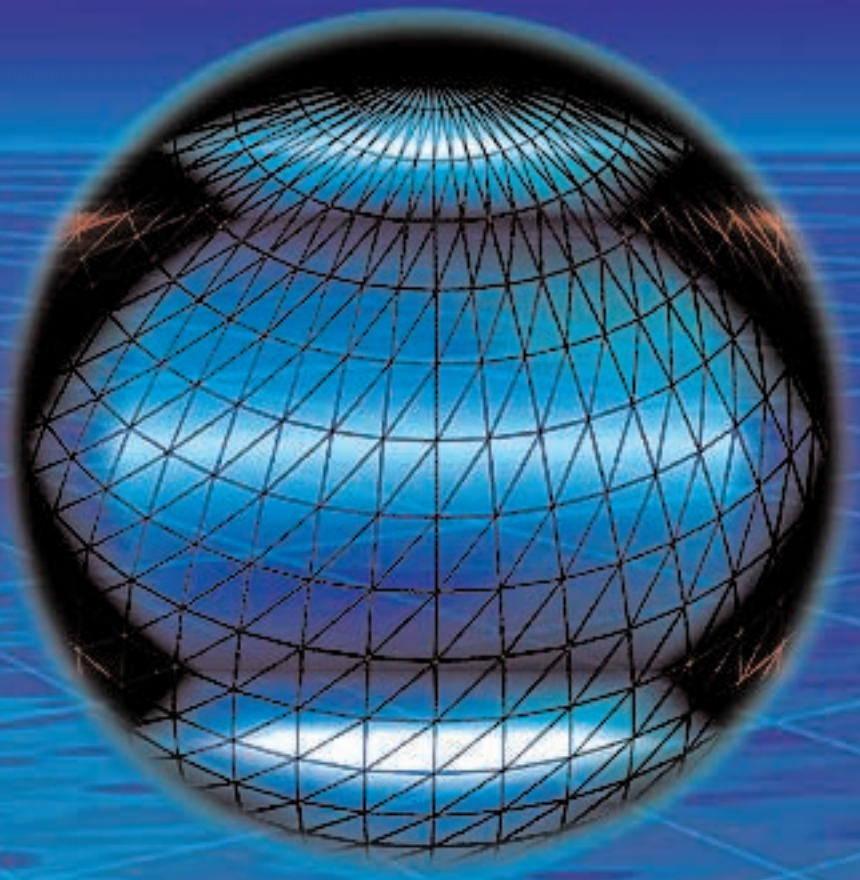
IBM'S MAGAZINE FOR AIX DEVELOPERS DECEMBER 1999

AIXpert

FOCUS ON WEBSPHERE APPLICATION SERVER

Building and Managing Your Web Applications



Inside This Issue...

- The WebSphere Application Server
- AIX Performance Tuning: Memory
- Distributed Computing with Java RMI

IBM

AIX Performance Tuning: Focus on Memory

By Wayne Huang, Lee Cheng, and Matt Accapadi

This article, the second in a series of AIX performance tuning discussions, focuses on memory usage. It summarizes information that will help you understand the characteristics and dynamics of AIX Virtual Memory Management system including discussions on real memory, virtual memory, free list, persistent segments, working segments, page stealing, and other aspects of virtual memory. It also details how to monitor and tune the usage of system memory through standard and advanced AIX® performance tuning tools. Three real-world case studies are included to illustrate some important knowledge gained from our experience.

AIX implements a memory management component to extend the address capability beyond the amount of real memory (RAM) that the physical hardware can support. Because the address space does not correspond one-to-one with real memory, it is called *virtual memory*. Virtual memory can be thought of as an extended set of memory addresses. Programs use these virtual addresses rather than real addresses to fetch or store instructions, data, and file content. When programs are executed, the virtual addresses are converted into real memory addresses in order for the system hardware to carry out the necessary instructions correctly.

To facilitate copying between virtual and real memory, AIX divides virtual memory into pages of 4 KB each. The real memory is made into a group of 4 KB units called *page frames*. Each page can reside in real memory or be stored on disk until it is needed. When the first address in a virtual memory page on disk is first touched, the CPU hardware cannot address such a virtual page directly and a page fault occurs.

During a page fault, AIX obtains one frame from the pool of free real-memory frames called the *free list* (this may cause page stealing, which is described later in this article), copies the page content from disk to the frame, translates the virtual addresses into real addresses, and continues the execution. The process of address translation/address mapping, and the copying of page content from disk to main memory happens when the data is needed. This represents the AIX implementation of *demand paging*.

