurrent competing classes need of resources, it is not degraded. If another class is active, then WLM will degrade a class in relation to its resource allocation and that of the other class(es). WLM modifies the penalty incurred by the threads (or processes) according to their resource allocation. The highest priority is achieved by the lowest penalty (which is 0), and the lowest priority is achieved by the highest penalty, (which is 66 in AIX Version 4.3.3). The largest unequal differentiation using WLM is achieved with one class having a penalty of 0 and another class having a penalty of 66.

SAP R/3 is a complex multi-process application with medium to very large memory requirements operated in client-server environments. Each SAP R/3 system consists of at least one SAP R/3 application server central instance, 0 -  $n$  additional application server instances, and an RDBMS database system. Each of these components comprises many processes. Figure 96 on page 241 depicts the basic SAP R/3 components. The first application server instance, referred to as the CI or central instance, contains several global components required by all others. Additional application servers can be added as needed. All these different components can be located on one host machine (referred to as a central system) or each on a separate host. All of these components together make up an SAP R/3 system.

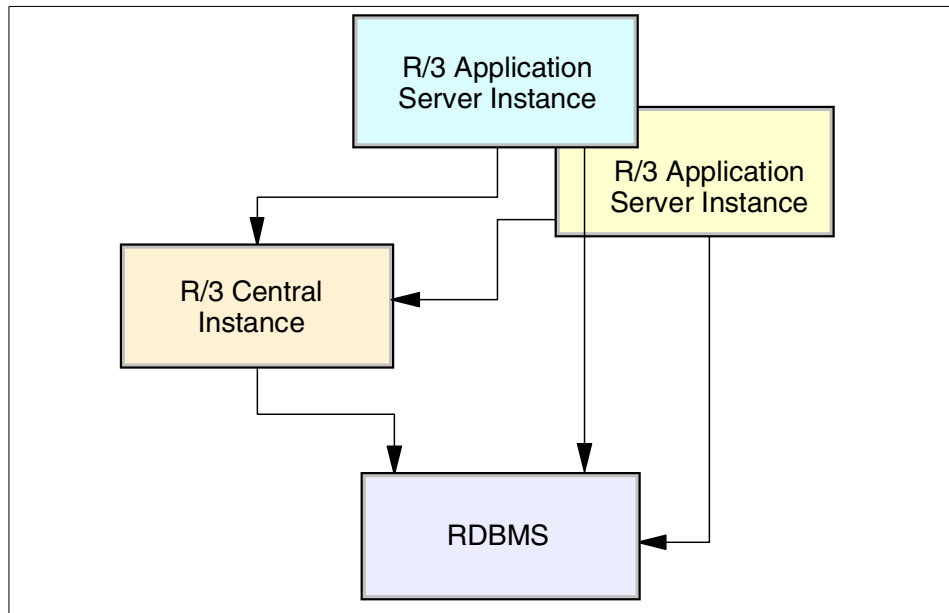


Figure 96. SAP R/3

A SAP R/3 central system (Database and Application server CI) commonly consists of more than 50 processes (Figure 97 on page 242).

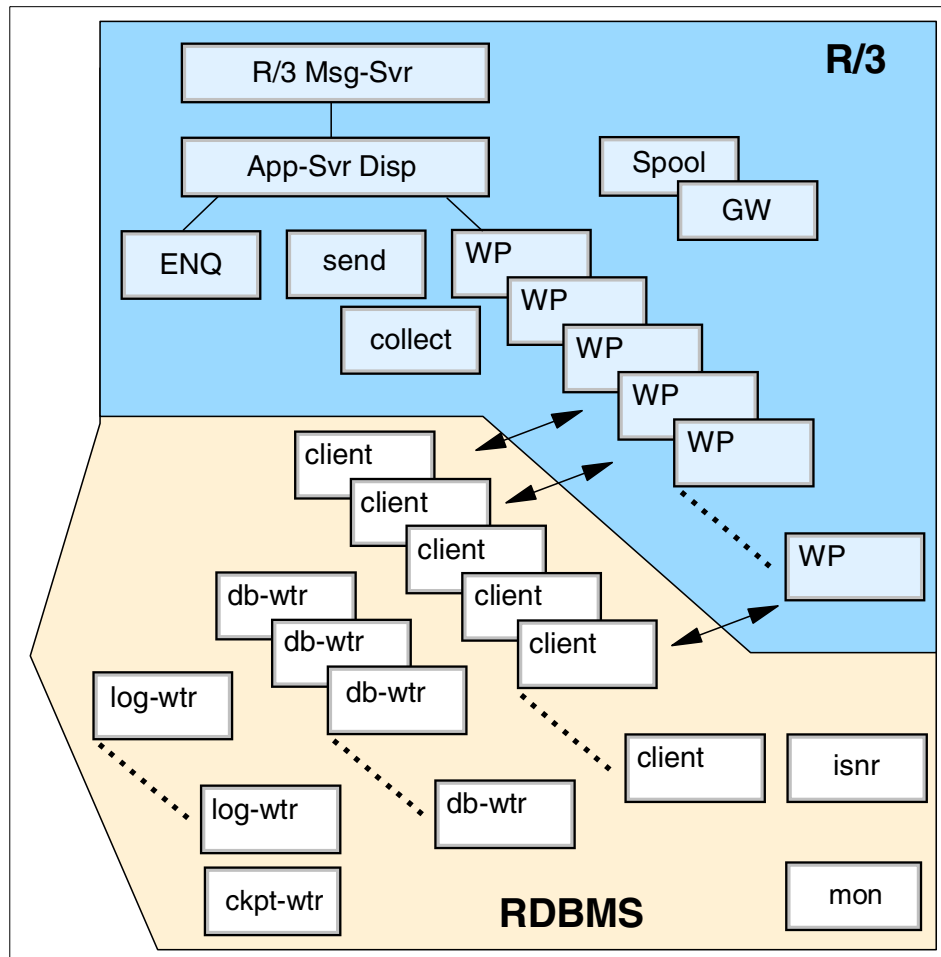


Figure 97. SAP R/3 central system

### 8.1.2.3 Multiple SAP R/3 systems consolidation objectives

In the series of tests which were run, the overall objective was to allow multiple SAP R/3 systems to run concurrently on a single server with predictable and controllable results, in a similar manner as if the SAP R/3 systems would separately run on a machine with the same CPU resources available as their given WLM entitlement. The expected results were defined to be response time and throughput for a specified CPU consumption. Given a predefined CPU resource allocation, a specific load level can be achieved by the SAP R/3 system with a consistent response time. If the workload increases, the response times will also increase (in a predictable manner). The objective is to make this true for multiple SAP R/3 systems running



concurrently on a single large machine. It should be possible to protect one SAP R/3 system from being penalized from the resource usage of a concurrent SAP R/3 system. In the case of contention, each SAP R/3 system must be confineable within its own allocation boundaries. A server consolidation environment has the additional benefit of allowing applications to have access to much more resources than is actually their due, if the resources are free.

WLM allows implementation of a priority chain, which will determine the behavior of resource allocation when various priority applications increase their requirements. In an SAP R/3 server consolidation environment, this would allow lower priority applications to take advantage of any remaining system resources without infringing on the priority application. It must be possible to define predictable response times and load throughput for the priority application, regardless of what the lower priority neighbors are doing.

In a server consolidation environment, it is not entirely possible to mask the presence of additional load on the machine caused by other applications. For multiple SAP R/3 systems, there will and must be some level of degradation due to the large number of processes being scheduled. However, the behavior of the individual applications must remain predictable in accordance with their allocation of machine resources.

On the plus side, when the competition is not using its full resource allocation, an SAP R/3 system can exceed its allotment and achieve much better response times as a result.

Without WLM, an increasing load on one SAP R/3 system will directly affect the behavior of all others as there is no way to limit one from consuming resources required by the others. SAP R/3 server consolidation without WLM cannot provide predictable results for any given SAP R/3 system when taking advantage of the total CPU resources.

#### **8.1.2.4 Multiple systems of equal size and equal priority**

This pattern is used to define the concept of logically dividing the machine into equal portions and allocating these portions to the individual SAP R/3 systems.

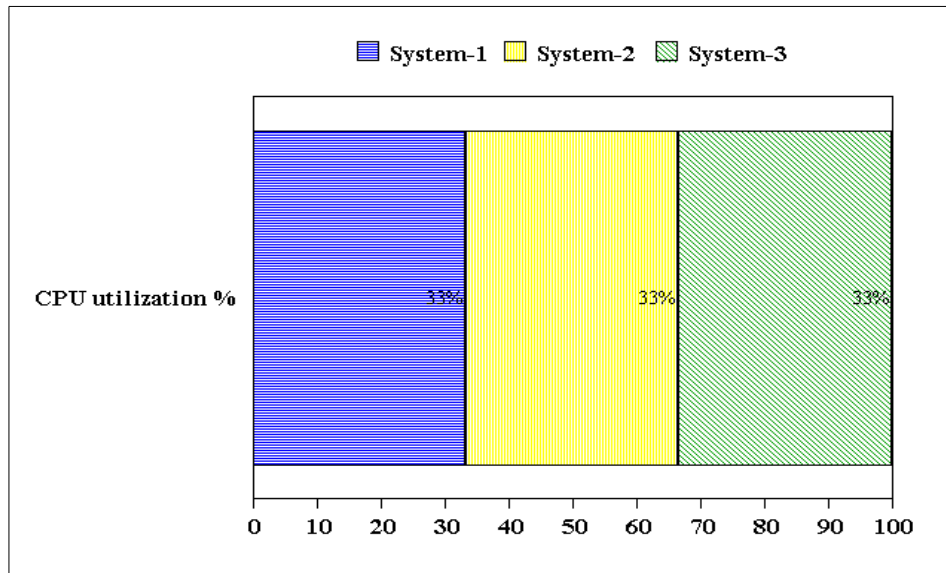


Figure 98. Three system consolidation of SAP R/3

For example, a three-system consolidation would give 33 percent of the system resources to each of the three SAP R/3 systems (Figure 98). This is the easiest to manage of all the scenarios as it is the easiest to define, and all of the processes regardless of class will have a similar scheduling behavior. In the test scenarios presented here, it was proven that the individual systems could be protected from starving. The processes of the system exceeding their allocations are the ones to pay the throughput and response time penalty, not all processes. This is a major benefit of WLM. In the tests done for this pattern, three separate SAP R/3 systems were run simultaneously, as depicted above, and were successfully contained within their entitlements.

In an attempt to test the scalability of this pattern beyond three SAP R/3 system (when only three systems were available), a multi-threaded test tool was used to simulate a fourth application. Although the scheduling of threads in this case differed from that of the SAP R/3 processes, its behavior was competitive enough to verify the pattern. In this case then, the machine resources were divided equally between the three SAP R/3 systems, and one multi-threaded non-SAP R/3 application of a similar competitive nature. For these tests the pattern could be repeated and load increased on any given system contained to that system. Due to the fact that all process classes have the same penalty behavior, it seems likely that this pattern could be extended

beyond the four applications tested. At four SAP R/3 systems, the pattern showed no indication of losing effectiveness (Table 19).

Table 19. WLM configuration for each system

# Systems	Min %	Max %	Shares	Tier
3	33	100	33	1
4	25	100	25	1

**Tips for configuring systems of equal size**

Dividing the system into equal shares is relatively easy. Table 19 on page 245 shows an example of the configuration used for the three SAP R/3 system tests. The division of the machine resources must consider the requirements of the largest system. If the systems are very different in size, or if one system must be allotted more than 50 percent of the CPU resources, then this pattern cannot work. In this case the unequal pattern is the only possibility. It is recommended not to use maximums to try to contain the systems, as they are not necessary for equal share distribution and can lead to worse response time performance overall. The best results were achieved by setting the maximum to 100 percent.

**8.1.2.5 Multiple systems of unequal priority**

Unequal distribution (see Figure 99 on page 245) is the far more difficult pattern to achieve for many reasons. In real life it will also be the most difficult to correctly maintain and monitor, but it is the most likely pattern to be expected as SAP R/3 systems will have a large range of capacity requirements.

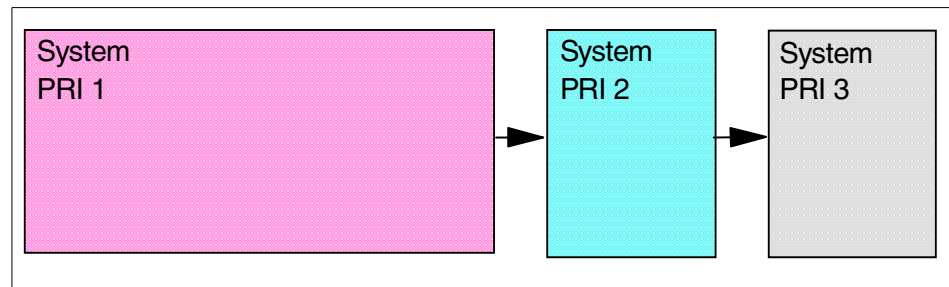


Figure 99. Resource usage hierarchy for unequal priority systems

It must be able to protect a large system from resource encroachment by multiple smaller systems as well as vice versa. It must also allow the definition of a hierarchy of priorities.

**Multiple systems of varying priority (priority chain)**

This pattern would be applicable when the SAP R/3 systems are being consolidated and the exact characteristics of the SAP R/3 systems' load are not known, but their relative importance is. From the scheduling point of view, such a pattern represents multiple classes of processes with different degradation behaviors. This pattern must allow for any system to be defined as the highest priority system, regardless of its size, and allow dynamic shifting of allocation boundaries according to activity and relative priority.

**Note**

The expression "size" in the paragraph above is equivalent to CPU utilization. Normally the CPU capacity requirements will be a direct result of the number of users. The number and type of users will have a direct influence on the amount of memory required for user contexts, table, program buffers, and so forth.

From the monitoring point of view, this pattern intensifies the uncertainty as to whether a system did not use its full allotment because it did not need it, or because it could not get it. In the case of equal priorities, the system can take the resources it needs up to its allotment. In the case of unequal priorities, lower priority systems can only take resources if they are not being used by higher priority systems.

In SAP R/3 server consolidation terms, this pattern allows us to define a hierarchy of systems in which the priority system maintains predictable response times and throughput with increasing load at the cost of those lower in the chain. This system has absolute priority.

In the tests done for this pattern, it was possible to construct an effective priority hierarchy of three SAP R/3 systems. As the distribution pattern implemented by the penalty priorities (see Table 20) generated in this test already showed some indications of stain at extreme over-commitment, it is doubtful that this pattern can be expanded beyond three SAP R/3 systems with the same quality.

The WLM Configuration for this pattern was as follows:

*Table 20. WLM configuration - penalty priorities*

<b>Sys priority</b>	<b>Tier</b>	<b>Min %</b>	<b>Max %</b>	<b>Shares</b>	<b>Penalty</b>
1	1	100	100	80	0
2	9	0	100	20	10

Sys priority	Tier	Min %	Max %	Shares	Penalty
3	9	0	1	1	66

Please refer to the tips for configuring unequal distribution for important information on number of instances and processes.

#### 8.1.2.6 Systems of unequal size but equal priority

From the logical view, this is similar to equal share distribution using unequally sized containers. The behavior should be that a SAP R/3 system can use all the resources within its allotment, degrade when it exceeds its allotment, and leave its neighbors undisturbed. This is an ideal server consolidation pattern for SAP R/3 systems of equal priority, but unequal size. A system used to consolidate multiple production SAP R/3 systems may find this pattern the most useful.

In this WLM configuration, the minimums and maximums are used with the objective of allowing the SAP R/3 system to run without constraints within the boundaries of its allocation, and try to degrade the processes as soon as it exceeds its allocation. The tests showed that WLM reacts very quickly to a “boundary dispute” under these conditions, immediately jumping the penalty to the maximum and just as quickly removing the penalty as soon as the application is back within its boundaries. This pattern showed very good resource control during high contention. This aim was achieved by setting the minimum to equal the maximum CPU limit, and both of these at the level of resource allocation.

Again, these limitations do not restrict a system from taking more resources if the resources are free. The priority penalty will be increased for the processes of the class that exceeds its entitlement, but they will still get the resources until another class requires them. A process class that has remained within its resource allotment will automatically have a higher priority, as it will have a zero penalty.

This pattern was tested for three systems; one large and two equal sized (see Table 21). The larger system was protected against both the smaller systems even when the load was increased on both of these systems simultaneously. Both smaller systems were in turn protected from the larger system and each other.

Table 21. WLM configuration unequal size but equal priority

System	Tiers	Min %	Max %	Shares
1	1	50	50	2
2	1	25	25	1
3	1	25	25	1

### 8.1.2.7 One priority system with several additional systems

One final pattern was tested that combines aspects of unequal priorities with equal resource distribution. In this case the aim was to define a pattern in which there was one primary system of absolute priority, and several systems (see Figure 100) equal amongst themselves but all of lower priority than the primary system. In this pattern, whatever resources not begin consumed by the primary system will be equally shared between the remaining systems (see Table 22).

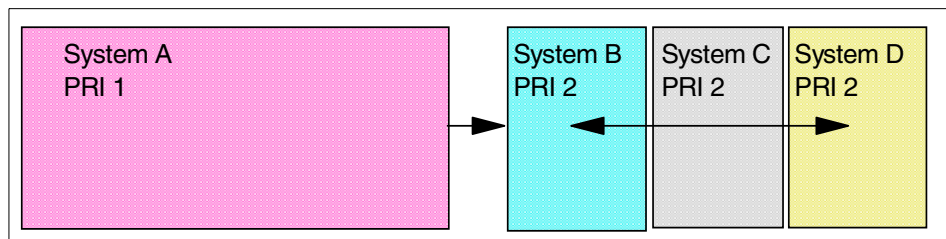


Figure 100. One priority system with several additional systems

Table 22. WLM configuration one priority system

System	Tier	Min %	Max %	Shares	Instances	Work procs.
A	1	100	100	100	3	60
B	9	33	100	33	1	20
C	9	33	100	33	1	20
D	9	33	100	33	1	20

### Tips for configuring systems of unequal size

These patterns are far more complex to configure. They require care in both the WLM and the SAP R/3 configurations. At the beginning of this document, the behavior of WLM's interaction with the AIX scheduler was discussed, and the idea of multiple AIX runqueues, a runqueue per CPU. These topics are important in obtaining effective WLM for unequal distribution.