Free List and Page Stealing

The free list is the collection of available real-memory page frames maintained by the Virtual Memory Manager (VMM) to accommodate page faults. Unless plenty of fresh unused memory is available, a VMM component called a *page stealer* must occasionally "steal" some real-memory frames from pages


Wayne Huang


Lee Cheng


Matt Accapadi

currently owned by running processes, or steal some stale memory pages no longer associated with active processes to replenish the free list.

A frequently asked question is why the free list is so short. Does the short free list indicate a memory problem? The answer is no; a short free list does not mean a problem. The VMM, by default, does not intend to maintain a long free list. If a large number of pages are stolen each time, the amount of active real pages for running system and application processes will be reduced accordingly. Therefore, the "working set" of the running processes now has less real memory, which could trigger subsequent paging requests for the programs that are running to get back some necessary pages.

The free list size is tunable in AIX. Two thresholds control the action of the page stealer. The system default for the high threshold is computed as two frames per megabyte of memory with a cap set at 128 pages. The low threshold is set at eight frames below the high threshold. These two thresholds are controlled by `maxfree` and `minfree` in the `vmtune`.

Persistent and Working Segments

Virtual memory pages on the physical disks are classified into three different types, based on the location of the actual backing stores. Pages are collected into a virtual segment that can be one of the three types:

- Local persistent segments
- Working segments
- Client persistent segments (remote files and executables, CD-ROM file systems)

Pages of a persistent segment have permanent disk storage locations. Data files and executable program text are mapped to persistent segments. When a page has to be freed up because of page stealing or other system actions, the VMM will determine if this page has been changed (dirty or not). If the page is dirty, it will be written back to the disk location; if not, there is no other step required, and its frame is simply reassigned to the free list.

The pages of working segments are transitory—they exist only during their use by a process—and have no permanent disk locations. The stack and data region of a process and the kernel text segment are mapped to working segments. When a program is executing, the working segment pages, such as the process heap, must have disk locations to occupy when they cannot be kept in real memory. This disk space is the well-known paging space or swap space.

In the client persistent segment, the client segments are mapped to remote files, remote executables, and CD-ROM file systems.

Figure 1 shows the three types of memory segments, the backing store locations, and the relationship with the real memory frames.

Computational vs. File Memory

Frequently in UNIX, memory pages are divided into computational and file pages. Computational pages, or computational memory, are pages that belong to working storage segments or program text segments. The working segment and a portion of the persistent segment (the process code) in Figure 1 belong to the computational memory.

File pages, or file memory, consist of the remaining pages. Those pages are often occupied by file content.

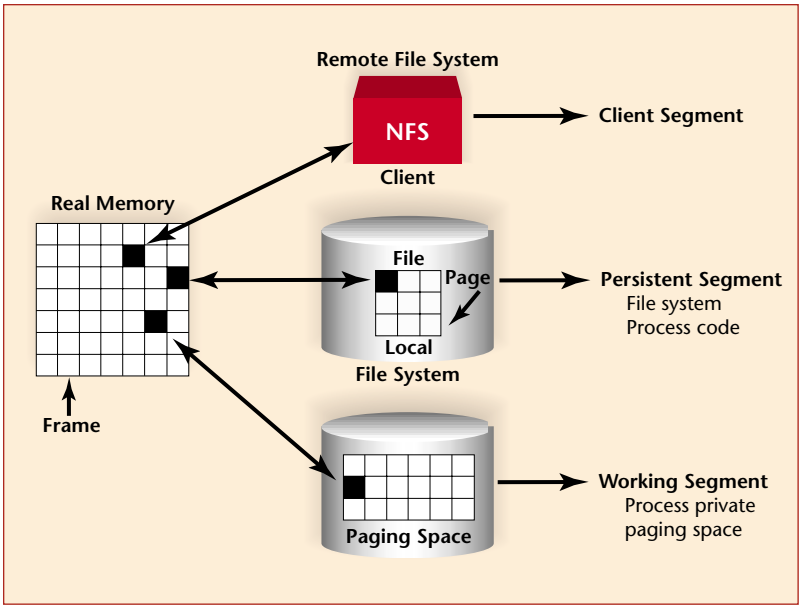


Figure 1. VMM memory segments

Figure 2 shows the relationship between computational and file memory. It also shows the three segment types within the same diagram for comparison.

How Page Stealer Works

The page stealer is invoked when the number of free-page frames reaches the low threshold (the `minfree` value). The page stealer tries to replenish the free list until it reaches the high threshold—the `maxfree` value. Both `minfree` and `maxfree` are tunable using `vmtune`.

The page stealer employs a technique called *clock algorithm*, which points a clock hand to cycle through each entry in the Page Frame Table (PFT) searching for pages to steal. The LRU policy is used to determine the page-stealing targets. To implement an effective, yet efficient LRU policy, the stealer takes advantage of a *referenced* and a *modified* bit for each page as an indication of what pages have been recently used or referenced, and if those pages have been modified (dirty).

The page stealer first examines the referenced bit for each page. If the page is unreferenced and stealable (that is, not pinned and meets other page-stealing criteria), it is immediately stolen and placed on the free list. Pages with the referenced bit on will not be stolen, but their reference bit is reset, effectively "aging" the reference so that the page can be stolen the next time the page stealer visits.

When a page is about to be stolen, the page stealer also examines the modified bit. The modify flag indicates whether the page was changed since it was brought into memory. If the modified flag is set, a page-out operation is made before recovering the page frame.

The PFT also maintains the segment type for each page entry in the table. The page stealer uses this information to balance paging activities for computational and file pages in the real memory. For example, the sequential reading of a large data file into memory should not cause the program text pages to be paged out. The reason is clear that a program code is likely to be executed many, many times in a controlled loop, while the file data tend to be relatively inert.

Figure 3 shows how the page stealer operates against entries in a page frame table. Note that after the first pass of the clock hand, those pages without a referenced bit set are stolen and placed into the free list.

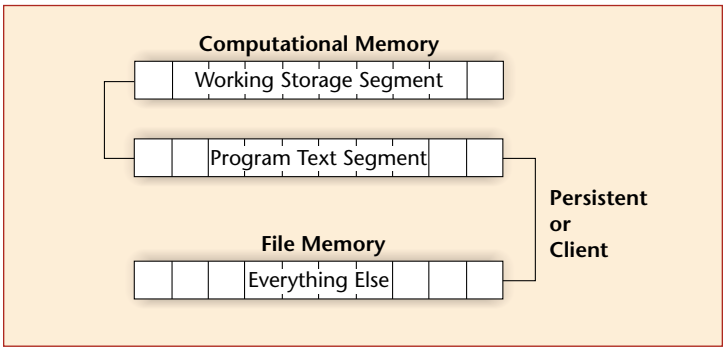


Figure 2. Computational vs. file memory

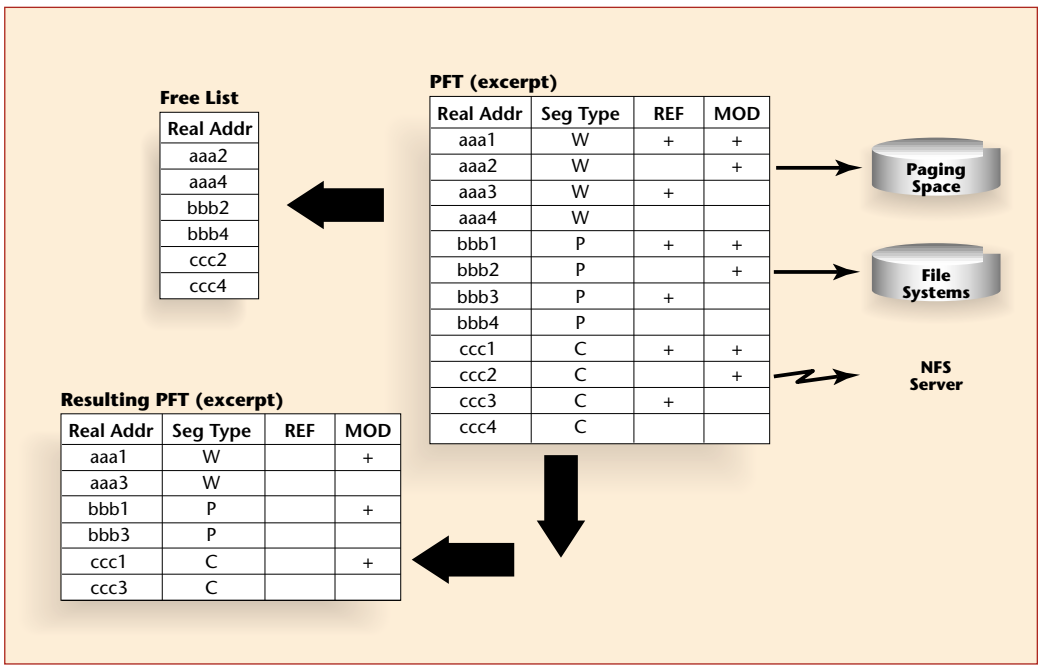


Figure 3. Page stealing operation

Memory Load Control[1]

Some circumstances, such as insufficient real memory with a large number of processes running concurrently, may offer no good candidates for page stealing. A page that is just paged out to make room for one process may have to be paged in immediately, just to allow the original process to continue. This can cause continuous paging in and out, and the system is thrashing.