In patterns used to create a priority chain, there are a number of high priority processes belonging to the preferred system, and possibly many lower priority processes belonging to all the subsequent systems. Due to process affinity (also mentioned earlier) for a specific CPU, the situation can easily arise where a runqueue is dominated by low priority processes. In this case, there is no way for this CPU to be preempted by a process of the high priority system.

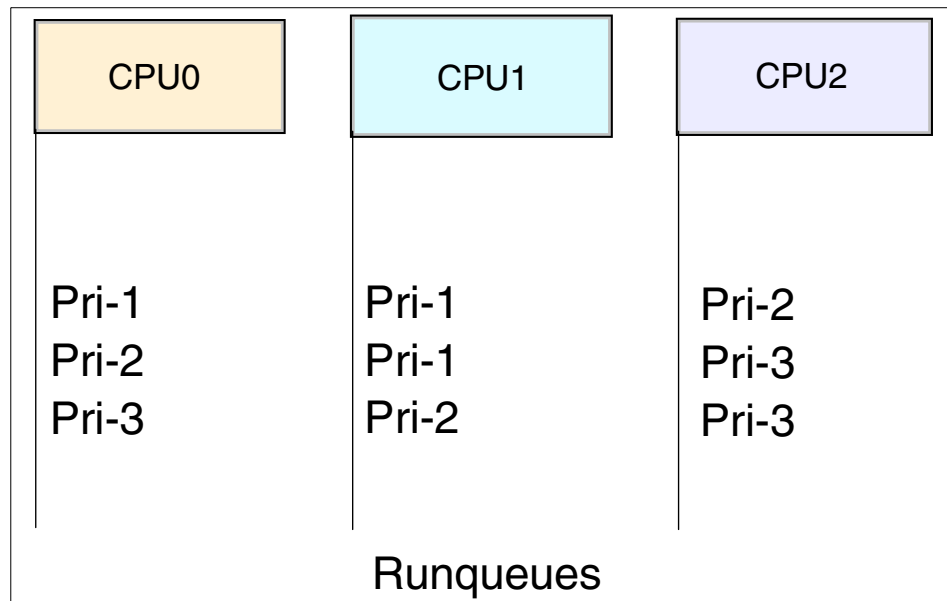


Figure 101. Runqueues

Figure 101 shows three separate runqueues for three CPUs. CPU-2 has only lower priority processes in its runqueue. Although AIX does some level of process redistribution, this affinity still makes itself noticeable under these conditions. In this example each system has an equal number of processes and is still keeping the ratio of lower priority to high priority processes at 2:1.

#### 8.1.2.8 Process distribution recommendation

For a priority chain, the higher priority SAP R/3 system should have a number of dispatch and work processes at least equal to the total of all the lower

priority systems to successfully compete. In the tests, the number of processes for the priority chain were defined as in Table 23.

Table 23. Priority chain

Priority	Instances	Work processes
1	3	60
2	2	40
3	1	20

In the pattern for supporting systems of unequal size but equal priority, each SAP R/3 system should have a percentage of work processes representing its share of system resources.

In the test scenarios, we used the following work processes ratio (Table 24).

Table 24. Work processes

Ratio %	Instances	Work processes
50	2	40
25	1	20
25	1	20

A further consideration about work processes pertains to a temporary limitation in the SAP R/3 kernel at the time of the tests. During the whole of the test series, the SAP R/3 kernel was not able to handle more than 20 dispatch and work processes per application server instance. Due to this restriction, multiple instances were used when the tests needed to exceed this number for a given SAP R/3 system. An SAP R/3 instance can normally support up to 99 processes and, therefore, under normal circumstances a single instance would have been used and the number of processes increased. The fact that multiple SAP R/3 instances were used in this test scenario may have had other positive effects on the test results is documented here. This approach may be beneficial in real environments.



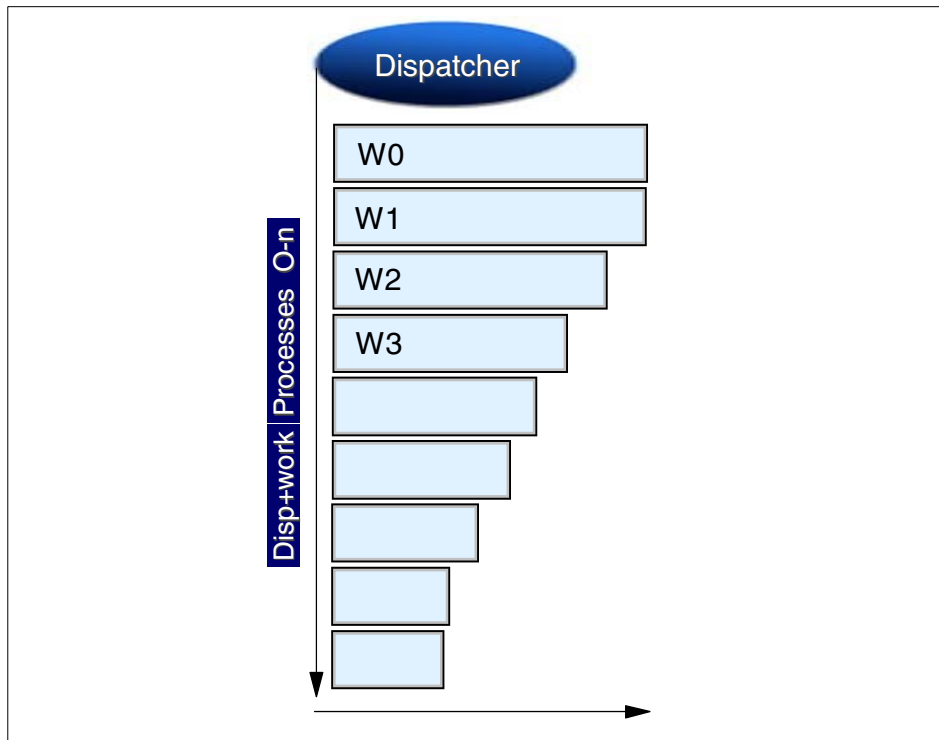


Figure 102. CPU utilization per process

Figure 102 depicts the typical workload behavior of the dispatcher and work processes in a SAP R/3 system. Work is distributed by the dispatcher to the first work process in the chain that is not busy. Normally the processes near the top of the chain do the majority of the work. Please note that the positions in this chain are static.

During the tests, there were up to three SAP R/3 system instances active in order to achieve the optimal balance of work process processes according to SAP R/3 system priority shown in Figure 103 on page 252. This would have enhanced load distribution across a larger number of work processes even in low load tests, as well as providing multiple workload dispatcher processes per SAP R/3 system.

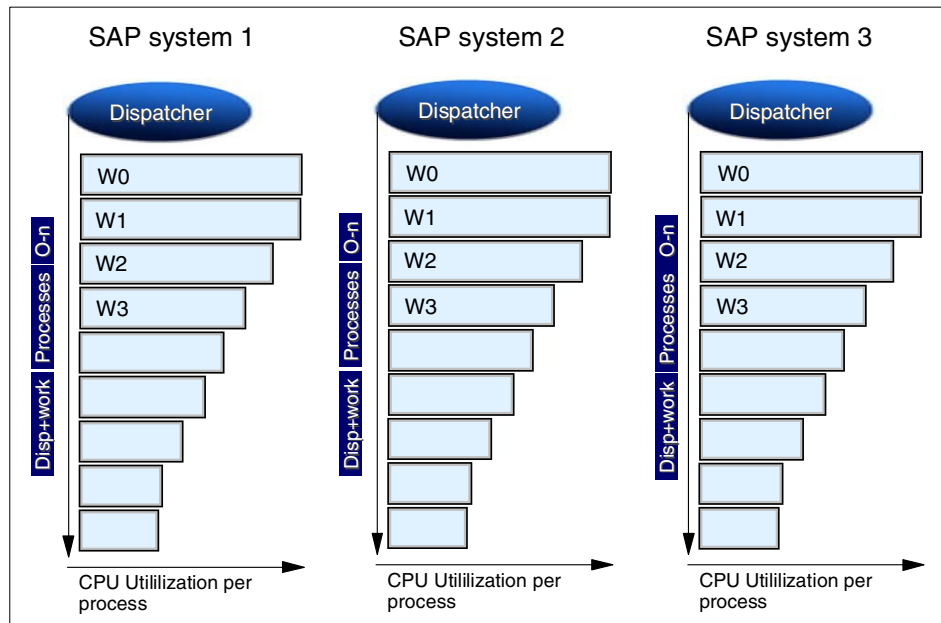


Figure 103. CPU utilization per process

## 8.2 Customer experience - WLM and a compute server for research

The following section describes how WLM is used on a central AIX server in a research environment where interactive and batch work is done on the same machine.

When WLM was presented by IBM at the SHARE conference (IBM user organization) in Anaheim, CA in March 2000, it was obvious that WLM was the long awaited tool to overcome some problems in managing AIX, especially distributing resources according to installation-specified policies. In April, WLM was installed and ran successfully in passive mode. In May 2000, it was decided to use WLM on the production system in active mode.

### 8.2.1 The installation

The Forschungszentrum Jülich GmbH (Research Center Jülich), one of 16 Helmholtz research centers in Germany, links all its work to the common denominator, "The future is our mission." A staff of 4300 are devoted to investigating current issues in the areas of energy, environment, life, information, and matter in one of the largest research institutes in Europe. In Jülich, scientists from many different disciplines including physics, chemistry,

biology, medicine, and engineering work closely together. This work results in contributions to basic research and long-term programs, applied research, and key technologies. For more information about the Jülich Research Center, visit the following Web site:

<http://www.fz-juelich.de>

The Central Institute for Applied Mathematics (ZAM) within Forschungszentrum Jülich is responsible for the planning, installation, and operation of the supercomputers and central server systems, and of the campus-wide computer networks and communication systems. The services comprise all functions of a computer center including user support.

As part of the John von Neumann Institute for Computing (NIC), ZAM provides supercomputer resources for the scientific community in Germany. For more information about the Central Institute for Applied Mathematics, visit the following Web site:

<http://www.fz-juelich.de/zam>

ZAM runs one of the most powerful scientific computer centers in Europe with six supercomputers, an IBM server, and a series of systems for special purposes, such as visualization and communications.

For a detailed configuration, see the following Web site:

<http://www.fz-juelich.de/zam/CompServ/services/config.html>

### **8.2.2 Central AIX system**

The central computing system offers a wide spectrum of application software. It is used interactively and offers batch services for long running jobs. The hardware and software configuration of the system is as follows:

- RS/6000 44P-270, 4 Way, 8GB RAM
- Operating System AIX 4.3.3-03
- Batch-System LoadLeveler V1.3
- Overall peak performance 4.8 Gigaflops
- Concurrent users (peak) approximately 150
- Joined users approximately 1650
- Disk capacity for user data 360 GB

This system allows users without local computing resources to access Unix applications via X-terminals or PCs with an appropriate X emulation. It is an

application server for software. It is available as a computing resource for scalar, interactive, and batch work. In particular, applications with demands for large virtual memory run extremely well on this machine.

### 8.2.3 Problems

When the same server is used for interactive and batch work, the distribution of resources between these two different workloads is a difficult task. On one side, interactive work should experience the optimum performance to give scientists the best response time for their current work. On the other side, batch jobs using several hours of CPU time should have reasonable turn-around times.

Batch jobs in this environment are typically CPU-bound.

When we tried to maximize system utilization by allowing as many batch jobs to run as there were processors, interactive users complained about excessive response times.

When the number of simultaneous batch jobs was reduced, batch users complained about idle system time and long queues for their batch jobs.

Another problem showed up during the production period; interactive X-terminal users often started Netscape processes on the central machine because they had no other workstation or PC to browse the Internet. Depending on the Web site visited, these Netscape processes sometimes went into a tight CPU loop without the user getting anything useful. What is worse, these tight CPU loops were not automatically ended through the *cpu\_hard* parameter in */etc/security/limits*.

### 8.2.4 A pre-WLM solution

To overcome the problems in AIX releases without WLM, the following rules were adopted and put in place:

- Half of the CPUs are reserved for interactive work only at prime times (workdays from 8:00 a.m. to 6:00 p.m.).
- At least one CPU is reserved for interactive work all the time.
- Interactive work is limited to 30 CPU minutes per process.
- Batch jobs (submitted through LoadLeveler) can use up to 10 hours of CPU time.
- Batch jobs (submitted through LoadLeveler) run at a lower priority (higher nice values).

- Netscape processes are killed without warning if they have used 30 minutes of CPU time.

### 8.2.5 The WLM solution with AIX Version 4.3.3-02

The WLM files listed in Table 25 were defined for peak times (Monday through Friday, 8:00 a.m. to 6:00 p.m.).

Table 25. WLM configuration for peak time

Tier	Class	User	Application	CPU	Memory
9	slow		/usr/local/ netscape/ netscape_aix4	min=0 max=10 share=1	min=0 max=10 share=1
2	batch	batuser1		min=0 max=100 share=100	min=0 max=100 share=100
2	batch	batuser2		min=0 max=100 share=100	min=0 max=100 share=100
2	batch	batuser3		min=0 max=100 share=100	min=0 max=100 share=100
2	batch	batuser.		min=0 max=100 share=100	min=0 max=100 share=100
2	batch	batuser99		min=0 max=100 share=100	min=0 max=100 share=100
0	System	root		min=10 max=100 share=200	min=13 max=100 share=200
0	System	loadl		min=10 max=100 share=200	min=13 max=100 share=200
0	System	admusr		min=10 max=100 share=200	min=13 max=100 share=200
0	System	dispatch		min=10 max=100 share=200	min=13 max=100 share=200

Tier	Class	User	Application	CPU	Memory
1	Default			min=20 max=100 share=100	min=20 max=100 share=100

Two adjustments were made for offpeak time:

- tier value batch class = tier value Default class
- shares batch class = 1/2 shares Default class

Table 26. WLM configuration for offpeak time

Tier	Class	User	Application	CPU	Memory
9	slow		/usr/local/ netscape/ netscape_aix4	min=0 max=10 share=1	min=0 max=10 share=1
1	batch	batuser1		min=0 max=100 share=50	min=0 max=100 share=50
1	batch	batuser2		min=0 max=100 share=50	min=0 max=100 share=50
1	batch	batuser3		min=0 max=100 share=50	min=0 max=100 share=50
1	batch	batuser.		min=0 max=100 share=50	min=0 max=100 share=50
1	batch	batuser99		min=0 max=100 share=50	min=0 max=100 share=50
0	System	root		min=10 max=100 share=200	min=13 max=100 share=200
0	System	loadl		min=10 max=100 share=200	min=13 max=100 share=200
0	System	admusr		min=10 max=100 share=200	min=13 max=100 share=200

Tier	Class	User	Application	CPU	Memory
0	System	dispatch		min=10 max=100 share=200	min=13 max=100 share=200
1	Default			min=20 max=100 share=100	min=20 max=100 share=100

With these definitions, WLM was started in passive mode, and the `wlmstat` output was analyzed. After some minor adjustments, WLM was run in active mode:

Peak time (Monday till Friday, 8 am to 6 pm): `wlmctrl -d peak`

Other: `wlmctrl -d offpeak`

The 30 minutes CPU time limit for interactive processes was still in effect in `/etc/security/limits`.

#### 8.2.5.1 Major advantages of this solution

More batch jobs could be started without disturbing interactive users because, in peak times, the batch jobs with their lower tier value could absorb the CPU cycles of the machine that would go idle otherwise. In this way, a higher batch load could take advantage of the overlaps of I/O and CPU demands.

The priority of Netscape processes can now never be higher than any other processes.

#### 8.2.5.2 Disadvantage of this solution

Sometimes, batch users wanted to do some interactive work at the same time. Because their user ID was defined in the rules file belonging to the batch class they had to use different user IDs to prevent the system from running their interactive work with the batch tier.

This is, of course, not practical and creates an administrative nightmare. So, the inheritance feature of WLM allowing the class inheritance of processes started by LoadLeveler was really needed badly.

### 8.2.6 The second WLM solution with AIX 5L

Among many additional features, the new functions of WLM released in AIX 5L allow the class inheritance of processes started by a batch system (that is

LoadLeveler). With this enhancement, the definition of the WLM files is now very easy. Table 27 shows the WLM configuration with AIX 5L.

Table 27. WLM configuration with AIX 5L

Tier	Class	Inheritance	User	Application	CPU	Memory
9	slow	no		/usr/local/ netscape/ netscape_aix4	min=0 max=10 share=1	min=0 max=10 share=1
2	batch	yes		~loadl/bin/ LoadL_starter	min=0 max=100 share=100	min=0 max=100 share=100
0	System	no	root		min=10 max=100 share=200	min=13 max=100 share=200
0	System	no	loadl		min=10 max=100 share=200	min=13 max=100 share=200
0	System	no	admusr		min=10 max=100 share=200	min=13 max=100 share=200
0	System	no	dispatch		min=10 max=100 share=200	min=13 max=100 share=200
1	Default	no			min=20 max=100 share=100	min=20 max=100 share=100

These changes allow batch processes to inherit the class characteristics of the LoadLeveler starter process. They combine the advantages of the previous WLM release and get rid of its disadvantages.

### 8.2.7 Conclusion

WLM allows an installation to administer the system in a much more flexible way compared to previous AIX releases. There is no additional effort to install WLM because it is included in the AIX kernel. Configuring WLM is very easy:

1. Start up with a simple classification model.
2. Run WLM in passive mode.
3. Collect statistics with `wlmstat` to determine how the shares and limits should be set.



4. Set shares and limits.
5. Run WLM in active mode.
6. Collect statistics with `wlmstat` to see if the defined goals are achieved.
7. Modify shares, limits, rules, and tiers.
8. Repeat the last two steps a few times.
9. Perform step 1 through 8 with various alterations and save them into different WLM configurations.
10. Decide which is your best WLM configuration.

This process could be done over a few days. It is a powerful tool to allow resources to be distributed to users in an installation-defined policy. For the first time, service level agreements can be negotiated and enforced in a production environment.



---

## Appendix A. AIX Workload Manager API routines

The WLM API routines are described in this appendix from a technical viewpoint for practical utilization purposes.

---

### A.1 The Include file - sys/wlm.h

#### **Purpose**

Defines the constants, data structures, and function prototypes used by the Workload Manager Application Programming Interface (API) routines.

#### **Description**

The wlm.h file defines the *wlm\_args*, *wlm\_assign*, *wlm\_info*, *wlm\_bio\_class\_info\_t*, and *wlm\_bio\_dev\_info\_t* structures. These structures are used by the WLM API functions in the libwlm.a library.

#### **Data structures**

The *wlm\_args* structure is used to pass class information to WLM when using the API functions to create, modify, or delete a class.

#### A.1.1 wlm\_args

The *wlm\_args* structure has the following fields:

Field	Description
versflags	The four high order bits contain a version number used by the API to maintain binary compatibility in the event of future modifications of the data structures. The rest of the integer will be used to pass flags to the API functions when needed. This field should be initialized with a logical OR between the version number, WLM_VERSION, and whatever flags are needed by the target function. One flag common to all the API call is WLM_MUTE, which is used to suppress the output of error messages from the WLM library on stderr.
confdir	Null-terminated string. This field must be initialized with the name of the WLM configuration the target API function applies to (when applicable - see individual API routines). Alternatively, this field can be set to a null string (\0) to indicate that the class addition/modification is to be applied only to the WLM kernel data and not to the class description files.

Field	Description
class	This field is a structure of type struct, <i>class_definition</i> , which contains all the information pertaining to the superclass or subclass needed by the target API function. The fields in this structure can be initialized by a call to <i>wlm_init_class_definition</i> so that programmers will only have to initialize the fields they wish to modify.

The main structure in *class\_definition* is the class description, struct *class\_descr* with the following fields:

Field	Description
res	<p>An array of type struct <i>wlm_bounds</i> containing for each resource type:</p> <p>min: Minimum limit: value between 0 (default) and 100.</p> <p>shares: Shares number: value between 1 and 65535. The value -1 (default) indicates that the given resource is not managed by WLM for this class.</p> <p>softmax: Soft maximum limit: value between 0 and 100 (default). Must be greater than or equal to min.</p> <p>hardmax: Hard maximum limit: value between 0 and 100 (default). Must be greater than or equal to min and softmax.</p> <p>The resource types are defined as WLM_RES_CPU, WLM_RES_MEM, and WLM_RES_BIO. Each value represents the index in the array of the element corresponding to the type of resource.</p>
tier	Tier number for the class: value between 0 (default) and 9.
inheritance	Flag to indicate whether a new process should be automatically classified on <i>exec</i> using the assignment rules (value 0, which is the default), or inherit the class from its parent process (value 1).
localshm	This attribute indicates whether memory segments in this class remain local to the class (value 1) or if they go to the Shared class (value 0, the default), when accessed by a process belonging to another class.

Field	Description
assign_uid	User ID of the user allowed to manually assign processes to this class. When specified, it must be a valid user ID. The default when this attribute is not specified is that no user is authorized (WLM_NOGUID).
assign_gid	Group ID of the group of users allowed to manually assign processes to this class. When specified, it must be a valid group ID. The default when this attribute is not specified is that no group is authorized (WLM_NOGUID).  If both assign_uid and assign_gid are left to their default value (WLM_NOGUID), only root can assign processes to the class.
admin_uid	The user ID of the user allowed to administrate the subclasses of the superclass (superclass only).
admin_gid	Group ID of the users allowed to administrate the subclasses of the superclass (superclass only).  If both admin_uid and admin_gid are left to their default value (-1), only root can administrate the subclasses of this superclass.
name	The null-terminated full name of the class in the form <i>supername</i> for a superclass, and <i>supername.subname</i> for a subclass. The superclass and subclass names are both limited to 16 characters. There is no default value for this field.

In addition to the class description, *class\_definition* adds two fields:

Field	Description
rset_name	A null-terminated character string containing the name of the resource set (partition) the class is restricted to (when applicable). The default is that the class can access all the resources on the system.
descr_field	A null-terminated character string containing the description text of the class. This is an optional field; there is no default.

### A.1.2 wlm\_assign

The *wlm\_assign* structure is used to manually assign processes or groups of processes to a specified superclass or subclass using the `wlm_assign` routine. The *wlm\_assign* structure has the following fields:

Field	Description
<code>wa_versflags</code>	The four high order bits contain a version number used by the API to maintain binary compatibility in the event of future modifications of the data structures. The rest of the integer will be used to pass flags to the API functions when needed. This field should be initialized with the version number, <code>WLM_VERSION</code> . The flag, <code>WLM_MUTE</code> , can be used to suppress the output of error messages from the WLM library on <code>stderr</code> .
<code>wa_pids</code>	The address of an array containing the process identifiers (pid's) of the processes to be manually assigned.
<code>wa_pid_count</code>	The number of pid's in the array above.
<code>wa_pgids</code>	The address of an array containing the process group IDs (pgid's) of the process groups to be manually assigned.
<code>wa_pgid_count</code>	The number of pgid's in the array above.
<code>wa_classname</code>	The full name of the superclass ( <i>supername</i> ) or the subclass ( <i>supername.subname</i> ) of the class to which you want to manually assign processes.

### A.1.3 wlm\_info

The *wlm\_info* structure is used to extract information about the current configuration parameters and current resource utilization of the active classes using the function `wlm_get_info`.

The *wlm\_info* structure has the following fields:

Fields	Description
<code>i_descr</code>	The class description of type struct, <i>class_descr</i> , described above.

Fields	Description
<code>i_regul</code>	<p>A per-resource type array of structures of type struct <i>wlm_regul</i> containing the following fields:</p> <p><i>consum</i>: The resource consumption of the class expressed as a percentage of the total resource available.</p> <p><i>total</i>: This 64 bit number represents the total amount of the resource consumed by the class since its creation (or since WLM started). The unit is CPU ticks for CPU, a number of pages * seconds for memory and the total number of 512 byte blocks for disk I/O.</p> <p>The indexes into the array of the various resources are defined as above by WLM_RES_CPU, WLM_RES_MEM, and WLM_RES_BIO.</p>
<code>i_class_id</code>	Class identifier (index of internal kernel class related to classes, <i>class_control_block (ccb[])</i> table).
<code>i_cl_pri</code>	Priority delta applied to the threads in the class (CPU regulation).
<code>i_cl_inuse</code>	The current number of processes in the class.
<code>i_cl_npages</code>	The number of memory pages currently allocated to the class.
<code>i_cl_mem_hwm</code>	The maximum number of (resident) memory pages this class has had since its creation (memory high water mark).
<code>i_cl_change_level</code>	Incremented every time there is a change in the current WLM configuration. For use by the WLM monitoring tools.

There are two structures used to get the I/O statistics using `wlm_get_bio_stats` depending on whether the application wants per-class or per-device statistics.

#### A.1.4 wlm\_bio\_class\_info\_t

The *wlm\_bio\_class\_info\_t* structure is used to gather I/O statistics per class and per device. This structure contains the following fields:

Field	Description
wbc_dev	Device identifier (dev_t).
wbc_cid	Class identifier (index of the internal kernel class related to classes <i>class_control_block</i> (ccb[] table). The connection between the class ID and the class name can be done using <i>wlm_get_info</i> , which returns both the class name (in field <i>i_descr</i> ) and the class ID (in <i>i_class_id</i> ) in the <i>wlm_info</i> structure.
wbc_regul	A structure of type struct, <i>wlm_regul</i> , already described, containing the disk I/O statistics for the given class and device: Resource utilization expressed as a percentage of the total available throughput of the device (consum) and the total number of 512 byte blocks read/written from and to the device by processes in the class since the creation of the class or since WLM started (whichever happened last).
wbc_delay	Delay (in milliseconds) imposed to the I/Os of the processes in the class to the device in order to limit the utilization of this device by the processes in this class when it is consuming more than its entitlement.

#### A.1.5 wlm\_bio\_dev\_info\_t

The *wlm\_bio\_dev\_info\_t* structure is used to gather the global statistics for a given device. It takes into account all I/Os to and from the device by all the classes accessing the device. This structure contains the following fields:

Field	Description
wbd_dev	Device identifier (dev_t).
wbd_active_cntrl	Number of classes actively accessing the device.
wbd_in_queue	Number of requests in the device queue.



Field	Description
wbd_last	<p>Device statistics for the last second. This field is an array of integer values. Symbolic values defined in the header file describe each index in the array:</p> <p><i>WBS_OUT_RTHRPUT</i>: Number of blocks actually read from the device (I/O completed).</p> <p><i>WBS_OUT_WTHRPUT</i>: Number of blocks actually written to the device (I/O completed).</p> <p><i>WBS_IN_RTHRPUT</i>: Requested number of blocks to read from the device.</p> <p><i>WBS_IN_WTHRPUT</i>: Requested number of blocks to write to the device.</p> <p><i>WBS_REQUESTS</i>: Number of requests (read/write).</p> <p><i>WBS_QUEUED</i>: Number of requests queued.</p> <p><i>WBS_STARVED</i>: Number of requests starved (not serviced during the time interval).</p> <p>For the <code>wbd_last</code> field, these numbers represent activity during the last second (for instance, the number of requests queued during the last second).</p>
wbd_max	<p>This field contains the maximum values observed since the device was first used (after WLM started) for all the entries of the array described above (for instance, the maximum number of blocks actually read from the device in one second since the device was first accessed).</p>
wbd_av	<p>This field contains the average values for all the entries in the array (for instance, the average number of requests in the device queue).</p>
wbd_total	<p>This field is an array of 64 bit integers parallel to the arrays above that contains, for all the entries, the total of all the values measured every second since the device was first accessed (for instance the total number of blocks written to the device since the device was first accessed).</p>

---

## A.2 WLM API functions error codes

The various API functions may return one or several of the following error codes:

WLM_BADVERS	Bad Version number passed in versflags.
WLM_NOTINITED	No prior call to <code>wlm_initialize</code> .
WLM_ALREADYINIT	There already has been a prior call to <code>wlm_initialize</code> .
WLM_UNSUPP	Operation or flags value not supported.
WLM_OPENERR	A file could not be opened.
WLM_CREATERR	A file could not be created.
WLM_MKDIRERR	A directory could not be created.
WLM_WRITERR	An attempt to write in a file did not succeed.
WLM_REMERR	An attempt to remove a file did not succeed.
WLM_RENAMERR	An attempt to rename a file did not succeed.
WLM_SYMLERR	An attempt to create a symbolic link did not succeed.
WLM_NOMEM	Not enough memory.
WLM_NOCLASS	The specified class does not exist.
WLM_RNOCLASS	A class specified in the rules file does not exist.
WLM_EXISTS	The specified class already exists.
WLM_MAXCLASSES	The maximum number of classes has been reached.
WLM_RMPREDEF	Predefined classes, such as Default and System, cannot be removed.
WLM_NOSUBS	The target superclass has no subclasses.
WLM_HASSUBS	The target superclass has subclasses.
WLM_SHAREDSUB	Shared superclass cannot have subclasses.
WLM_SHAREDLIM	Shared class can have shares and limits set only for memory.
WLM_BADDEFshr	Default shares value specified in the shares file is invalid.
WLM_BADDEFlim	Default limits value specified in the limits file is invalid.
WLM_BADLIMfmt	Value specified for minimum or maximum resource limit invalid.
WLM_BADSHRFMT	Value specified for resource shares is invalid.
WLM_BADTIER	Tier values must be between 0 and 9.
WLM_BADSHARES	Shares values must be between 1 and 65535.
WLM_BADMIN	Minimum resource limits values must be between 0 and 100.
WLM_BADSMAX	The soft maximum limit values must be between 1 and 100.
WLM_BADHMAX	The hard maximum limit values must be between 1

	and 100.
WLM_BADCNAME	Class names must be alphanumeric.
WLM_TOOLONG	The specified class name is too long.
WLM_MINSMAX	The minimum limit cannot be greater than the soft maximum limit.
WLM_SMAXHMAX	The soft maximum limit cannot be greater than the hard maximum limit.
WLM_SUMMINS	The sum of the minimum limits for a given resource and a given tier cannot exceed 100 percent.
WLM_BADINHER	The value specified for the class inheritance attribute is invalid.
WLM_LOADERR	A class cannot be loaded into the kernel.
WLM_RULESERR	The assignment rules table cannot be loaded into the kernel.
WLM_SETERR	The WLM state transition requested is illegal.
WLM_QUERYERR	Cannot query wlm state.
WLM_MANYRULES	Too many assignment rules.
WLM_MANYITEMS	Too many items in an assignment rule.
WLM_RULERR	An assignment rule has an invalid format.
WLM_BADLIST	The process attribute list of an assignment rules is invalid.
WLM_BADUSR	The specified user ID is not valid on the system.
WLM_BADRUSR	A user name specified in the rules file is invalid on the system.
WLM_BADUID	The specified user ID is not valid on the system.
WLM_BADGRP	The specified group ID is not valid on the system.
WLM_BADRGRP	A group name specified in the rules file is invalid on the system.
WLM_BADGID	The specified group ID is not valid on the system.
WLM_BADTAG	An invalid tag is specified in a rule.
WLM_BADTYP	An invalid type is specified in a rule.
WLM_NOSHRRULE	Cannot specify the rule for a Shared class.
WLM_NOWILDCRD	Wildcards are not allowed in this field.
WLM_STATERR	One (or more) file names specified in the application field of an assignment rule could not be accessed. The corresponding names are ignored (warning).
WLM_EMPTYRULE	None of the file names specified in the application field of an assignment rule could be accessed. The rule is ignored (warning).
WLM_RUNERR	The WLM library was not able to execute a command needed for the specific function. This is not an application error but, most likely, a system

	administration problem. The commands used by the library are basic AIX commands such as <code>lsuser</code> , <code>lsgroup</code> , <code>echo</code> , and <code>grep</code> .
WLM_BADCONFIG	Invalid configuration name.
WLM_CLASSMIS	No class definition found.
WLM_EMPTYATTR	No valid attributes found in attributes string for <code>wlm_classify</code> .
WLM_MULTATTR	Multiple specifications not allowed in attributes string for <code>wlm_classify</code> .
WLM_EXCLATTR	Exclusions not allowed in attributes string for <code>wlm_classify</code> .
WLM_ATTERR	Attribute format error in attributes string for <code>wlm_classify</code> .
WLM_BADATTUSR	Unknown user in attributes string for <code>wlm_classify</code> .
WLM_BADATTGRP	Unknown group in attributes string for <code>wlm_classify</code> .
WLM_BADATTAPP	Application file in attributes string for <code>wlm_classify</code> could not be accessed.
WLM_BADATTTAG	Invalid tag in attributes string for <code>wlm_classify</code> .
WLM_BADATTTYP	Invalid type in attributes string for <code>wlm_classify</code> .
WLM_TOOMANYATT	Too many items in attributes string for <code>wlm_classify</code> .
WLM_WILDCRDATT	Wildcards not allowed in attribute field.
WLM_RUNERRATT	Cannot expand attribute.
WLM_BADLISATT	Invalid list in attributes string for <code>wlm_classify</code> .
WLM_TOOLONGATT	Attribute list for <code>wlm_classify</code> too long.
WLM_EFAULT	Bad parameter address.
WLM_NOTCOMPLETE	Warning: could not assign all processes ( <code>wlm_assign</code> was partially successful).
WLM_NOTRUNNING	WLM is not running.
WLM_ESRCH	No such processes.
WLM_TOOMANYPID	Process ID list too long.
WLM_EPERM	Permission denied.
WLM_CANTASSIGN	Internal error: Could not make assignment.
WLM_TAGTOOLONG	Tag is too long.
WLM_BADFLAGS	Invalid flags value.
WLM_CANTSETTAG	Internal error: Could not set tag.
WLM_CANTCHECK	Unable to check the configuration.
WLM_TOOSMALL	Output buffer too small.
WLM_BADRSET	Bad Rset attribute for a class.
WLM_CHOWNERR	Cannot change file owner.
WLM_LOCKERR	Cannot take file lock.
WLM_ERRNO	A system call returned an error.
WLM_BADCLNAME	Class name invalid: Some class names cannot be used for internal reasons. For instance, Default.

WLM_BADSUPER	Bad superclass for subclass assignment.
WLM_NOTASSGND	Process has not been manually assigned to a class.
WLM_RULTOOLNG	Rule exceeds 4096 characters in length (WLM_RULE_LEN).
WLM_NOADMINSUB	adminuser/admingroup attributes not applicable to subclasses.

**Note**

The wlm.h header file provides a complete list of error codes.

---

## A.3 Initialization routines

There are two initialization routines in the API; `wlm_init_class_definition` and `wlm_initialize`.

### A.3.1 `wlm_init_class_definition`

**Purpose:** Initializes a variable of type struct *class\_definition*, defined in `<sys/wlm.h>` for use as an argument to WLM API function calls.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_init_class_definition (wlmargs)
struct wlm_args *wlmargs;
```

**Description:** The `wlm_init_class_definition` routine initializes (or reinitializes) the data structure of the type struct, *class\_definition*, which is part of the argument of type struct *wlm\_args* pointed to by `wlmargs` (field `class`) so that this data structure can be used as an argument for the class management routines of the WLM API library. The purpose of this call is to allow applications to initialize only the fields that are relevant for the operation they execute. For example, to change a CPU limit or share for an existing class, after a call to `wlm_init_class_definition`, the application will just have to initialize the fields corresponding to the values it wishes to modify. This routine initializes all values to specific invalid values so that the WLM library routines can find out which fields have been explicitly initialized by the user. This way, they can set or modify only the corresponding attributes.

When creating a class, for instance, it is different to leave a class attribute at its invalid value set by `wlm_initialize` than to set its value to the current default value for the attribute. In the former case, the attribute will not appear in the property file. In the latter, it will appear and be set with the value

passed. This makes a difference if a WLM administrator decides to change the default value for an attribute using the special stanza, *default*, in a property file. For instance, the system default for the inheritance attribute is *no*. If, at some point in time, a WLM administrator wants the inheritance to be *yes* by default, using this special stanza, all the classes in the classes property file, for which the inheritance attribute has not been specified will now use the default of *yes*. Those for which the inheritance attribute has been specified with its old default of *no* will not have inheritance.

**Parameter:**

`wlmargs`            The address of the struct *wlm\_args* data structure containing the *class\_definition* structure to be initialized. Only the `versflags` field of the *wlm\_args* structure passed needs to be initialized with `WLM_VERSION`.

**Return Values:** Upon successful completion, a value of 0 is returned. If the `wlm_init_class_definition` routine is unsuccessful, a non 0 value is returned.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, “WLM API functions error codes” on page 268.

### A.3.2 `wlm_initialize`

**Purpose:** Prepares WLM for use by an application.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_initialize (flags)
int flags;
```

**Description:** The `wlm_initialize` routine initializes the WLM API for use with an application program. It is mandatory to call `wlm_initialize` prior to using the WLM API. Otherwise, all other WLM API function calls will return an error. If `wlm_initialize` is used in a multi-threaded application, the routine should be called by the main thread before additional threads are started.