AIX uses a memory load-control algorithm to detect if the system is about to thrash. It can then decide to suspend some active processes and delay the initiation of new processes for a period of time. Five parameters control this memory load-control algorithm and behavior. The five parameters are tunable using the `schedtune` program.

Figure 4 describes the memory load control parameters and their meaning.

Many newer and larger systems today have enough real memory to easily support day-to-day applications. To reduce the possible system overhead in memory load control for large machines, the Virtual Memory Manager memory load control is turned off automatically when the system is installed, if the system has RAM of 128 MB or higher.

Paging Space Allocation Algorithms: Early, Late, and Deferred

Other virtual memory aspects associated with the page allocation timing requirement include:

- Early paging space allocation
- Late paging space allocation
- Deferred paging space allocation

The detailed description for the three page-allocation modes is beyond the intended level of this article. See the AIX manuals for more in-depth information.

[1] The memory load control described in this section is rarely activated for newer systems with large amounts of real memory. It is included here, however, to explain the important concept of system and process thrashing, the system actions to prevent thrashing, and the fine-grain memory load control you can do should memory resource become a problem.

Parameters	Description
`-h`	Memory overcommitment threshold for the system (default is 6, or 0 if more than 128 MB RAM). If po/fr is greater than 1/h[1] during a one-second period, the system tries to suspend processes to avoid possible thrashing. The higher the `h` value, the earlier the alarm for thrashing. If h = 0, memory load control is off.
`-w`	Wait time before reactivating suspended processes (default is 1 second). This time is the amount of time that the po/fr ratio must be below 1/h before the suspended processes can be reactivated. With a default of 1 second, the system will try aggressively to reactivate suspended processes. A large `w` value can cause poor performance because of delayed recovery for suspended processes. The default 1 second is a good value to stay on.
`-p`	Memory overcommitment threshold for processes (default of 4). If rp/pf is greater than 1/p (which indicates a particular process is thrashing), this process becomes a candidate for suspension. The higher the `p` value, the earlier the alarm for process thrashing. If p = 0, the process suspension is prohibited.
`-m`	Minimum degree of multiprogramming (default of 2). This parameter provides the bare minimum low limit for concurrent processes (not threads). When memory load control is at work, it cannot suspend any more processes once the `m` value is reached. ◆ A large `m` for a small (memory) system results in an ineffective memory load control, since it does not allow enough processes to be suspended even when it is necessary. ◆ A small `m` for a large system can cause overly aggressive process suspension.
`-e`	Exempt period for reactivated processes (default of 2 seconds). If a process is just reactivated, it will be given a chance to run for at least the amount of time specified by the `e` parameter before it can be considered a candidate for suspension again. The default—2 seconds—is appropriate for most cases.

[1] *Note:*
`po`: Number of page writes in the last second
`fr`: Number of page steals in the last second
`Rp`: Number of repages in the last second, for a particular process
`Pf`: Number of page faults in the last second, for a particular process

Figure 4. Memory load control parameters

More Physical Memory

You now understand how virtual memory supplements real memory, the different types of virtual memory segments, and the page replacement algorithm used by the page stealer. Next, we will discuss how to determine whether a system has a memory bottleneck. We will illustrate some commands and their options for monitoring memory usage. For detailed information, see the AIX publication CD-ROM or visit the AIX Web site at http://www.rs6000.ibm.com/doc_link/en_US/a_doc_lib/aixgen.

Common Memory Commands

The first category of commands are those most commonly used for memory monitoring. They are also available on other UNIX platforms.

vmstat

This command was described in our CPU usage tuning article.[2] The versatile `vmstat` command provides an overview of CPU, disk, and memory activity in a one-line-per-report format. It is usually invoked with a specified interval and count. The interval is the number of seconds between two measurements and the count is the number of intervals for `vmstat` to run. Another common usage is to run `vmstat` specifying only the interval without the count parameter to generate reports continuously until the command is canceled.

Figure 5 shows a sample output generated on an AIX 4.3.2 system by running `vmstat 2 14`.

The command `vmstat 2 14` generates a one-line system activity summary report every two seconds for a total of 14 reports. Except for the avm and fre columns, the first report contains per-second average statistics since the startup of the system. Subsequent reports contain statistics collected during the interval after the preceding report.

The detailed description of the columns conveys useful information about memory usage.

Memory: Information related to both real and virtual memory. Although neither column (avm or fre) provides any hint of performance problems caused by memory shortage, it is important to understand what they mean.

- **avm:** Active virtual memory pages that have been accessed. For releases prior to AIX 4.3.2, this value corresponds to the used paging space shown using the `lsps -s` command, which displays the paging space information in bytes. With AIX 4.3.2 the default page space policy is "deferred." Therefore, the avm value is usually much higher than the amount of paging space used. If the avm value indicates that virtual memory accessed exceeds the total amount of real memory,

kthr		memory		page						faults			cpu			
r	b	avm	fre	re	pi	po	fr	sr	cy	in	sy	cs	us	sy	id	wa
0	0	15185	218	0	0	0	0	0	0	116	197	20	0	0	99	0
0	1	15186	216	0	0	0	0	0	0	233	166	44	0	0	99	0
0	1	15186	216	0	0	0	0	0	0	228	40	42	0	0	99	0
0	1	15186	216	0	0	0	0	0	0	237	53	45	0	0	99	0
0	1	15213	142	0	0	0	0	0	0	230	81	44	0	1	97	2
0	3	15495	106	0	4	7	266	2515	0	360	2430	276	3	15	5	78
0	2	15501	79	0	43	47	161	307	0	346	1016	325	4	18	6	71
0	1	16619	75	0	27	69	710	2818	0	358	731	307	23	13	21	43
0	2	16640	72	0	1	44	181	740	0	378	303	294	3	6	16	75
1	1	16641	58	0	0	10	79	1756	0	379	812	192	28	2	59	10
0	1	16719	88	0	7	32	123	2529	0	373	1425	315	1	6	14	78
0	2	16799	77	0	6	25	115	211	0	414	622	262	6	4	19	72
0	2	16812	72	0	38	47	118	375	0	569	1234	432	3	4	1	92
0	3	16840	78	0	20	22	153	774	0	446	560	314	2	3	0	94

Figure 5. The vmstat command

[2]Cheng, Lee, Huang, Wayne, and Accapadi, Matt. "AIX Performance Tuning: CPU Usage." *AIXpert* (July 1999).

then paging could occur. If the free paging space is below the preset system threshold, the operating system stops creating new processes, or even begins killing existing processes. Although more paging space does not mean better performance, it is critical to have enough paging space configured on a system.

- **fre:** The number of free memory frames; that is, the length of the free list maintained by the page stealer.

  The fre number shows the size of the free list. If a fre number is constantly and significantly higher than the `maxfree` value (which can be displayed or changed using `vmtune` command), then the system is probably not experiencing any memory constraint. If a consistently small fre value oscillates between `minfree` and `maxfree`, there is no need to be alarmed. As described earlier, the page stealer only does the minimum amount of work to keep the free list above `minfree` value. It is not unusual for the free list size to remain small.

Page: Information about page faults and paging activity, which is averaged over the interval and given in pages per second.

- **re:** Page reclaims per second. If a page fault occurs and the target page is currently on the free list yet to be reassigned, this is considered a reclaim without incurring any new I/O request. Nevertheless, page reclaims were not implemented in the AIX Version 4 multiprocessor kernel because of the low performance gain in return for the high implementation cost.
- **pi:** Pages paged in from paging space only.
- **po:** Pages paged out to paging space only.
- **fr:** Pages freed by page replacement algorithm.
- **sr:** Pages scanned by page replacement algorithm.
- **cy:** The number of complete cycles (per second) or scans (per second) of the entire page frame table. Since `vmstat` only shows integers, this field is usually zero.

The two columns to examine for a memory bottleneck are pi and po. These represent the number of paging space page-ins and page-outs respectively, and are good indicators of the paging space activity. If these two values are consistently nonzero, a memory bottleneck likely exists.

During the sampling period with high pi and po numbers (shown in Figure 5), both the block queue thread count and I/O wait time also increased. The block-queue thread count increases because threads are blocked waiting for pages to be paged in. The I/O wait time increases because the processor is idle while threads are still in the block queue. However, a zero I/O wait time does not necessarily indicate that no memory bottleneck exists. If the workload has enough processes to keep the processors busy, the increase of I/O wait may not necessarily be observed.