**Parameter:**

`flags`            The format is the same as the `versflags` field of the *wlm\_args* structure. The value for the argument must have the version number in the upper 4 bits (`WLM_VERSION`) possibly ORed with a flag in the lower 28 bits.

**Return Values:** Upon successful completion, a value of 0 is returned. If the `wlm_initialize` routine is unsuccessful, a non 0 value is returned.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, “WLM API functions error codes” on page 268.

---

## A.4 Application tag

The routine described in this section is the one used to tag a process;  
`wlm_set_tag`.

### A.4.1 `wlm_set_tag`

**Purpose:** Sets the current process' tag and related flags

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
#include <sys/user.h>
int wlm_set_tag (tag, flags)
char *tag;
int *flags;
```

**Description:** The tag is a new attribute of a process that can be set using the WLM function `wlm_set_tag`. This tag is a character string with a maximum length of `WLM_TAG_LENGTH` (not including the null terminator). Process tags can be displayed using the `ps` command. The tag is also one of the process attributes used in the assignment rules to automatically assign a process to a given class. The main utilization of the tag attribute is to allow WLM administrators to discriminate between several instances of the same application, which, typically, have the same user and group IDs, execute the same binary, and, therefore, would end up in the same class using the standard classification criteria. When an application sets its tag using `wlm_set_tag`, it is automatically reclassified according to the current assignment rules, and the new tag is taken into account when doing this reclassification. In addition to the tag itself, the application can also specify flags indicating to WLM whether a child process should inherit the tag from its parent after a `fork` and/or an `exec` system call. A process does not require any special privileges to set its tag.

**Parameters:**

<code>tag</code>	The address of a character string. An error will be returned if this tag is too long.
<code>flags</code>	The address of an integer interpreted in a manner similar to the <code>versflags</code> field of the <code>wlm_args</code> structure passed to other API routines. The integer pointed to by <code>flags</code> should be initialized with <code>WLM_VERSION</code> . In addition, one or more of the following

values can be ORed to WLM\_VERSION:

**SWLMTAGINHERITFORK** The children of this process will inherit the parent's tag on `fork`.

**SWLMTAGINHERITEXEC** The process will retain its tag after a call to `exec`. Both flags can be set to specify that the children of a tagged process will inherit the tag on `fork` and then retain it on `exec`.

**Return Values:** Upon successful completion, a value of 0 is returned. In case of error, a non zero value is returned.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, "WLM API functions error codes" on page 268.

---

## A.5 Class management

The class management routines are `wlm_read_classes`, `wlm_create_class`, `wlm_change_class`, and `wlm_delete_class`.

### A.5.1 `wlm_read_classes`

**Purpose:** Read the characteristics of superclasses or subclasses.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_read_classes (wlmargs, class_tbl, nclass)
struct wlm_args *wlmargs;
struct class_definition *class_tbl
int *nclass
```

**Description:** The `wlm_read_classes` routine is used to get the characteristics of the superclasses or of the subclasses of a given superclass of a WLM configuration. If the name of a configuration is passed in the `confdir` field, the routine `wlm_read_classes` will read the property files of the classes of the specified configuration. If `confdir` is set to a null string (`\0`), `wlm_read_classes` will read the property files of the in-core classes if WLM is on. If WLM is off, `wlm_read_classes`, with a null string as the configuration name, will fail. Note that if WLM is on and a null string was passed in the `confdir` field, `wlm_read_classes` will return the characteristics of the classes as they are known by WLM at the time of the call. These values may be different from the values in the property files of the configuration pointed to by `/etc/wlm/current`. For instance, if a WLM administrator has modified the property files for the



configuration pointed to by `/etc/wlm/current` but has not refreshed WLM yet. Another example would be if applications dynamically created or modified classes through the API without saving the changes in the current configuration property files. If your application specifically needs to access the properties of the classes as described in the `/etc/wlm/current` configuration, you must specify *current* as the configuration name in `confdir`. If the name of a valid superclass of the given configuration is passed in the `name` field of the *class\_descr* substructure of *wlm\_args*, `wlm_read_classes` will read the property files for the subclasses of this superclass. If a null string (`\0`) is passed in the `name` field, `wlm_read_classes` will read the property files for the superclasses of the WLM configuration described above. When `wlm_read_classes` is successful, the characteristics of the superclasses or subclasses are copied into the array of *class\_definition* structures pointed to by *class\_tbl*. The integer value pointed to by *nclass* indicates the maximum number of class definitions to be copied. Upon successful return from the function, this value reflects the actual number of classes read. If the number of elements copied by `wlm_read_classes` is smaller than the number of elements passed as an argument, this means that all the classes have been read. If it is equal, it may mean that some classes were not copied into the *class\_tbl* array because its size is too small. The maximum number of classes read by `wlm_read_classes` is 32 when reading superclasses and 10 when reading subclasses characteristics. Upon successful return from `wlm_read_classes`, the substructure *class* of type struct *class\_definition* of the structure pointed to by *wlmargs* contains the default values of the various class attributes for the returned set of classes. This operation does not require any special privileges and is accessible to all users.

**Parameters:**

`wlmargs` The address of a struct *wlm\_args* data structure. The following fields of the *wlm\_args* structure and the embedded substructures need to be provided:

- `versflags` Needs to be initialized with `WLM_VERSION`.
- `confdir` The name of a WLM configuration. It must be either the name of a valid subdirectory of `/etc/wlm` or a null string (starting with `\0`).
- `name` The name of a superclass existing in the specified configuration, or a null string.

All the other fields can be left uninitialized.

`class_tbl` The address of an array of structures of type struct *class\_definition*. Upon successful return from

`wlm_read_classes`, this array will contain the characteristics of the classes read.

`nclass` The address of an integer containing the maximum number of elements (class definitions) for `wlm_read_classes` to copy into the array above. If the call to `wlm_read_classes` is successful, this integer will contain the number of elements actually copied.

**Return Values:** Upon successful completion, a value of 0 is returned. If the `wlm_read_classes` routine is unsuccessful, a non 0 value is returned.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, “WLM API functions error codes” on page 268.

## A.5.2 `wlm_create_class`

**Purpose:** Creates a new WLM class.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_create_class (wlmargs)
struct wlm_args *wlmargs;
```

**Description:** The `wlm_create_class` routine creates a new class for a given WLM configuration using the values passed in the data structure of type `struct wlm_args` pointed to by `wlmargs`. If the name of a configuration is passed in the `confdir` field, the routine updates the WLM properties files for the target configuration. When creating the first subclass of a superclass, the routine will create the WLM property files in a subdirectory of `/etc/wlm/<confdir>` with the name of the superclass. The newly-created property files will have entries for the Default and Shared subclasses automatically created in addition to entries for the new subclass. If a null string (`\0`) is passed in the `confdir` field, the new superclass or subclass will be created only in the in-core WLM data. No WLM property file will be updated. The structure of type `struct class_definition`, which is part of `struct wlm_args`, has normally been initialized with a call to `wlm_init_class_definition`. Once this has been done, programmers just need to initialize the fields of this structure that have no default value (for example, the name of the new class) or for which the desired value is different from the default value. For a description of the possible values for all the class attributes and their default values, refer to the description of `wlm.h` in Appendix A.1 on page 261.

The caller must have root authority to create a superclass and must have administrator authority on a superclass to create a subclass of the superclass.

**Parameter:**

<code>wlmargs</code>	The address of the struct <code>wlm_args</code> data structure containing the <code>class_definition</code> structure for the new class to be created. The following fields of the <code>wlm_args</code> structure and the embedded sub-structures need to be provided:
<code>versflags</code>	Needs to be initialized with <code>WLM_VERSION</code> .
<code>confdir</code>	The name of the WLM configuration the new class is to be added to. It must be either the name of a valid subdirectory of <code>/etc/wlm</code> or an empty string (starting with <code>\0</code> ). If the name is a valid subdirectory, the new class data will be added to the given WLM configuration's class description files. If the name is a null string, no description files will be updated. The new class will be created and the data passed to the kernel immediately.
<code>name</code>	The name of the superclass or of the subclass to be created. If this is a subclass name, it must be of the form, <code>supername.subname</code> . There is no default for this field.

All the other fields can be left at their default value if the user does not wish to use specific values.

**Return Values:** Upon successful completion, a value of 0 is returned. If the `wlm_create_class` routine is unsuccessful, a non 0 value is returned.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, “WLM API functions error codes” on page 268.

### A.5.3 `wlm_change_class`

**Purpose:** Changes some of the attributes of a class.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_change_class (wlmargs)
struct wlm_args *wlmargs;
```

**Description:** The `wlm_change_class` routine changes attributes of an existing superclass or subclass. The attributes of the class that can be dynamically modified by a call to `wlm_change_class` are the tier number, class inheritance, class description string, resource shares and limits, and the resource set name (all attributes, except, of course, the name of the class). If the name of a valid configuration is passed in the `confdir` field, the routine updates the WLM property files for the target configuration. If a null string (`\0`) is passed in the `confdir` field, the changes are applied only to the in-core WLM data. No WLM property files will be updated. The structure of type struct `class_definition`, which is part of struct `wlm_args`, should be initialized with a call to `wlm_init_class_definition`. Once this has been done, programmers just need to initialize the fields of this structure that are required (for example, the name of the class to be modified) and the fields corresponding to the class attributes one wants to modify. For a description of the possible values for the various class attributes and their default values, refer to the description of `wlm.h` in Appendix A.1 on page 261.

The caller must have root authority to change the attributes of a superclass and must have administrator authority on a superclass to change the attributes of a subclass of that superclass.

**Parameter:**

<code>wlmargs</code>	The address of the struct <code>wlm_args</code> data structure containing the <code>class_definition</code> structure for the new class to be created. The following fields of the <code>wlm_args</code> structure and the embedded substructures need to be provided:
<code>versflags</code>	Needs to be initialized with <code>WLM_VERSION</code> .
<code>confdir</code>	The name of the WLM configuration the target class belongs to. It must be either the name of a valid subdirectory of <code>/etc/wlm</code> or an empty string (starting with <code>\0</code> ). If the name is a valid subdirectory, the relevant class description files in the given configuration will be modified. If the name is a null string, no description files will be updated. The modified class attributes will be passed immediately to the kernel.
<code>name</code>	The name of the superclass or of the subclass to be modified. If this is a subclass name, it must be of the form <code>supername.subname</code> . There is no default for this field. All the other fields can be left at their initial value as set by <code>wlm_init_class_definition</code> , if the user does not wish to change their current values.

**Return Values:** Upon successful completion, a value of 0 is returned. If the `wlm_change_class` routine is unsuccessful, a non-zero value is returned.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, “WLM API functions error codes” on page 268.

#### A.5.4 `wlm_delete_class`

**Purpose:** Deletes a class.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_delete_class (wlmargs)
struct wlm_args *wlmargs;
```

**Description:** The `wlm_delete_class` routine deletes an existing superclass or subclass. A superclass cannot be deleted if it still has subclasses other than Default and Shared defined. If the name of a valid configuration is passed in the `confdir` field, the routine updates the WLM property files for the target configuration, removing all references to the class to be deleted. If a null string (`\0`) is passed in the `confdir` field, the changes are applied only to the in-core WLM data. No WLM property file will be updated.

The caller must have root authority to delete a superclass and must have administrator authority on a superclass to delete a subclass of the superclass.

**Parameter:**

`wlmargs`            The address of the struct `wlm_args` data structure containing the information about the class to be deleted. The following fields of the `wlm_args` structure and the embedded sub-structures need to be provided:

<code>versflags</code>	Needs to be initialized with <code>WLM_VERSION</code> .
<code>confdir</code>	The name of the WLM configuration the target class belongs to. It must be either the name of a valid subdirectory of <code>/etc/wlm</code> or an empty string (starting with <code>\0</code> ). If the name is a valid subdirectory, the relevant class description files in the given configuration will be modified. If the name is a null string, no description files will be updated. The class will be removed immediately from the kernel WLM data structures.

`name` The name of the superclass or of the subclass to be deleted. If this is a subclass name, it must be of the form *supername.subname*. There is no default for this field.

All the other fields can be left uninitialized for this call.

**Return Values:** Upon successful completion, a value of zero is returned. If the `wlm_delete_class` routine is unsuccessful, a non-zero value is returned.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, “WLM API functions error codes” on page 268.

---

## A.6 WLM management

The WLM management routines are `wlm_set`, `wlm_load`, and `wlm_assign`.

### A.6.1 `wlm_set`

**Purpose:** Changes or queries the state of WLM.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_set (flags)
int *flags;
```

**Description:** The `wlm_set` routine is used to set, change, or query the mode of operations of WLM. The state of WLM can be:

- *OFF*: WLM does not classify processes, monitor, or regulate resource utilization.
- *ON in passive mode*: WLM classifies the processes and monitors their resource usage, but does no regulation.
- *ON in active mode*: This is the normal operating mode where WLM classifies processes, and monitors and regulates their resource usage.

**Parameter:**

`flags` The address of an integer interpreted in a manner similar to the `versflags` field of the `wlm_args` structure passed to the other API routines. The integer pointed to by `flags` should be initialized with `WLM_VERSION`. In addition, one or more of the following values can be ORed to `WLM_VERSION`:

- WLM\_TEST\_ON to just query the state of WLM without altering it.
- WLM\_OFF to turn WLM off.
- WLM\_ACTIVE to turn WLM on in active mode, or transition from passive to active mode.
- WLM\_PASSIVE to turn WLM on in passive mode or transition from active to passive mode.
- WLM\_BIND\_RSETS to request that WLM take the resource set bindings into account.

Not all combinations of the aforementioned flags are valid:

- WLM\_OFF, WLM\_ACTIVE, and WLM\_PASSIVE are mutually exclusive.
- WLM\_BIND\_RSETS is ineffective when used together with WLM\_OFF.
- Only WLM\_TEST\_ON is allowed to non root users.

**Return Values:** Upon successful completion, a value of 0 is returned and the current state of WLM is returned in the integer pointed to by flags. The return value will be WLM\_OFF, WLM\_ACTIVE or WLM\_PASSIVE. When WLM was on in either active or passive mode, the WLM\_BIND\_RSETS flag is added when WLM uses resource sets bindings.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, “WLM API functions error codes” on page 268.

## A.6.2 wlm\_load

**Purpose:** Loads a WLM configuration into the kernel.

**Library:** Workload Manager Library (libwlm.a)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_load (wlmargs)
struct wlm_args *wlmargs;
```

**Description:** The `wlm_load` routine loads into the kernel the property files for the WLM configuration passed in the `confdir` field of the `wlm_args` structure. If no superclass name is given in the name field of the `class_definition` substructure, the routine loads the class properties for all the superclasses of the target configuration. If a superclass name is given, only the subclasses of the given superclass are refreshed. Flags passed in the flags portion of the

versflags field can be used to modify the mode of operation of WLM. The values are identical to the flag values passed to the `wlm_set` API routine. Not all combinations of parameters are allowed, and different combinations may require different levels of privilege as explained below:

- The name of a configuration must be passed in the `confdir` field in order to start or update WLM. `wlm_load` updates or starts WLM using the properties files from the given configuration. Only root can specify the name of a configuration different from the currently active configuration (specified as *current* in `confdir`).
- When WLM is on (the operation is an update), if the name of the configuration passed in the `confdir` field of the `wlm_args` structure is the name of the currently-active configuration, the name of a superclass can be given in the `name` field in order to update only the subclasses of the given superclass. This functionality is accessible to root and to users with administration privileges on the subclasses of the superclass. `wlm_load` cannot be used in this context to alter the state of WLM (start, stop, or switch between active and passive modes).
- If the caller of `wlm_load` has root privileges and does not specify a superclass, the flags passed in `versflags` can be used to alter WLM's mode of operation; start WLM in active or passive mode; switch between active and passive modes, and/or enable/disable the rset bindings.

**Parameter:**

<code>wlmargs</code>	The address of the struct <code>wlm_args</code> data structure containing the <code>class_definition</code> structure. The following fields of the <code>wlm_args</code> structure and the embedded sub-structures can be provided:
<code>versflags</code>	Needs to be initialized with <code>WLM_VERSION</code> . Optionally, some of the flags used when calling <code>wlm_set</code> in order to change the mode of operation of WLM can be given by the root user. The valid values are <code>WLM_ACTIVE</code> , <code>WLM_PASSIVE</code> , and <code>WLM_BIND_RSETS</code> . Of course, <code>WLM_ACTIVE</code> and <code>WLM_PASSIVE</code> are mutually exclusive. The flag, <code>WLM_SAME_STATE</code> , should be used if the application does not wish to change the current mode of operation of WLM.
<code>confdir</code>	The name of the WLM configuration to be loaded into the kernel. It must be either the name of a valid



subdirectory of `/etc/wlm` or the string `current` to refer to the active configuration.

`name` The name of a superclass. This is used to refresh only the subclasses of a given superclass.

**Return Values:** Upon successful completion, a value of 0 is returned. If the `wlm_load` routine is unsuccessful, a non 0 value is returned.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, “WLM API functions error codes” on page 268.

### A.6.3 `wlm_assign`

**Purpose:** Manually assigns processes to a class or cancels prior manual assignments for processes.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_assign (args)
struct wlm_assign *args;
```

**Description:** The `wlm_assign` function is used to:

- Assign a set of processes specified by their process identifiers (pids) or process group identifiers (pgids) to a specified superclass or subclass, thus overriding the automatic class assignment or a prior manual assignment.
- Cancel a previous manual assignment, allowing the processes to be subjected to the automatic assignment rules again.

The target processes are identified by their process ID (pid) or by their process group ID (pgid). The `wlm_assign` routine allows you to specify processes using a list of pids, a list of pgids, or both.

A manual assignment will remain in effect (and a process will remain in its manually assigned class) until:

- The process terminates.
- WLM is stopped. When WLM is restarted, the manual assignments in effect when WLM was stopped are lost.
- The class the process has been assigned to is deleted.
- The manual assignment for the process is canceled.
- A new manual assignment overrides a prior one.

The name of a valid superclass or subclass must be specified to manually assign the target processes to a class. The assignment can be done or canceled at the superclass level, the subclass level, or both. Flags in the `wa_versflags` field described below are used to specify whether the requested operation is an assignment or cancellation, and at which level.