The ratio of fr/sr also can indicate the level of memory overcommitment. For example, a fr/sr ratio of 1:4 means the page stealer, on average, must examine four pages to put one on the free list. The higher the ratio, the busier the memory. High pi and po numbers accompanied with a high fr/sr ratio suggest a significant memory bottleneck.

vmstat -s

The pi and po columns do not include paging activities from the file space. Using the `vmstat` command with `-s` option will produce a report that includes the total I/O activity initiated by the virtual memory manager since the system was booted.

The sample output in Figure 6 is generated on an AIX 4.3.2 system by running `vmstat -s`. The following four fields in the report are helpful for monitoring memory usage:

- **page-ins:** Total number of pages read in by the VMM, from both paging space and file space.
- **page-outs:** Total number of pages written out by the VMM, both to file space and paging space.

- **paging space page-ins:** VMM initiated page-ins from paging space only.

- **paging space page-outs:** VMM initiated page-outs to paging space only.

The value for paging space page-ins can be subtracted from the value for total page-ins to produce the number of pages read from persistent storage (file). Likewise, the value for paging space page-outs can be subtracted from the value for total page-outs to get the number of persistent pages (files) written.

Since it is not possible to specify an interval when using the `-s` option and the page count reported is the cumulated count since system initialization, it is common to run this command by executing `vmstat -s` at the beginning and end of the interval. The difference between like fields of successive reports can then be calculated to find the paging activity during the interval. A simple script `vmstatit` in the *AIX Performance Tuning Guide* (SC23-2365) does this semi-automatically. Figure 7 shows the `vmstatit` script with a slight variation.

```
5456310 Total address translation faults
  20219 Page-ins
 556220 Page-outs
   1224 Paging space page-ins
  10455 Paging space page-outs
      0 Total reclaims
1662478 Zero filled pages faults
   9343 Executable filled pages faults
 296930 Pages examined by clock
      4 Revolutions of the clock hand
  31665 Pages freed by the clock
 302398 Backtracks
      0 Lock misses
    552 Free frame waits
      0 Extend XPT waits
  13932 Pending I/O waits
 262470 Start I/Os
 262470 Iodones
65517396 CPU context switches
380144205 Device interrupts
      0 Software interrupts
      0 Traps
643367907 Syscalls
```

Figure 6. vmstat -s output

When the version of the `vmstatit` shell script in Figure 7 is run as `vmstatit "cp /unix /tmp/unix"` and `/unix` has not been read recently, the result is shown in Figure 8. The total page-ins during the measuring period is the delta between 71700 and 72364—664. The paging space page-in is the delta between 18264 and 18268—4. The majority of pages read in during this interval is clearly the file pages of `/unix`.

ps command

To see how much memory each process is using, as shown in Figure 9, the `ps au` command can provide a snapshot of the memory and CPU usage. The SZ column shows the page space used in 1 KB units and RSS shows the real memory consumed

```
vmstat -s >temp.file
time $1  >results
vmstat -s >>temp.file
echo "total page-ins:"               >>results
grep "^[ 0-9]*page-ins"    temp.file >>results
echo "Paging space page-ins:"        >>results
grep "pagi.*ins"           temp.file >>results
echo "Total page-outs:"              >>results
grep "^[ 0-9]*page-outs"   temp.file >>results
echo "Paging space page-outs:"       >>results
grep "pagi.*outs"          temp.file >>results
```

Figure 7. vmstatit script

```
real    0m9.55s
user    0m0.00s
sys     0m0.37s

Total page-ins:          71700 page-ins
                         72364 page-ins

Paging space page-ins:   18264 paging space page-ins
                         18268 paging space page-ins

Total page-outs:         706428 page-outs
                         707299 page-outs

Paging space page-outs:  25859 paging space page-outs
                         25892 paging space page-outs
```

Figure 8. Output of vmstatit

by the process in 1 KB units. The %MEM indicates the percentage of real memory used by the process. It is calculated by dividing the RSS value by the size of the real memory of the machine in 1 KB units, multiplied by 100, then rounded to the nearest full percentage point.

Unfortunately, both %MEM and RSS tend to exaggerate the real memory usage of the process because they do not consider the sharing of program code with other processes. Furthermore, the majority of processes in the system show a %MEM of 0.0 because of rounding.

Figure 9 shows output of the `ps au` command.

AIX-specific Commands

The second category of commands are those implemented specifically for AIX. The command that falls into this category is `svmon`.

svmon

This command enables more in-depth understanding of memory usage. The various options provide different types of memory usage statistics, such as system-wide global statistics, individual segments of each process, detailed information of a specific segment, and other information. The command requires that you have a good understanding of AIX virtual memory architecture to comprehend the report fully. For more information on memory usage, see the `svmon` command in AIX manuals or the *AIX Performance Tuning Guide*.

Reducing the Amount of Paging

The following occurrences during the monitoring process may indicate a memory bottleneck in the system:

1. The pi and po columns in the `vmstat` report are consistently nonzero.
2. The number of pages read or written as calculated from `vmstat -s` command are high.
3. The I/O wait time from the `vmstat` report is constantly high.

Adding more real memory can definitely help the paging situation. If the majority of paging is from/to paging space, creating separate spaces on multiple volumes can help. If the majority of paging is from/to the file space, tuning the I/O performance of the file system involved may help. In addition, tuning the VMM to reduce the amount of paging can also be beneficial.

vmtune

This command modifies the behavior of VMM. Since incorrect usage of the command can cause a profound impact to system performance, only root can execute it.

The bos.adt.samples fileset must be installed to access this command. This fileset is located in `/usr/samples/kernel`. Figure 10 shows the output of running `/usr/samples/kernel/vmtune` without any options.

USER	PID	%CPU	%MEM	SZ	RSS	TTY	STAT	STIME	TIME	COMMAND
root	14814	5.8	4.0	10092	10208	pts/1	A	16:02:24	0:03	/usr/netscape/nav
root	16090	0.0	2.0	6272	6060	pts/1	A	16:02:28	0:00	(dns helper)
root	15260	0.0	0.0	264	296	pts/2	A	16:02:50	0:00	ps au
root	14494	0.0	0.0	488	516	pts/2	A	16:00:11	0:00	-ksh
root	14058	0.0	0.0	488	364	pts/1	A	16:00:10	0:00	-ksh
root	16610	0.0	0.0	484	428	pts/1	A	16:02:22	0:00	ksh /bin/netscape
root	4162	0.0	0.0	500	216	pts/0	A	Aug 25	0:00	/bin/ksh
root	4940	0.0	0.0	352	24	lft0	A	Aug 25	0:00	/usr/sbin/getty /

Figure 9. Output of the ps au command

The first part of the output displays the current VMM settings. Three fields of high interest include:

- **minperm.** Shows the number of file pages below which the page stealer will steal either file or computational pages, regardless of repaging rate. The bottom half of the report displays this number converted to the percentage of real memory. This sample output has `minperm` set to be 19.7% of the real memory.

- **maxperm.** Shows the number of file pages above which the page stealer will steal only the file pages. This number converted to percentage of real memory is displayed at the bottom half of the report. This sample output has `maxperm` set to be 78.7% of the real memory.

- **LRUbucket.** Represents a new parameter added beginning with AIX 4.3. It shows the size (in 4 KB pages) of the bucket into which the real memory is divided. This parameter is required to ensure good VMM performance in the latest RS/6000® server machines that have large quantities of memory. In the sample output, the default is set to 131072 frames, which is equivalent to 512 MB. This is the number of page frames that will be examined at one time by the page stealer looking for pages to replace. The page stealer scans for a free frame as needed. During the first pass, reference bits of pages are reset. If it does not find enough free frames and it conducts a second scan, the frames that still have the reference bit off will be put on the free list. Systems with very large memory may have too many frames to scan before the second pass; therefore, the memory is divided into buckets. The page stealer will scan the frames in the bucket and start the second pass as necessary, then move to the next bucket.

Two fields in the lower part of the report are worth mentioning: the number of file pages in the real memory and its percentage of total real memory (`numperm`). In this sample output, `numperm` is shown as 3.8%.

If pi and po values in the `vmstat` report are consistently significant and the `numperm` value is near `maxperm`, it may be beneficial to lower the `maxperm` value to, say 60%, using `vmtune -P60`. With a lower `maxperm` value, the page stealer will likely steal from file pages and give working pages more chances to remain in real memory, thereby reducing paging activities from/to paging space.

```
vmtune: current values

-p              -P              -r              -R              -f              -F
minperm         maxperm         minpgahead      maxpgahead      minfree         maxfree
12897           51590           2               8               120             128

-N              -W              -M              -w              -k              -c
pd_npages       maxrandwrt      maxpin          npswarn         npskill         numclust
524288          0               52410           2048            512             1


-b              -B              -u              -l                              -s
numfsbufs       hd_pbuf_cnt     lvm_bufcnt      lrubucket       defps           sync_release_ilock
93              112             9               131072          1               0

number of valid memory pages=65512          maxperm=78.7% of real memory
maximum pinable=80.0% of real memory        minperm=19.7% of real memory
number of file memory pages=2508            numperm=3.8% of real memory
```

Figure 10. vmtune output

If pi and po values in the vmstat report are insignificant and the vmstat -s report shows the majority of paging is from/to file space, then it may be beneficial to raise the maxperm value to give file pages preference. A typical example of this situation would be a system that runs as an NFS server.