In order to assign a process to a class or cancel a prior manual assignment, the caller must have authority both on the process and on the target class. These constraints translate into the following:

- The user root can assign any process to any class.
- A user with administration privileges on a given superclass (that is, the user or group name matches the user or group names specified in the attributes, `adminuser` and `admingroup`, of the superclass) can manually reassign any process from one of the subclasses of this superclass to another subclass of the superclass.
- A user can manually assign his/her own processes (same real or effective user ID) to a superclass or a subclass for which he/she has manual assignment privileges (that is, the user or group name matches the user or group names specified in the attributes, `authuser`, and `authgroup` of the superclass or the subclass).

This defines three levels of privilege among the persons who can manually assign processes to classes, root being, of course, the highest. In order for a user to modify or terminate a manual assignment, he/she must be at the same level of privilege or higher than the person who issued the last manual assignment.

**Parameter:**

<code>args</code>	The address of the struct <code>wlm_assign</code> data structure containing the parameters for the desired class assignment. The following fields of the <code>wlm_assign</code> structure and the embedded sub-structures can be provided:
<code>wa_versflags</code>	Needs to be initialized with <code>WLM_VERSION</code> . The flags values available, defined in the header file <code>&lt;sys/wlm.h&gt;</code> , are the following: <code>WLM_ASSIGN_SUPER</code> , <code>WLM_ASSIGN_SUB</code> , <code>WLM_ASSIGN_BOTH</code> , <code>WLM_UNASSIGN_SUPER</code> , <code>WLM_UNASSIGN_SUB</code> , and <code>WLM_UNASSIGN_BOTH</code> .
<code>wa_pids</code>	The address of the array containing the process

	identifiers (pid's) of processes to be manually assigned. When this list is empty, a NULL pointer can be passed together with a count of zero (0).
<code>wa_pid_count</code>	The number of elements (pids) in the above array. Could be zero (0) if using only pgid's to identify the processes.
<code>wa_pgids</code>	The address of the array containing the process group identifiers (pid's) of processes to be manually assigned. When this list is empty, a NULL pointer can be passed together with a count of zero (0).
<code>wa_pgid_count</code>	The number of elements (pgids) in the above array. Could be zero (0) if using only pid's to identify the processes. If both pid's and pgid's counts are zero, no process will be assigned, but the operation will be considered successful.
<code>wa_classname</code>	The full name of the superclass, <i>supername</i> , or the subclass, <i>supername.subname</i> , of the class you want to manually assign processes to. The class name field is ignored when canceling an existing manual assignment.

**Return Values:** Upon successful completion, a value of zero (0) is returned. If the `wlm_assign` routine is unsuccessful, a non-zero (0) value is returned. A *partial success* return code will be returned if some of the target processes are not found (to account for process terminations). If none of the processes in the lists can be found, this will be considered an error.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, "WLM API functions error codes" on page 268.

---

## A.7 WLM statistics

The WLM statistics routines are `wlm_get_info` and `wlm_get_bio_stats`.

### A.7.1 `wlm_get_info`

**Purpose:** Read the characteristics of superclasses or subclasses.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_get_info (wlmargs, info, count)
```

```

struct wlm_args *wlmargs;
struct wlm_info *info
int *count

```

**Description:** The `wlm_get_info` routine is used to get the characteristics of the classes defined in the active WLM configuration together with their current resource usage statistics. For a detailed description of the fields of the structure, `wlm_info`, refer to the description of the header file, `<sys/wlm.h>`, in the Files Reference documentation. By default, the scope of the `wlm_get_info` routine is all the superclasses and all the subclasses. This scope can be limited to a subset of the classes using flags in the `versflags` field of `wlm_args` and/or a superclass or subclass name in the `name` field of the substructure, `class_definition` of `wlm_args`. The information related to the superclasses and subclasses within the scope of `wlm_get_info` will be copied to the array of `wlm_info` structures pointed to by `info`. The total number of classes for which information is copied to the array at `info` is limited to the value of the integer pointed to by `count`. If the routine is successful, the value of the integer pointed to by `count` is set to the actual number of classes copied. If the value passed to the routine for the count is equal to zero (0), `wlm_get_info` does not copy any class statistics but sets this count to the number of classes in scope for the specific set of parameters. This is a way of finding out how big an array is needed to get all the information for a given set of classes (superclasses and/or subclasses).

`wlm_get_info` does not require any special privileges and is accessible to all users. `wlm_get_info` will fail if WLM is off.

**Parameters:**

`wlmargs` The address of a struct `wlm_args` data structure. The following fields of the `wlm_args` structure and the embedded sub-structures need to be provided:

- `versflags` Needs to be initialized with `WLM_VERSION`.  
 Optionally, the following flag values can be ORed to `WLM_VERSION`:
- `WLM_SUPER_ONLY`: Limits the scope to superclasses only.
  - `WLM_SUB_ONLY`: Limits the scope to subclasses only.
  - `WLM_VERBOSE_MODE`: Shows the system defined subclasses, Default and Shared, even if

they have not been modified by a WLM administrator.

WLM\_SUPER\_ONLY and WLM\_SUB\_ONLY are mutually exclusive.

name This field must contain either a null string, or the name of a valid superclass or subclass (in the form Super.Sub). This field can be used in conjunction with the flags to further narrow the scope of `wlm_get_info`:

- If the name of a subclass is provided, `wlm_get_info` will return the statistics only for the specified subclass.
- If the name of a superclass is provided and none of the WLM\_SUPER\_ONLY and WLM\_SUB\_ONLY flags are provided, `wlm_get_info` will return the statistics for the specified superclass and all its subclasses.
- If the name of a superclass is provided together with WLM\_SUPER\_ONLY, `wlm_get_info` will return only the statistics for the specified superclass.
- If the name of a superclass is provided together with WLM\_SUB\_ONLY, `wlm_get_info` will return the statistics for all the subclasses of the specified superclass.

All the other fields of the `wlm_args` structure can be left uninitialized.

info The address of an array of structures of type struct `wlm_info`. Upon successful return from `wlm_get_info`, this array will contain the WLM statistics for the classes selected.

count The address of an integer containing the maximum number of elements (of type `wlm_info`) for `wlm_get_info` to copy into the aforementioned array. If the call to `wlm_get_info` is successful, this integer will contain the number of elements actually copied. If the initial value is equal to zero (0), `wlm_get_info` will set this value to the number of classes selected by the specified combination of `versflags` and `name` above.

**Return Values:** Upon successful completion, a value of zero is returned. If the `wlm_get_info` routine is unsuccessful, a non-zero value is returned.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, “WLM API functions error codes” on page 268.

### A.7.2 `wlm_get_bio_stats`

**Purpose:** Read the WLM disk I/O statistics per class or per device.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/types.h>
#include <sys/wlm.h>
int wlm_get_bio_stats (dev, array, count, class, flags)
dev_t dev;
void *array;
int *count;
char *class;
int flags;
```

**Description:** The `wlm_get_bio_stats` routine is used to get the WLM disk I/O statistics. There are two types of statistics available:

- The statistics about disk I/O utilization per class and per devices, returned by `wlm_get_bio_stats` in `wlm_bio_class_info_t` structures.
- The statistics about the disk I/O utilization per device, all classes combined, returned by `wlm_get_bio_stats` in `wlm_bio_dev_info_t` structures.

The type of statistics returned by the function are related to the value of the `flags` argument. The `flags` argument, together with the `dev` and `class` arguments, is used to restrict the scope of the function to a class or a set of classes and/or a device or a set of devices. It is also used to restrict the statistics to superclasses only, subclasses only, and to a set of devices.

`wlm_get_bio_stats` does not require any special privileges and is accessible to all users. `wlm_get_bio_stats` will fail if WLM is off.

**Parameters:**

`flags` Needs to be initialized with `WLM_VERSION`. Optionally, the following flag values can be ORed to `WLM_VERSION`:

- `WLM_SUPER_ONLY`: Limits the scope to superclasses only.
- `WLM_SUB_ONLY`: Limits the scope to subclasses only.

- `WLM_BIO_CLASS_INFO`: Per class statistics requested.
- `WLM_BIO_DEV_INFO`: Per device statistics requested.
- `WLM_BIO_ALL_DEV`: Requests statistics for all devices. When this flag is set, the value passed in the `dev` argument is ignored.
- `WLM_BIO_ALL_MINOR`: Requests statistics for all devices associated with a given major number. When this flag is set, only the major number part of the value passed in the `dev` argument is used.
- `WLM_VERBOSE_MODE`: Shows the system-defined subclasses, Default and Shared, even if they have not been modified by a WLM administrator.

One of the flags, `WLM_BIO_CLASS_INFO` or `WLM_BIO_DEV_INFO` (and only one), must be specified. `WLM_SUPER_ONLY` and `WLM_SUB_ONLY` are mutually-exclusive.

<code>dev</code>	<p>Device identification (major, minor) of a disk device.</p> <ul style="list-style-type: none"> <li>• If <code>dev</code> is equal to 0, the statistics for all devices are returned (even if <code>WLM_BIO_ALL_DEV</code> is not specified in the flags argument).</li> <li>• If <code>dev</code> is not equal to 0 and <code>WLM_BIO_ALL_MINOR</code> is specified in the flags argument, the statistics for all disk devices with the same major number specified in <code>dev</code> are returned.</li> <li>• If <code>dev</code> is not equal to 0 and <code>WLM_BIO_ALL_MINOR</code> is not specified in the flags argument, only the statistics for the disk device with the major and minor numbers specified in <code>dev</code> are returned.</li> </ul>
<code>array</code>	<p>Pointer to an array of <code>wlm_bio_class_info_t</code> structures (when <code>WLM_BIO_CLASS_INFO</code> is specified in the flags argument) or an array of <code>wlm_bio_dev_info_t</code> structures (when <code>WLM_BIO_DEV_INFO</code> is specified in the flags argument).</p>
<code>count</code>	<p>The address of an integer containing the maximum number of elements to be copied into the array above. If the call to <code>wlm_get_bio_stats</code> is successful, this integer will contain the number of elements actually copied.</p>
<code>class</code>	<p>A pointer to a character string containing the name of a superclass or subclass. If <code>class</code> is a pointer to an empty string (<code>""</code>), the information for all classes is returned. The class</p>

parameter is taken into account only when the flag, `WLM_BIO_CLASS_INFO`, is set.

**Return Values:** Upon successful completion, a value of 0 is returned and the value pointed to by *count* is set to the number of elements copied into the array of structures pointed to by *array*. If the `wlm_get_bio_stats` routine is unsuccessful, a non-zero value is returned.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, “WLM API functions error codes” on page 268.

---

## A.8 WLM classification

The WLM classification routines are `wlm_check` and `wlm_classify`.

### A.8.1 `wlm_check`

**Purpose:** Checks automatic assignment rules and/or determines the class a process with a specified set of attributes will be classified in.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_assign (config)
char *config;
```

**Description:** The `wlm_check` function checks the coherency of the assignment rules files (syntax, existence of the classes, validity of user and group names, application path names, and other consistency checks) for the configuration whose name is passed as an argument. If *config* is a null pointer or points to an empty string, `wlm_check` performs the checks on the configuration files in the configuration pointed to by `/etc/wlm/current`.

**Parameter:**

`config`            A pointer to a character string. This pointer should be:

- The address of a character string representing the name of a valid configuration (a subdirectory of `/etc/wlm`)
- A null pointer
- A pointer to a null string (“”)

If *config* is a null pointer or a pointer to a null string, the configuration files in the directory pointed to by `/etc/wlm/current` (active configuration) will be checked for errors. Otherwise, the configuration files in the directory,



/etc/wlm/<config\_name>, will be checked.

**Return Values:** Upon successful completion, a value of 0 is returned. If the `wlm_check` routine is unsuccessful, a non-zero value is returned.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, “WLM API functions error codes” on page 268.

### A.8.2 `wlm_classify`

**Purpose:** Given a list of process attributes, `wlm_classify` determines which class or classes this process will be assigned to.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_classify (config, attributes, class, len)
char *config;
char *attributes;
char *class;
int *len;
```

**Description:** This routine must receive the name of a valid configuration and a set of process attributes in a format identical to the one of the rules files (assignment rules). On output, the names of the classes are copied into the area pointed to by `class`. The integer pointed to by `len` contains the size of the class names area on input, and the number of matches on output. If the area pointed to by `class` is not big enough to contain the names of all the potential matches, an error is returned.

The normal use of this routine is to explicitly provide all the process classification attributes; user name, group name, application path name, and tag (when applicable). This should give a match to a single class, but, in order to implement what-if scenarios, the interface allows some of the attributes to be unspecified by entering a hyphen (-) instead. This may lead to the process being assigned to multiple classes, depending on the values of the unspecified attributes. If all the attributes are left unspecified, an error is returned.

The attributes string is provided in a format identical to the one of the attributes in the rules file; a list of attribute values separated by spaces. The order of the attributes in the assignment rules is:

1. Reserved: must be a hyphen (-)
2. User name
3. Group name

4. Application pathname
5. Process type
6. Tag

A valid specification for the attributes string could be:

```
- bob staff /usr/bin/emacs -
```

or:

```
- - devlt /usr/bin/cc -
```

The class names returned by the function in the class buffer will be fully-qualified, null-terminated class names of the form, *supername.subname*.

This function does not require any special privileges and can be called by all users.

**Parameters:**

<code>config</code>	A pointer to a string containing the name of a valid WLM configuration (the name of a subdirectory of /etc/wlm). If a null string (\0) is given, <code>wlm_classify</code> will use the in-core class and rules definitions.
<code>attributes</code>	The address of a string, with the format described above, containing a list of values for the process attributes used for automatic classification of processes.
<code>class</code>	A pointer to a buffer where the name of the class or classes the process could be assigned to are returned as consecutive, null-terminated character strings.
<code>len</code>	A pointer to an integer containing the length, in bytes, of the buffer pointed to by <code>class</code> when calling <code>wlm_classify</code> , and the actual number of class names copied into the class buffer upon successful return.

**Return Values:** Upon successful completion, a value of zero is returned. In case of error, a non zero value is returned. When a non-zero value is returned, the content of the class buffer and the value of the integer pointed to by `len` are unspecified.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, “WLM API functions error codes” on page 268.

---

## A.9 WLM accounting

The WLM accounting initialization routines per class are `wlm_initkey`, `wlm_class2key`, `wlm_key2class`, and `wlm_endkey`.

### A.9.1 `wlm_initkey`

**Purpose:** Allocates and initializes the classes to keys translation table.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_initkey(struct wlm_args *args, void **ctx)
```

**Description:** The `wlm_initkey` routine allocates a block of memory, builds the keys to class names translation table and returns its address into the `ctx` argument.

**Parameters:**

`args` The address of the struct `wlm_args` data structure containing the structure to be initialized. Only two fields need to be initialized in the `wlm_args` structure pointed to by `args`:

`confdir` Specifies the null-terminated name of the WLM configuration to be searched (the name can be "current" to specify the current configuration). If the configuration name passed is an empty string (starts with '\0'), then all the configurations in `/etc/wlm` are searched.

`versflags` Initialized with `WLM_VERSION` and optionally `WLM_MUTE`.

**Return Values:** If the `wlm_initkey` routine is successful, a value of 0 is returned. If the `wlm_initkey` routine is unsuccessful, an error code is returned.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, "WLM API functions error codes" on page 268.

### A.9.2 `wlm_class2key`

**Purpose:** Class name to key translation.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_class2key(struct wlm_args *args, wlm_key_t *key)
```

**Description:** The `wlm_class2key` routine generates a 64 bit numeric key from a WLM class name. This routine is provided for applications gathering high volumes of per class usage statistics or accounting data. This routine allows those applications to save storage space by compressing the class name (up to 34 characters long) into a 64 bit integer. The `wlm_key2class` routine can then be used to get the key to class name conversion for data reporting purposes.

The `wlm_class2key` routine does not check that the specified class exists. It just checks that the name has a valid format for a class name. When successful, the routine returns the corresponding key (sometimes also called signature) in the area pointed to by the return values.

**Parameters:**

<code>args</code>	The address of the struct <code>wlm_args</code> data structure containing the structure to be initialized. Only two fields need to be initialized in the <code>wlm_args</code> structure pointed to by <code>args</code> :
<code>cl_def.data.descr.name</code>	Specifies the null terminated full name of the class <code>&lt;super_name&gt;.&lt;subname&gt;</code> for a subclass).
<code>versflags</code>	Initialized with <code>WLM_VERSION</code> and optionally <code>WLM_MUTE</code> .

**Return Values:** If the `wlm_class2key` routine is successful, a value of 0 is returned. If the `wlm_class2key` routine is unsuccessful, an error code is returned.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, “WLM API functions error codes” on page 268.

### A.9.3 `wlm_key2class`

**Purpose:** Retrieves a class name from a key.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_key2class(struct wlm_args *args, wlm_key_t key, void *ctx)
```

**Description:** The `wlm_key2class` routine retrieves a class name from a 64 bit key calculated using the `wlm_class2key` routine. The key to class translation is made by going through the WLM configuration files for the configuration named in the `wlm_args` structure pointed to by `wlmargs` (or all the WLM

configuration files, if no configuration name is given), and translating all the class names to a 64 bit key until the matching key is found.

This process is time consuming. For applications which need to translate several 64 bit keys back to class names, WLM offers the routines `wlm_initkey` and `wlm_endkey` that can be used in conjunction with `wlm_key2class` to speed up the searches. The `wlm_initkey` routine allocates a block of memory, calculates the keys corresponding to the class names in the configuration(s) in scope, stores the names with the corresponding keys in the memory buffer, and returns its address. This address can then be passed to `wlm_key2class` in the `ctx` argument, so that `wlm_key2class` only needs to search through the memory buffer. When all the keys have been translated into class names, the application calls `wlm_endkey`, which will free the memory buffer. Alternatively, for an application translating only one key, it is possible to directly call `wlm_key2class` with a null pointer in the `ctx` argument. This will cause `wlm_key2class` to internally call `wlm_initkey` and `wlm_endkey`.

The way the class names are retrieved by going through the WLM configuration files implies that if a class has been deleted between the time the class name was converted into a key and the call to `wlm_key2class`, the name corresponding to the key is not found, and `wlm_key2class` returns an error.

#### Parameters:

<code>args</code>	The address of the struct <code>wlm_args</code> data structure containing the structure to be initialized. Four fields need to be initialized in the <code>wlm_args</code> structure pointed to by <code>args</code> :
<code>confdir</code>	This field needs to be initialized as described in <code>wlm_initkey</code> if <code>wlm_initkey</code> has not been previously invoked ( <code>ctx == NULL</code> ). Otherwise, the <code>confdir</code> field is ignored.
<code>versflags</code>	This field needs to be initialized with <code>WLM_VERSION</code> , or optionally <code>WLM_MUTE</code> .
<code>ctx</code>	The context handler returned by <code>wlm_initkey</code> (if <code>wlm_initkey</code> has been called previously), or a NULL pointer otherwise. If a null pointer is passed, <code>wlm_key2class</code> will automatically invoke <code>wlm_initkey</code> (passing it the <code>wlmargs</code> argument), retrieve the class name corresponding to the key, and free the allocated context area using <code>wlm_endkey</code> .
<code>key</code>	The search key.

**Return Values:** When the `wlm_key2class` operation is successful, the first class name matching the value of the key is returned in the name sub-field of the `wlm_args` structure pointed to by `wlmargs`.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, “WLM API functions error codes” on page 268.

#### A.9.4 `wlm_endkey`

**Purpose:** Frees the classes to keys translation table.

**Library:** Workload Manager Library (`libwlm.a`)

**Syntax:**

```
#include <sys/wlm.h>
int wlm_endkey(struct wlm_args *args, void *ctx)
```

**Description:** The `wlm_endkey` routine frees the classes to the keys translation table. The memory area pointed to by `ctx` is freed.

**Parameters:**

<code>args</code>	The address of the struct <code>wlm_args</code> data structure containing the structure to be initialized:
<code>versflag</code>	This field is the only field in the structure that needs to be initialized with <code>WLM_VERSION</code> , or optionally <code>WLM_MUTE</code> .
<code>ctx</code>	Points to the memory area to be freed.

**Return Values:** When the `wlm_endkey` operation is successful, it returns a value of 0, and, if unsuccessful, returns an error code.

**Error Codes:** For a list of the possible error codes returned by the WLM API functions, see Section A.2, “WLM API functions error codes” on page 268.

## Appendix B. Sample workload program

This appendix describes the sample program that was used to generate workload during the development of this redbook. It launches a number of CPU bound threads, creates network traffic, allocates memory, and generates disk I/O.

The sample program hog.c:

```
static char sccsid[] = "@(#)93 1.0 hog.c 8/30/99 11:30";
/*
 * COMPONENT_NAME: hog
 *
 * WRITTEN BY: Tim Leo
 *
 * FUNCTIONS: Exercises SMP CPU Load (utilization), Disk I/O and Memory Usage
 *
 *           To be used for testing AIX WLM (Workload Manager)
 *
 * OBJECT CODE ONLY SOURCE MATERIALS
 * (C) COPYRIGHT International Business Machines Corp. 1989, 1991
 * All Rights Reserved
 *
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * Copyright (c) 1980 Regents of the University of California.
 * All rights reserved. The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 */
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <strings.h>
#include <errno.h>
#include <stdlib.h>
#include <locale.h>
#include <sys/limits.h>
#include <nl_types.h>
#include <sys/param.h>

/* Start of Global Vars *****/
char *cp; /* Current Program name */
int login; /* true if invoked as login shell */
char firstchar; /* first char of name of prog invoked as */

extern int optind;
extern char *optarg;

usage() /* Prints help info for hog Program */
```

```

{
    printf("%s: usage is %s [-t thread_count [-n iptarget]] [-m memory_valu
e] [-d] [-?] \n",cp,cp);
    printf("\n\nARGUMENTS: \n");
    printf("-t thread_count : Launch a number of CPU Bound Threads \n");
    printf("                where thread_count is an integer specifying\n
");
    printf("                the number of threads to launch.\n\n");
    printf("-m memory_value : Determines the amount of working memory to be
allocated and\n");
    printf("                utilized by the main program to be touched co
ntinuously,\n");
    printf("                where memory_value is an integer specifying t
he number \n");
    printf("                of MB (MegaBytes) to allocate.\n\n");
    printf("-d                : Generates Disk I/O.\n\n");
    printf("-n iptarget       : Threads should generate network I/O.\n");
    printf("                where iptarget is either an IP address or hos
tname\n");
    printf("                (the default iptarget is loopback) \n");
    printf("-?                : Obtains this screen of help info and exit. \n
");
}

/* More Global Vars *****/

int loop_stat;
pthread_mutex_t m;
char *waste_of_space=NULL;
size_t megs=0;

struct arg {
    char *string;
    int net;
};
typedef struct arg arg_t;

/* Sample Memory Grabber Thread Routine *****/
void *Memory_Thread(void *x)
{
    printf("Starting memory scan...\n");
    while(1)
    {
        if ((waste_of_space =(char *)calloc(megs,1))==NULL)
            printf("%s: Could not allocate memory...errno was %d\n",cp,errno);
        else {
            system("date");
            printf("%s has allocated a %d byte array of storage...\n",cp, (int)megs);
            sleep(10);
            free(waste_of_space);
        }
    }
}

```



```

/* Sample Disk I/O Generator Thread Routine *****/
void *Gen_File_IO(void *x)
{
    while (1)
    {
        system ("dd if=/unix of=/wlm_fs1/test1 count=10000 2>/dev/null" );
        system ("dd if=/wlm_fs1/test1 of=/wlm_fs2/test2 count=10000 2>/dev/null");
        /* system ("run_disks 4"); */
        system ("date");
        printf("%s: Finished a Disk Cycle...\n",cp);
    }
    pthread_exit( (void *)1);
}

/* Sample CPU Bound Thread Routine *****/
void *Thread(void *x)
{
    int l;
    while (1)
    {
        /* l=0; while (l < 1000) l++; Delay loop */
        usleep(5);
        pthread_mutex_lock(&m);
        loop_stat++;
        pthread_mutex_unlock(&m);
        if ((loop_stat%1000)== 0){
            system ("date");
            printf("%d transactions so far...\n",loop_stat);
        }
        /* Turn on network traffic */
        if (((arg_t*)x)->net==1) system( ((arg_t *)x)->string);
    }
    pthread_exit( (void *)1);
}

/* hog main program *****/
main(argc, argv)
    char **argv;
    int argc;
{
    /* Start of Local Vars for Main *****/
    #ifdef PATH_MAX
    #undef PATH_MAX
    #endif
    #define PATH_MAX 257 /* Maximum path +\0 */

```

```

register int i, j;
    int tp, thread_max;
    int mflag=0;
int dflag=0;
    int tflag=0;
    pthread_t h_th[32000];
    pthread_t d_th;
    pthread_t m_th;
    arg_t arg_th;
    char *iptarget="`hostname`";
    char temp[PATH_MAX];

    arg_th.net=0;
    system("date");
    (void) setlocale (LC_ALL, "");

    login = (argv[0][0] == '-');
    cp = rindex(argv[0], '/');
    firstchar = login ? argv[0][1] : (cp==NULL) ? argv[0][0] : cp[1];
    cp = argv[0]; /* for Usage */

    printf("\n");
    printf("%s Beta test version.\n",cp);
    printf(" (C) COPYRIGHT International Business Machines Corp. 1999\n");
    printf(" All Rights Reserved\n");
    if (argc==0) {usage(); exit(1);}
    while ((tp = getopt(argc, argv, "t:n:m:d?" )) != EOF)
        switch(tp) {
            case 't':
                if (tflag==1) { printf("%s: Multiple -t switch
ignored \n",cp); break; }

                if (*optarg==0) {
printf(
                    "%s: The number of threads to launch was not
specified with the '-t' option \n", cp);
                    printf("%s: By default 1 thread will be laun
ched.\n",cp);

                    thread_max=1;
                }
                else thread_max=atoi(optarg);
                /* check for errors here */
                if ((thread_max < 1) || (thread_max > 32000)) {
                    printf(
                        "%s: invalid number of threads requested (mu
st be between 1 and 32000)\n",cp);
                    usage();
                    exit(1);
                }
                tflag=1;
                break;

            case 'n':
                if (tflag==0) {printf("%s: -t switch must be sp
ecified first to use -n option\n",cp);
                    usage(); exit(2);
                }
        }

```

```

if (arg_th.net==1) { printf("%s: Multiple -n sw
it ch ignored \n",cp); break; }
        if (*optarg != 0) iptarget = optarg;
        if(sprintf(temp,"spray %s -l 1024 -c 50 2> /dev
/null > /dev/null",iptarget)) printf("%s: spr ift erno was %d\n %s\n",cp,erno
,temp);
        arg_th.string=temp;
arg_th.net=1;
        break;

        case 'm':
        if (mflag==1) { printf("%s: Multiple -m switch
ignored \n",cp); break; }

        if (*optarg==NULL) {
        printf(
"%s: Memory value must be specified with 'm'
option \n", cp);

        usage();
        exit(1);
        }
        else megs=atoi(optarg);
        /* check for errors here */
        if ((megas < 0) || (megas >= 64000)) {
        printf(
"%s: invalid memory value requested (must be
between 0 and 64000) MB\n",cp);
        usage();
        exit(1);
        }
        megs *= (size_t) (1024*1024);
        mflag=1;
        break;

        case 'd':
        if (dflag==1) { printf("%s: Multiple -d switch
ignored \n",cp); break; }
        dflag=1;
break;

        case '?':
        usage();
        exit(1);
        break;

        default:
        printf("%s: Bad flag '%s' option ignored\n",cp,
tp);
        }
/* End of argument finder */
/* Debug routine */
#ifdef DEBUG
printf("\n%s: debug active : thread_max=%d, tflag=%d, mflag=%d, dflag=%d \n meg
s=%d, arg_th.string=%s, arg_th.net=%d \n Normal debug exit \n",cp,thread_max,tf
lag,mflag, dflag, megs, arg_th.string, arg_th.net);
exit(0);
#endif

```

```

printf("\n%s: Configuration Summary \n",cp);
/* CPU Bound Thread Launch Stuff */
if (tflag==1) {
    pthread_mutex_init(&m, NULL);
    for (i=0,j=1;i<thread_max;i++) {
        if (pthread_create(&h_th[i], NULL, Thread, &arg_th))
        {
            j=0;
            printf("Launched %d threads so far...Error Launching more, error was %d\n",i,errno);
            break;
        }
    }
    if (j==1) printf("Launched %d thread(s) \n",--i);
} else printf("No CPU Bound Threads Launched...\n");

/* Disk I/O Stuff */
if (dflag==1)
    if (pthread_create(&d_th, NULL, Gen_File_IO, &arg_th))
        printf("%s: Couldn't create Disk I/O Thread...errno was %d\n",cp,errno);
    else printf("Launched I/O Generator Thread \n");
/* End of Thread Launching */

/* Memory Use Stuff */
if (mflag==1) {
    if (pthread_create(&m_th, NULL, Memory_Thread, &arg_th))
        printf("%s: Couldn't create Memory Thread...errno was %d\n",cp,errno);
    else printf("Launched Memory Allocator Thread \n");
}
pthread_exit(0);
printf("%s:Normal Exit",cp);
system("date");
exit(0);
}

```

This script was compiled with the following make file:

```

#Make file for loadgen benchmarks.

clean:
    rm dssserver oltpserver backupserver loadgen

all: hog.c
    cc_r -g -o loadgen hog.c -bD:0x80000000 -lm
    cp loadgen /home/dssadm/dssserver
    cp loadgen /home/oracle/oltpserver
    cp loadgen /home/adsm/backupserver

(0) itsosrv1:/wlm/scripts#

```

- Three users were created to run this test: *oracle*, *dssadm*, and *adsm*.
- Two groups were created: *dba* and *admin*.
- Two filesystems were created on different disks: */wlm\_fs1* and */wlm\_fs2*.
- Due to the test system being a 12-way SMP with 1 GB Memory, the arguments displayed in the next screenshot were chosen to start the different workloads; *OLTP*, *DSS*, and *backup*.

The goal was that the OLTP workload always has enough resources. This workload mainly consumes CPU and memory resources and is, therefore, competing with the backup and the DSS workload for these resources.

The backup workload consumes mainly disk I/O resources next to CPU and memory.

The DSS workload consumes CPU, memory, and disk I/O resources.

```
(0)itsosrv1:/# cat /home/oracle/oltp.sh
oltpserver -t 100 -n 9.3.240.10 -m 128 >> ~oracle/oracle.log
(0)itsosrv1:/# cat /home/dssadm/dss.sh
dssserver -t 50 -m 256 -d >> ~dssadm/dss.log
(0)itsosrv1:/# cat /home/adsm/back.sh
backupserver -t 25 -m 256 -d >> ~adsm/back.log
(0)itsosrv1:/#
```

Parameters used to start the different workloads:

- |                 |   |
|-----------------|---|
| -t thread_count | Launch a number of CPU Bound threads where <i>thread_count</i> is an integer specifying the number of threads to launch.  |
| -n iptarget     | Threads should generate network I/O where <i>iptarget</i> is either an IP address or hostname.  |
| -m memory_value | Determines the amount of working memory to be allocated and utilized by the main program. Memory will be touched continuously, where <i>memory_value</i> is an integer specifying the number of MB (Megabytes) to allocate. |
| -d              | Generates disk I/O.   |

After defining the parameters, the three scripts, *oltp.sh*, *dss.sh*, and *back.sh*, were started by executing the *start.sh* script:

```
(0) itsosrv1:/wlm/scripts# pg start.sh
/usr/samples/kernel/vmtune -P 30
su - oracle -c "~oracle/oltp.sh & "
su - dssadm -c "~dssadm/dss.sh & "
su - adsm -c "~adsm/back.sh & "
(0) itsosrv1:/wlm/scripts#
```

**Important:**

In `/etc/security/limits` the `data file` entry was set to -1 (for unlimited).

**Suggestions:**

It proved to be helpful to first take the actual WLM configuration and run WLM in passive mode. Get a performance report for all classes every 10 seconds and this for five minutes as shown in the following screenshots:

```
^C(130) itsosrv1:/# wlmstat 10 30
CLASS CPU MEM BIO
Unclassified 0 0 0
Unmanaged 0 0 0
Default 0 0 0
Shared 0 1 0
System 0 8 0
oltp 0 0 0
oltp.Default 0 0 0
oltp.Shared 0 0 0
oltp.spray 0 0 0
dss 0 0 0
backup 0 0 0
.....
.....
```

To collect more detailed information, the following statistics can be run:

```
(0) itsosrv1:/# wlmstat -c -v
      CLASS tr i #pr CPU sha min smx hmx des rap urap pri
Unclassified 0 0 1 0 -1 0 100 100 100 0 97 10
Unmanaged 0 0 0 0 -1 0 100 100 0 0 97 10
Default 0 0 1 0 -1 0 100 100 0 0 97 97
Default.Default 0 0 1 0 1 0 100 100 100 100 48 48
Default.Shared 0 0 0 0 -1 0 100 100 0 0 96 96
Shared 0 0 0 0 -1 0 100 100 0 0 97 97
Shared.Default 0 0 0 0 1 0 100 100 100 100 48 48
Shared.Shared 0 0 0 0 -1 0 100 100 0 0 96 96
System 0 0 44 0 10 10 100 100 100 100 0 0
System.Default 0 0 44 0 1 0 100 100 100 100 0 0
System.Shared 0 0 0 0 -1 0 100 100 0 0 48 48
oltp 0 0 0 0 50 0 100 100 47 100 0 0
oltp.Default 0 0 0 0 -1 0 100 100 100 0 23 23
oltp.Shared 0 0 0 0 -1 0 100 100 0 0 23 23
oltp.spray 1 0 0 0 30 0 100 100 61 100 97 97
dss 0 0 1 0 20 0 100 100 100 100 0 0
dss.Default 0 0 1 0 1 0 100 100 100 100 0 0
dss.Shared 0 0 0 0 -1 0 100 100 0 0 48 48
backup 0 0 0 0 35 0 100 100 63 100 0 0
backup.Default 0 0 0 0 1 0 100 100 100 100 0 0
backup.Shared 0 0 0 0 -1 0 100 100 0 0 48 48
(0) itsosrv1:/#
```

```
(130) itsosrv1:/# wlmstat -m -v
      CLASS tr i #pr MEM sha min smx hmx des rap urap npg
Unclassified 0 0 1 0 -1 0 100 100 100 0 511 0
Unmanaged 0 0 0 0 -1 1 100 100 0 100 0 2
Default 0 0 1 0 -1 0 100 100 0 0 511 240
Default.Default 0 0 1 0 1 0 100 100 100 100 255 240
Default.Shared 0 0 0 0 -1 0 100 100 0 0 510 0
Shared 0 0 0 1 -1 0 100 100 100 0 511 3230
Shared.Default 0 0 0 1 1 0 100 100 100 98 260 3230
Shared.Shared 0 0 0 0 -1 0 100 100 0 0 510 0
System 0 0 44 8 10 10 100 100 99 100 0 30720
System.Default 0 0 44 8 1 0 100 100 100 85 38 30753
System.Shared 0 0 0 0 -1 0 100 100 0 0 255 0
oltp 0 0 0 0 50 0 100 100 44 100 0 0
oltp.Default 0 0 0 0 -1 0 100 100 100 0 127 0
oltp.Shared 0 0 0 0 -1 0 100 100 0 0 127 0
oltp.spray 1 0 0 0 30 0 100 100 86 100 512 0
dss 0 0 1 0 20 0 100 100 30 100 0 0
dss.Default 0 0 1 0 1 0 100 100 100 100 0 0
dss.Shared 0 0 0 0 -1 0 100 100 0 0 255 0
backup 0 0 0 0 35 0 100 100 77 100 0 0
backup.Default 0 0 0 0 1 0 100 100 100 100 0 0
backup.Shared 0 0 0 0 -1 0 100 100 0 0 255 0
(0) itsosrv1:/#
```

After the first run, the shares and tiers were changed, and further observations with `wlmstat` were made before WLM was turned into active mode.

**General recommendations**

An easy way to analyze a system running WLM is:

1. Start with a simple model.
2. Run WLM in passive mode.
3. Do some refinements.
4. Repeat the last two steps a few times.
5. Run WLM in active mode.
6. Collect statistics, via `wlmstat`, to see if the defined goals are achieved.
7. Modify shares, rules, and tiers.
8. Go to step 6.
9. Decide which is your best WLM configuration.



---

## Appendix C. Sample Korn shell scripts for manual assignment

In this appendix, the scripts used for the manual assignment examples of Chapter 5, “Manual assignment” on page 177 are listed. They are also available for practical use on the floppy disk provided with the redbook.

---

### C.1 Oracle example script

This is the script used in the Oracle example described in Section 5.3, “Oracle database example” on page 188:

```
#!/bin/ksh
# Sample script to perform manual assignment of processes whose different
# instances can be differentiated by their output in ps -ef.
# Examples of this kind of processes are ORACLE database instances.
#
# Create a configuration file /etc/wlm/ma.conf with the following format:
# One line for each combination of:
#           <Instance name> <Class> <Inheritance>
# where:
#           o Instance Name is the ORACLE instance.
#           o Class is the name of the class to assign the processes to;
#             Either 'supername' for superclasses or 'supername.subname'
#             for subclasses.
#           o Inheritance is a flag, which should be set to yes if you
#             want all processes belonging to a process group, whose
#             leader is the process being manually assigned, to be
#             manually assigned too, or no, otherwise.
# MANUAL is an array of three positions, which one of them being:
#           o Position 0: Instance name.
#           o Position 1: Class name.
#           o Position 2: Inheritance flag.

##
# DIRECTORIES
##
WLMDIR=/etc/wlm

##
# VARIABLES
##
CONFFILE=$WLMDIR/ma.conf
PATH=/usr/bin:/usr/sbin:$PATH

##
# FUNCTIONS
##
```

```

getpids()
{
    echo `ps -ef | grep $1 | grep -v grep | awk '{ print $2 }'` | sed \
        `s/ /,/g`
}

##
# MAIN
##
(while read LINE
do
    set -A MANUAL $LINE

    echo "Changing the inheritance attribute on class ${MANUAL[1]}..."
    OLDINH=`lsclass -f ${MANUAL[1]} | grep inheritance | awk '{ print \
        $3 }' | sed "s/\"//g"`
    [ ! "$OLDINH" ] && OLDINH="no"
    chclass -a inheritance=${MANUAL[2]} ${MANUAL[1]}
    echo "Refreshing WLM..."
    wlmcntrl -u

    echo "Getting PIDS' list for instance ${MANUAL[0]}..."
    PIDLIST=$(getpids ${MANUAL[0]})

    if [ -z "$PIDLIST" ]
    then
        echo "No processes found for class ${MANUAL[1]}, skipping \
            assignment ..."
    else
        echo "Manually assigning the processes to class ${MANUAL[1]}..."
        wlmassign ${MANUAL[1]} $PIDLIST
    fi

    echo "Resetting old inheritance value on class ${MANUAL[1]}..."
    chclass -a inheritance="$OLDINH" ${MANUAL[1]}
    echo "Refreshing WLM..."
    wlmcntrl -u
done
) < $CONFFILE

```

---

## Appendix D. Sample program for application tag

In this appendix, the program, `settag.c`, used for the application tag example of Section 6.1.3 on page 197 is listed. It is also available for practical use on the floppy disk provided with the redbook.

---

### D.1 `settag.c`

```
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <sys/wlm.h>

/* Program for launching and tagging an application */
main (argc,argv)
char **argv;
int argc;
{
    int rc,flags;
    if (argc != 3) {
        usage(argv[0]);
        exit(1);
    }
    flags= WLM_VERSION|SWLMTAGINHERITFORK|SWLMTAGINHERITEXEC;
    if (wlm_initialize(WLM_VERSION)) {
        perror("wlm_initialize");
        exit(1);
    }
    if (wlm_set_tag(argv[1],&flags)) {
        perror("wlm_set_tag");
        exit(2);
    }
    if (execlp(argv[2],argv[2],0)) {
        perror("execlp"); printf("Problem launching app...\n");
        exit(3);
    }
    exit(0);
}

usage(char *cp)
{
    printf("\n %s takes 2 arguments:\n",cp);
    printf("Usage: %s tag_name program_name \n",cp);
    printf("where: tag_name is the rule tag that program_name will inherit \
\n");
}
```



---

## Appendix E. Sample for CPU resource usage calculation

A sample spread sheet that contains the CPU resource usage data of Applications A, B, C, and D from Section 7.3.2, "Examples" on page 214 is listed below. This was obtained by monitoring Applications A, B, C, and D respectively, which ran on a system that has a capacity of 10,000 tpm (transaction per minute), separately. The resource usage was measured for each application at 10 minute intervals, for 500 minutes. The unit of the measurement is percentage.

Because the system capacity is 10,000 tpm, each percentage value in the spread sheet is easily converted, by multiplying by 100, to the actual tpm value that was consumed by each application at the moment of measurement.

This data is not from monitoring a real system, but was simulated as a general example.

Time unit	Application A	Application B	Application C	Application D	Sum of A, B, C, D
1	11	34	2	18	65
2	14	32	3	19	68
3	12	33	5	15	65
4	16	32	3	13	64
5	25	25	4	16	70
6	39	22	2	12	75
7	56	18	1	15	90
8	21	22	3	19	65
9	12	23	4	14	53
10	9	26	2	13	50
11	15	24	1	16	56
12	18	23	2	17	60
13	11	17	3	16	47
14	21	19	2	18	60
15	46	18	1	16	81
16	51	15	2	14	82
17	16	16	3	16	51
18	12	18	2	15	47
19	17	19	3	17	56
20	18	20	4	16	58
21	16	18	5	16	55
22	15	21	3	18	57
23	42	22	2	19	85
24	54	21	4	18	97
25	35	25	2	19	81
26	22	24	1	17	64
27	21	27	2	18	68
28	18	28	2	17	65
29	15	34	3	18	70
30	23	32	1	16	72
31	21	33	3	15	72
32	15	33	2	16	66
33	12	31	3	12	58
34	8	26	3	17	54
35	6	25	2	14	47
36	7	23	1	13	44
37	8	22	2	14	46
38	6	24	5	13	48
39	6	19	35	12	72
40	7	18	54	12	91
41	8	15	57	13	93
42	6	17	55	11	89
43	5	17	56	12	90
44	8	16	57	11	92
45	9	15	54	11	89
46	8	15	53	12	88
47	7	16	55	11	89
48	7	16	55	11	89
49	7	14	56	13	90
50	6	13	51	11	81
Total	868	1116	736	745	3465

---

## Appendix F. Using the additional material

This redbook contains additional material in diskette format. See the appropriate section below for instructions on using or downloading each type of material.

---

### F.1 Using the diskette

The diskette that accompanies this redbook contains the following:

<i>File name</i>	<i>Description</i>
<b>wlm_db2.pdf</b>	PDF file that describes the DB2 tests that were run during the redbook project as described in Section 5.4, "DB2 UDB" on page 190.
<b>ma_oracle.sh</b>	Sample script for manual assignment with Oracle
<b>settag.c</b>	Sample source code for Application Tag setting
<b>settag</b>	Sample binary for Application Tag setting

#### F.1.1 System requirements for using the diskette

The following system configuration is recommended for optimal use of the diskette.

<b>Operating System:</b>	AIX 5L for Power Version 5.1 or higher
<b>Processor:</b>	IBM RS/6000 or IBM @server pSeries

#### F.1.2 How to use the diskette

You can access the contents of the diskette by extracting the files on the diskette with `tar -xvf /dev/fd0` into your current directory.

---

### F.2 Locating the additional material on the Internet

The diskette material associated with this redbook is also available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG245977>

Alternatively, you can go to the IBM Redbooks Web site at:

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

Select the **Additional materials** and open the directory that corresponds with the redbook form number.





---

## Appendix G. Special notices

This publication is intended to help system administrators and technical support specialists implement and use AIX Workload Manager efficiently. The information in this publication is not intended as the specification of any programming interfaces that are provided by AIX 5L. See the PUBLICATIONS section of the IBM Programming Announcement for AIX 5L for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers

attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not, in any manner, serve as an endorsement of these Web sites.

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

e (logo) 	IBM
LoadLeveler	MQSeries
Netfinity	Redbooks
Redbooks Logo 	RS/6000
S/390	SP
System/390	Wizard

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere., The Power To Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

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

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

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

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Lotus Notes is a registered trademark of Lotus Development Corporation.

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



---

## Appendix H. Related publications

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

---

### H.1 IBM Redbooks

For information on ordering these publications see “How to get IBM Redbooks” on page 321.

- *Introducing Tivoli Application Performance Management*, SG24-5508
- *Server Consolidation on RS/6000*, SG24-5507
- *AIX 5L Differences Guide Version 5.1 Edition*, SG24-5765

---

### H.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at [ibm.com/redbooks](http://ibm.com/redbooks) for information about all the CD-ROMs offered, updates, and formats.

CD-ROM Title	Collection Kit Number
IBM System/390 Redbooks Collection	SK2T-2177
IBM Networking Redbooks Collection	SK2T-6022
IBM Transaction Processing and Data Management Redbooks Collection	SK2T-8038
IBM Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
IBM AS/400 Redbooks Collection	SK2T-2849
IBM Netfinity Hardware and Software Redbooks Collection	SK2T-8046
IBM RS/6000 Redbooks Collection	SK2T-8043
IBM Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

---

### H.3 Other resources

These publications are also relevant as further information sources:

- *Workload Management Surges to Prominence in UNIX Servers*, D. H. Brown Associates, Inc.
- *AIX Performance Toolbox User's Guide V1.2 and V2.1*, SC23-2625

---

## H.4 Referenced Web site

The following Web sites are also relevant as a further information source:

- [http://www.rs6000.ibm.com/doc\\_link/en\\_US/a\\_doc\\_lib/aixgen/](http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixgen/)  
AIX online documentation
- [www.fz\\_juelich.de](http://www.fz_juelich.de)
- [www.fz\\_juelich.de/zam](http://www.fz_juelich.de/zam)
- [www.fz\\_juelich.de/zam/compserv/services/config.html](http://www.fz_juelich.de/zam/compserv/services/config.html)

## How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** [ibm.com/redbooks](http://ibm.com/redbooks)

Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

Send orders by e-mail including information from the IBM Redbooks fax order form to:

	<b>e-mail address</b>
In United States or Canada	pubscan@us.ibm.com
Outside North America	Contact information is in the "How to Order" section at this site: <a href="http://www.elink.ibm.com/pbl/pbl">http://www.elink.ibm.com/pbl/pbl</a>

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site: <a href="http://www.elink.ibm.com/pbl/pbl">http://www.elink.ibm.com/pbl/pbl</a>

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the "How to Order" section at this site: <a href="http://www.elink.ibm.com/pbl/pbl">http://www.elink.ibm.com/pbl/pbl</a>

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

### IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at <http://w3.itso.ibm.com/> and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at <http://w3.ibm.com/> for redbook, residency, and workshop announcements.





## Abbreviations and acronyms

<b>AIX</b>	Advanced Interactive Executive	<b>LPAR</b>	Logical Partitioning
<b>APAR</b>	Authorized Program Analysis Report	<b>LRU</b>	Least Recently Used
<b>API</b>	Application Programming Interface	<b>m</b>	Minimum limit
<b>ARM</b>	Application Response Measurement	<b>MB</b>	Mega Byte
<b>CCMS</b>	Computing Center Management System	<b>NFS</b>	Network File System
<b>CPU</b>	Central Processing Unit	<b>NIC</b>	John von Neumann Institute for Computing
<b>DB</b>	Database	<b>NIS</b>	Network Information Service
<b>DB2</b>	Database 2	<b>NUMA</b>	Non-Uniform Memory Access
<b>DBA</b>	Database Administrator	<b>OLAP</b>	On-line Analytical Processing
<b>DKIO</b>	Disk I/O	<b>OLTP</b>	On-line Transaction Processing
<b>GB</b>	Gigabyte	<b>OS</b>	Operating System
<b>GID</b>	Group Identification	<b>PGID</b>	Process Group Identifier
<b>GL</b>	Peoplesoft General Ledger	<b>PID</b>	Process Identifier
<b>HACMP</b>	High Availability Cluster Multi-Processing	<b>PTX</b>	Performance Toolbox
<b>HM</b>	Hard Maximum limit	<b>rset</b>	Resource set
<b>HTML</b>	Hypertext Markup Language	<b>SAP</b>	Systems, Applications, and Products in Data Processing
<b>IBM</b>	International Business Machines Corporation	<b>SD</b>	Sales and Distribution
<b>ID</b>	Identification	<b>SGA</b>	System Global Area
<b>I/O</b>	Input/Output	<b>SHED_FIFO</b>	First-In, First-Out Scheduling
<b>IPL</b>	Initial Program Load	<b>SHED_RR</b>	Round-Robin Scheduling
<b>ISICC</b>	IBM SAP International Competence Center	<b>SM</b>	Soft Maximum limit
<b>IT</b>	Information Technology	<b>SMIT</b>	System Management Interface Tool
<b>ITSO</b>	International Technical Support Organization	<b>SMP</b>	Symmetric Multi-Processing
<b>KB</b>	Kilo Byte		

<b>SPMI</b>	System Performance Measurement Interface
<b>SSA</b>	Serial Storage Architecture
<b>TAPM</b>	Tivoli Application Performance Management
<b>TCC</b>	SAP Technical Competence Center
<b>TCP</b>	Transmission Control Protocol
<b>TEC</b>	Tivoli Enterprise Console
<b>TDS</b>	Tivoli Decision Support
<b>TOC</b>	Total Cost of Ownership
<b>TPM</b>	Transactions per minute
<b>TTY</b>	Teletype Terminal
<b>UID</b>	User Identification
<b>URAP</b>	Uniform Resource Access Priority
<b>VMM</b>	Virtual Memory Manager
<b>WLM</b>	Workload Manager
<b>WSM</b>	Web-based System Manager
<b>ZAM</b>	Central Institute for Applied Mathematics

## Index

### Symbols

/etc/group 19, 20, 24  
/etc/inittab 110, 112, 189  
/etc/passwd 19, 20, 24  
/etc/security/limits 304  
/etc/wlm 41, 44, 57  
/etc/wlm/.running 52, 195  
/etc/wlm/current 52, 274  
/etc/wlm/current/rules 44  
/etc/wlm/standard 44  
/etc/wlm/templates 57  
/usr/include/sys/wlm.h 261  
/usr/lib/libwlm.a 195  
/usr/sbin/acct/acctcom 28  
/var/adm/pacct 27

### Numerics

32bit 25  
64bit 25

### A

Accounting 27  
    Displaying WLM class information 28  
acctcom 28  
Active mode 98  
Add a class 68  
Add a class to another configuration 78  
Add or remove processes 186  
Adding a class 62, 68, 73  
Adding a rule 83, 87  
Admingroup 19, 177  
Administration 43  
Adminuser 19, 177  
Advanced Menu 172  
    Example 173  
AIX  
    AIX 5L 9  
    V4.3.3 9, 15  
AIX 5L Version 5.0 9  
AIX 5L Version 5.1 10  
AIX CPU allocation 239  
AIX Version 4.3.3 ML 2 9  
AIX Version 4.3.3 ML 8 10  
AIX Workload Manager  
    See *also* WLM

Alternate configurations 67  
Analysis  
    Long-term 160  
API 25, 195  
API routines 261  
Application pathname 24  
Application Programming Interface (API) 195  
Application tag 25, 195, 273  
    Example 196  
    Example program 197  
    Sample program 309  
ARM API 203  
arm\_start 202  
arm\_stop 202  
Authgroup 20, 177  
Authuser 20, 177  
Automatic assignment 22

### B

Backward compatibility 15, 35  
Bar Display 164  
Base line 214  
Binary compatibility 201  
Bufferpools 191

### C

Case Study  
    PeopleSoft 223  
    SAP R/3 238  
Change Configuration 105  
Change/Show Characteristics of a class 71  
Change/Show Characteristics of a rule 86  
Changing a rule 85, 89  
chclass 63  
    Syntax 63  
Check assignment 91  
Class  
    Change/Show Characteristics 71  
Class Accounting 27  
Class assignment rules 23, 50  
    Application pathname 24  
    Application tag 25  
    Default class 26  
    Examples 26  
    Group 24  
    Process type 25

- System class 26
- User 24
- Class attributes 17, 45, 69
  - Admingroup 19
  - Adminuser 19
  - Authgroup 20
  - Authuser 20
  - Inheritance 18, 188
  - Localshm 21
  - localshm 188
  - Resource set (rset) 20
  - Tiers 18
- Class definitions
  - Refinement 54
- Class management 199
- Class name 12, 23
- Class report 137
- Class resource limits 33
- Class resource shares 31
- Class Set 66
- class\_definition 262, 263
- class\_descr 262
- class\_tbl 275
- Classes 12
  - Adding a class 62, 68, 73
  - Class attributes 69
  - Default 26, 109
  - General characteristics of a class 69
  - Listing classes 64
  - Listing the classes 72, 80
  - Removing a class 65, 73, 81
  - Subclasses 14
  - Superclasses 13
  - System 26, 109
  - Update a class 63
  - Updating a class 69, 79
  - Working with classes 62
    - Command line 62
    - SMIT 66
    - WSM 73
  - Working with sets of subclasses 66
- Classes file 45
- Classification process 22
- Command line 43
- Commands
  - acctcom 28
  - chclass 63
  - crash 41
  - cron 56, 109
  - dkstat 37
  - iostat 31, 37
  - kdb 41
  - lsclass 64
  - lsrset 95
  - mkclass 62
  - nice 38
  - ps 16, 121, 160
  - rmclass 65, 200
  - schedtune 39
  - smit wlm 43
  - smitty wlm 43
  - snap 41
  - svmon 112, 135
  - tee 112
  - topas 124
  - vm tune 39, 112
  - wlmassign 182
  - wlmcheck 54, 90
  - wlmcntrl 44, 99, 113
  - wlmset 21, 35, 81
  - wlmstat 15, 99, 112, 117, 201
  - wsm 43, 150
- Compatibility 15, 35
- Configuration steps 108
- Configurations
  - Checking the configuration
    - Command line 90
    - SMIT 91
    - WSM 91
  - Command line 57
  - SMIT 57
  - WSM 58
- Configuring WLM 258
- Copy a configuration 58, 59
- Copy class attributes 77
- CPU 30
- CPU maximum limit
  - AIX Version 4.3.3 35
- CPU maximum limits
  - AIX Version 4.3.3 81
- crash 41
- Create a configuration 58
- Create a new class 73, 105
- Create a new configuration 60
- Create a new rule 84
- Create a subclass 77
- Create assignment rules 54
- Create other configurations 56

- Create superclasses 54
- cron 56, 109
- Crontab 43
- Customer experience
  - Pre-WLM solution 254
  - WLM solution with AIX 5L 257
  - WLM solution with AIX Version 4.3.3-03 255

## D

- Data mining 53
- DB2 UDB 190
  - Agents 191
  - Bufferpools 191
  - Prefetchers 191
  - Process model 191
  - Snapshot monitoring 192
- DB2 UDB and WLM 192
- default stanza 45, 47, 49
- Delete Rule 89
- Design phase 53
- Design your classification 53
- devstrat 39
- Disk device driver interaction 39
- Disk I/O 31, 39, 119
- Displaying WLM class accounting information 28
- DKIO 119
- dkstat 37
- Dump analysis 41
- Dynamic update 41

## E

- Edit a rule 89
- Enter configuration description 58
- Error codes WLM API 268
- exec 22, 25, 195, 197, 273

## F

- Fine tune your configurations 55
- fixed 25
- Flags
  - Global option flags 81
- fork 22, 195, 197, 273
- Functionality 9

## G

- Gather resource utilization data 54
- General characteristics of a class 69

- getpids 190
- Global option flags 81
- Group 24

## H

- HACMP 110
- Hard maximum limit (HM) 49
- Hints and Tips 107

## I

- Inheritance 18, 22, 179, 188
  - AIX Version 4.3.3 ML8 19
  - Automatic assignment 180
  - Child processes 180
  - Group members 181
  - Manual assignment 180
  - Subclass level 18
  - Superclass level 18
- Insert a rule 88
- iostat 31, 37
- ISV Case Studies 223

## J

- Jazizo 160
- Job attributes 11
- John von Neumann Institute for Computing 253

## K

- kdb 41
- Kernel interaction 36

## L

- Limits 29, 33, 76
  - Constraints 34
  - Hard maximum 34
  - Hard maximum limit (HM) 49
  - Maximum 32
  - Minimum 32, 33
  - Minimum limit (m) 49
  - Soft maximum 33
  - Soft maximum limit (SM) 49
- Limits file 49
- Limits versus shares 35, 110
- List all classes 72
- Listing the classes 64, 72, 80
- Listing the rules 86, 89
- LoadLeveler and WLM 113

- LoadLeveler and WLM interaction 115
- localshm 21, 188
- LPAR 2
  - Virtual LPAR 3
- LRU 34, 111
- lsclass 64
  - Syntax 64
- lsrset 95

## M

- Mainframe partitioning 2
- Manual assignment 22, 23, 177
  - AIX Version 4.3.3 188
  - DB2 UDB 190
  - First assignment 178
  - Methods 182
    - Applications 187
    - Command line 183
    - SMIT 184
    - WSM 186
  - Oracle example 188
  - Oracle example script 307
  - Sample Korn shell scripts 307
- maxfree 112
- maxperm 112
- Memory 30
- Memory regulation 111
- Memory resource management 71
- Memory segment classification 21
- minfree 112
- Minimum limit (m) 49
- minperm 112
- mkclass 62
  - Syntax 62
- Modes of operation 98
- Monitoring 40, 152
- Monitoring applications
  - WLM and Tivoli 203

## N

- New Class Wizard 73
- nice 38
- NIS 19, 24
- Non configured WLM startup 112
- NUMA 3

## O

- OLAP 108
- OLTP 53, 108, 224
- Oracle
  - Database instances 177
  - Example script 307
  - Instances example 188
  - Script
    - Configuration file 189
    - Data structure 190
    - Function 190
    - Process 190

## P

- Partitioning
  - Mainframe partitioning 2
  - NUMA 3
  - Physical partitioning 3
  - UNIX partitioning 2
- Passive mode 9, 40, 54, 98
- Performance tools
  - topas
    - CPU Utilization 129
- PeopleSoft
  - Case study 223
  - Case study method 225
  - OLTP benchmarks 224, 226
  - One batch - two OLTP benchmarks 232, 234
  - Two batch - two OLTP benchmarks 235
  - Two batch benchmarks 235
  - WLM configuration 228
- Performance Toolbox (PTX) 152
  - Console 156
  - Jazizo 160
  - Monitoring console 156
  - xmtrend 160, 161
- Performance tools 117
  - Daemon recording and configuration 161
  - ps 121
    - Examples 123
  - svmon 135
    - Class report 137
    - Examples 140, 146
    - Syntax 138, 145
    - Tier report 145
  - topas 124
    - EVENTS/QUEUES 127
    - Examples 132

- FILE/TTY 127
- MEMORY 128
- Network Interfaces 129
- NFS 129
- PAGING 128
- PAGING SPACE 129
- Physical Disks 130
- Processes 131
- WLM Classes 131
- Web-based System Manager (WSM) 150
- wlmmn 160
- wlmpfr 160
- wlmstat 117
  - Examples 120
- xmperf 153
- xmtrend 161
- PGID 186
- Physical memory 30
- Physical partitioning 3
- plock 25
- PPAR 3
- Practical experience 223
- Prefetchers 191
- Process accounting 27
- Process type 25
- Process type attribute 51
- Properties 59, 79, 97
- Property files 44
- ps 16, 121, 160
  - Examples 123
  - Syntax 121, 160

## R

- Reassignment 179
- Refresh Current Configuration 60
- Remove a class 65, 73
- Remove a configuration 58
- Removing a class 65, 73, 81
- Removing a rule 87, 89
- Report Displays 163
  - Bar Display 164
  - Snapshot Display 165
  - Tabulation Display 166
- Report Properties 167
  - Advanced Menu 172
  - Tier/Class Menu 171
  - Times Menu 168
- Resource limits 33

- Resource management 30
- Resource manipulation 150
- Resource set (rset) 20
- Resource sets
  - Add a new resource set 94
  - Command line 95
  - SMIT 97
  - Working with resource sets 92
    - SMIT 93
    - WSM 97
- Resource shares 31
- Resource target 32
- Resource usage monitoring 209
- Resource usage statistics 36
- Resources 29, 150
  - CPU 30, 38
  - Disk I/O 31, 39
  - Physical memory 30, 39
- resvd attribute 83
- rmclass 65, 200
  - Syntax 65
- rset 92
- rset registry 92
- Rules 82, 110
  - Adding a rule 83, 87
  - Change/Show characteristics 86
  - Changing a rule 85, 89
  - Edit a rule 89
  - Insert a rule 88
  - Listing the rules 86, 89
  - Removing a rule 87, 89
  - Working with rules 82
    - Command line 82
    - SMIT 83
    - WSM 87
- Rules file 50, 82

## S

- Sample workload program 297
- SAP R/3
  - Case Study 238
    - Configuring systems of equal size 245
    - Configuring systems of unequal size 248
    - Priority system with several additional systems 248
    - Process distribution recommendation 249
    - Products tested 238
    - Systems consolidation objectives 242

- Systems of equal size and equal priority 243
- Systems of unequal size 245
- Systems of unequal size but equal priority 247
- WLM classes versus OS processes 239
- Central system 242
- Standard benchmark tool 239
- SAP R/3 Case Study 238
- schedtune 39
- Scheduler interaction 38
- Select a configuration 58
- Server consolidation 1, 7, 206
  - Capacity sizing steps 209
  - Contras 207
  - Pros 207
- setgid 24, 25, 113
- setpri 25
- settag.c 197, 309
- Setting up WLM
  - A starting point 108
  - Before you start 108
  - Configuration steps 108
- setuid 24, 25, 113
- Shared memory segments 21
  - AIX Version 4.3.3 81
  - AIX Version 4.3.3 ML 8 21
- Shares 30, 31, 76
- Shares file 46
- Shares versus limits 35, 110
- Show all configurations 58
- Show Configuration Details 59, 80
- Show current focus 66
- Show Processes 151
- Show Subclasses 151
- Sizing 205, 208
  - All applications are mission-critical 216
  - Comparison of approaches 220
  - CPU 214
  - CPU resource usage calculation sample 311
  - Estimate for each application 210
  - Estimate for integrated applications 214
  - Examples 214
  - Memory and disk I/O bandwidth 221
  - Some applications are mission critical 217
- SMIT 43
- smitty
  - chgwlmrs 85
  - crewlmrs 84
- rset 93
- wlmaddclass 68
- wlmassign 184
- wlmchclass 69
- wlmclass\_gal 97
- wlmconfig 57
- wlmlsclass 72
- wlmmanage 102
- wlmrmclass 73
- snap 41
- Snapshot Display 165
- Snapshot Option Panel 172
- Soft maximum limit (SM) 49
- SP systems 110
- SPMI 152
- Start WLM 99
- Start Workload Management 102
- Start Workload Manager 104, 107
- Start/Stop/Update WLM 99
  - Command line 100
  - SMIT 102
  - WSM 103
- Statistics for WLM 154
- Stop WLM 99
- Stop Workload Management 102
- Stop Workload Manager 105, 107
- Subclasses 12, 14
  - Default 15
  - Shared 15
  - Work on a set of 68, 71, 72, 73, 84, 86, 87
  - Working with sets of 66
- Superclass administrator 12
- Superclasses 12, 13
  - Default 13
  - Shared 13
  - System 13
  - Unclassified 14
  - Unmanaged 14
- svmon 112, 135
  - Class report 137
  - Examples 140
  - Syntax 138
  - Reports 136
  - Tier report 145
  - Examples 146
  - Syntax 145
- SWLMTAGINHERITEXEC 274
- SWLMTAGINHERITFORK 274
- System Performance Measurement Interface 152



## T

- Tabulation Display 166
  - TAPM
    - Application instrumentation 202
    - Overview 202
    - Transaction simulation 202
  - TAPM and WLM 203
  - Target 32
  - tee 112
  - Things to do 107
  - Tier regulation 38
  - Tier report 145
  - Tier/Class Menu 171
  - Tiers 16, 109
    - AIX 5L versus AIX V4.3.3 17
  - Times Menu 168
  - Tivoli
    - Integration with WLM 202
  - Tivoli and WLM
    - Monitoring applications 203
  - Tivoli Application Performance Management (TAPM) 202
  - Tivoli Decision Support (TDS) 203
  - Tivoli Enterprise Console (TEC) 203
  - topas 124
    - CPU Utilization 129
    - EVENTS/QUEUES 127
    - Examples 132
    - FILE/TTY 127
    - MEMORY 128
    - Network Interfaces 129
    - NFS 129
    - PAGING 128
    - PAGING SPACE 129
    - Physical Disks 130
    - Processes 131
    - subcommands 126
    - Syntax 124
    - WLM Classes 131
  - Tree-Details 80
- ## U
- Uniform Resource Access Priority (URAP) 37
  - UNIX partitioning 2
  - UNIX system capacity sizing 205
  - Update a class 63
  - Update WLM 99
  - Update Workload Management 102

- Updating a class 63, 69, 79
- URAP 37
- User 24

## V

- Virtual LPAR 3
- VMM 34
- VMM interaction 39
- vmtune 39, 112

## W

- Web-based System Manager (WSM) 43, 150
- WLM
  - Accounting 27, 201, 293
    - Displaying WLM class information 28
  - Active mode 98
  - Administration 43
  - Alternative configurations 56
  - API routines 261
  - Application Programming Interface (API) 195
  - Basic elements 10
  - Before you start 108
  - Class accounting 27
  - Classes 12
  - Classification 201
  - Commands 43
  - Configuration 53
  - Configuration steps 53, 108
  - Configurations
    - Command line 57
    - SMIT 57
    - WSM 58
  - CPU allocation 240
  - Customer experience 252
  - Functionality 9
  - General recommendations 306
  - Global option flags 81
  - Integration with Tivoli products 202
  - Library 269
  - Management 200
  - Memory regulation 111
  - Modes of operation 98
  - Monitoring 40, 152
  - Operation 98
  - Overhead 40
  - Overview 10
  - Passive mode 98
  - Performance Tools 117

- Purpose of 6
- Resource management 30
- Resource Usage Statistics 36
- Resources 29
- Sizing recommendations 205
- Start/Stop/Update 99
- Statistics 201
- Status 104
- Things to do 107
- Tier regulation 38
- Turn on 55
- WLM accounting 293
- WLM and DB2 UDB 192
- WLM and LoadLeveler 113
- WLM and LoadLeveler interaction 115
- WLM and TAPM 203
- WLM and Tivoli
  - Monitoring applications 203
- WLM API
  - Accounting 201, 293
  - Application tag 273
  - Class management 274
  - Classification 290
  - Constants 261
  - Data structures 261
  - Function prototypes 261
  - Functions error codes 268
  - Management 280
  - Statistics 285
- WLM in a research environment 252
- WLM Report Browser 163
- wlm\_args 261
- wlm\_assign 187, 264, 283
  - Parameter 284
    - args 284
- wlm\_bio\_class\_info\_t 266
- wlm\_bio\_dev\_info\_t 266
- wlm\_change\_class 199, 277
  - Parameter 278
    - wlmargs 278
- wlm\_check 201, 290
  - Parameter 290
    - config 290
- wlm\_class2key 29, 195, 201, 293
  - Parameter
    - args 294
- wlm\_classify 201, 291
  - Parameters 292
    - attributes 292
    - class 292
    - config 292
    - len 292
- WLM\_Console Menu 162
- wlm\_create\_class 199, 276
  - Parameter 277
    - wlmargs 277
- wlm\_delete\_class 199, 200, 279
  - Parameter 279
    - wlmargs 279
- wlm\_endkey 195, 201, 296
  - Parameter
    - args 296
- wlm\_get\_bio\_stat 201, 203
- wlm\_get\_bio\_stats 288
  - Parameters 288
    - array 289
    - class 289
    - count 289
    - dev 289
    - flags 288
- wlm\_get\_info 201, 203, 285
  - Parameters 286
    - count 287
    - info 287
    - wlmargs 286
- wlm\_info 264
- wlm\_init\_class\_definition 276, 278
  - Parameter 272
    - wlmargs 272
- wlm\_initialize 197, 268
  - Parameter 272
    - flags 272
- wlm\_initkey 195, 201, 293
  - Parameter 293
    - args 293
- wlm\_key2class 29, 195, 201, 294
  - Parameter
    - args 295
- wlm\_load 200, 281
  - Parameter 282
    - wlmargs 282
- WLM\_MUTE 261
- wlm\_read\_classes 199, 274
  - Parameters 275
    - class\_tbl 275
    - nclass 276
    - wlmargs 275
- wlm\_set 200, 280

- Parameter 280
  - flags 280
- wlm\_set\_tag 177, 195, 197, 273
  - Parameters 273
    - flags 273
    - tag 273
- WLM\_VERSION 261, 273
- wlmassign 182
  - Syntax 182
- wlmcheck 54, 90
  - Syntax 90
- wlmcntrl 44, 99, 113
  - Syntax 100
- wlmmon 160
  - Differences to wlmperf 160
  - Prerequisite filesets 175
  - Report Displays 163
    - Bar Display 164
    - Snapshot Display 165
    - Tabulation Display 166
  - Report Properties 167
    - Advanced Menu 172
    - Tier/Class Menu 171
    - Times Menu 168
  - Snapshot Option Panel 172
  - WLM Report Browser 163
  - WLM\_Console Menu 162
- <\$.nopage *See also* wlmmon 163
- wlmperf 160
  - Differences to wlmmon 160
  - Prerequisite filesets 175
- wlmsched 39
- wlmset 21, 35, 81
  - Syntax 81
- wlmstat 15, 99, 112, 117, 201
  - Examples 120
  - Internal parameters 118
  - Syntax 117
- Work on alternate configurations 67
- Working with WLM configurations 56
- Workload management 5
  - Need for 1
- Workloads 7
- WSM 43
  - Advanced configuration tool 75
  - Class Assignment Rules 88, 89
  - Configurations/Classes 58, 74, 79, 80, 88, 91, 97, 106, 151, 186
  - Overview and Tasks 73, 103

- Resources 150
- wsm 150

## X

- xmperf 153
- xmservd 159
- xmtrend 160, 161
  - Daemon recording and configuration 161
  - Syntax 161



## IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at [ibm.com/redbooks](http://ibm.com/redbooks)
- Fax this form to: USA International Access Code + 1 845 432 8264
- Send your comments in an Internet note to [redbook@us.ibm.com](mailto:redbook@us.ibm.com)

<b>Document Number</b>	SG24-5977-01
<b>Redbook Title</b>	AIX 5L Workload Manager (WLM)
<b>Review</b>	
<b>What other subjects would you like to see IBM Redbooks address?</b>	
<b>Please rate your overall satisfaction:</b>	<input type="radio"/> Very Good <input type="radio"/> Good <input type="radio"/> Average <input type="radio"/> Poor
<b>Please identify yourself as belonging to one of the following groups:</b>	<input type="radio"/> Customer <input type="radio"/> Business Partner <input type="radio"/> Solution Developer <input type="radio"/> IBM, Lotus or Tivoli Employee <input type="radio"/> None of the above
<b>Your email address:</b> The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	<input type="checkbox"/> Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
<b>Questions about IBM's privacy policy?</b>	The following link explains how we protect your personal information. <a href="http://ibm.com/privacy/yourprivacy/">ibm.com/privacy/yourprivacy/</a>





Redbooks

# AIX 5L Workload Manager (WLM)











# AIX 5L Workload Manager (WLM)



**Redbooks**

**Effectively manage your system resources**

Effectively managing system resources in growing UNIX server environments has become a critical task. Each workload on the server must be assured the appropriate amount of system resources without penalizing mission-critical applications. AIX 5L Workload Manager provides a great set of tools and functionalities to efficiently accomplish this.

**Learn how to deploy the new functionality**

**Control the resource consumption of individual applications**

This IBM redbook exploits the entire functionality of AIX 5L Workload Manager, which has been enhanced in many ways since its introduction with AIX V 4.3.3. The new manual assignment feature allows you to separate, for example, multiple instances of a database. An API allows you to perform all the WLM administration and configuration tasks from a program. A step-by-step guide is provided for planning and configuring AIX WLM through file editing, AIX commands, the System Management Interface Tool (SMIT), or Web-based System Manager (WSM). Real-life examples have been added to demonstrate the impact and benefits of using AIX Workload Manager.

This IBM redbook is the ultimate guide for system architects, technical support specialists, and system administrators to planning, implementing, and administering a Workload Manager solution in a consolidated server environment.



**DISKETTE INCLUDED**

**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](http://ibm.com/redbooks)

SG24-5977-01

ISBN 0738422436