If you modify vmtune parameters, be aware that changes made by the vmtune command remain effective until the system is rebooted. If a permanent change in VMM parameters is necessary, an appropriate vmtune command should be put into inittab.

Case Studies

This section describes several real-world examples.

Case 1

Customers often report performance problems, then discover the cause was excessive paging in/paging out of working storage pages. This example describes a customer that had a system with 2 GB of RAM, yet performance was sluggish. The output of vmstat was similar to Figure 11.

The pi and po columns show much paging in/paging out and from/to the paging space devices. Threads are being blocked because the number of threads in the VMM wait queue (the b column) is nonzero.

To determine the amount of virtual memory being accessed on the system, multiply the value in the avm column by 4 KB, or 4096. Multiplying 301121 by 4096 equals approximately 1.2 GB of virtual memory. This system had 2 GB of RAM, but the virtual memory requirement is only 1.2 GB. So why is the system paging so much? The next step is to look at the vmtune command, shown in Figure 12.

The numperm value in Figure 12 is approximately 58%; that is, about 58% of the total RAM is filled up with file pages. The maxperm value is about 80%, which is the default, and minperm is the default value of 20%.

When page replacement must occur, the VMM compares numperm to maxperm. If numperm is less than maxperm but greater than minperm, then the VMM looks at repaging rates. In this particular case, the repaging rate for files must have been higher than for computational pages, so the VMM was only replacing computational pages—these pages are paged out to paging space and brought back in later when accessed again.

Depending on the type of files being accessed, it may be more beneficial to replace file pages instead of computational pages, even if the repaging rate for files is higher. For example, if file pages are just being accessed as read-only, then replacing one of these file pages from RAM simply means discarding that page—no page-out is necessary. The next time that page is accessed, it can be obtained from the file system.

If a computational page, such as working storage, needs to be replaced, it would most likely need to be paged out to paging space (one I/O), then brought back into memory when accessed again (another I/O). This requires twice as much work as replacing a file page.

kthr		memory		page						faults			cpu			
r	b	avm	fre	re	pi	po	fr	sr	cy	in	sy	cs	us	sy	id	wa
5	10	284390	0	0	45	164	2505	4502	0	2324	9544	5009	12	52	0	36
7	10	291387	122	0	11	216	1987	9415	0	2108	6175	5828	17	49	0	33
5	3	301157	145	0	37	86	2699	5886	0	2293	8078	3332	14	47	0	39
3	2	301489	1	0	17	51	2370	8674	0	2155	6666	3289	6	32	0	62
6	5	301589	371	0	12	134	2333	7061	0	2228	7083	4714	14	42	0	43
2	2	301188	132	0	20	65	2530	5105	0	2259	7119	3157	5	33	0	62
5	3	301725	131	0	13	104	3097	12212	0	2494	8926	4509	8	50	0	42
4	3	301121	184	0	7	106	3118	6872	0	2530	9531	3733	9	49	0	42

Figure 11. Output of vmstat

```
# /usr/samples/kernel/vmtune
vmtune: current values
-p              -P              -r              -R              -f              -F
minperm         maxperm         minpgahead      maxpgahead      minfree         maxfree
104619          418476          2               64              120             184

-N              -W              -M              -w              -k              -c
pd_npages       maxrandwrt      maxpin          npswarn         npskill         numclust
524288          0               419296          24576           6144            16

-b              -B              -u
numfsbufs       hd_pbuf_cnt     lvm_bufcnt
93              1632            9

number of valid memory pages=524120          maxperm=79.8% of real memory
maximum pinable=80.0% of real memory         minperm=20.0% of real memory
number of file memory pages=303316           numperm=57.9% of real memory
```

Figure 12. The vmtume command

In this case, we want to tune the system so that the VMM will only replace file pages—not working storage pages. Therefore, we decrease the `maxperm` value so that it is much lower than the `numperm` value. Since `numperm` is 58%, we lower `maxperm` to 40%. Therefore, when page replacement must occur, only file pages are replaced (because `numperm` is higher than `maxperm`).

Figure 13 shows the `vmtune` output from the customer system a few days after the customer implemented our tuning suggestion. Figure 14 shows the `vmstat` data.

Page replacement is still occurring because the fr/sr columns are nonzero. There is no more paging to/from the paging devices because only file pages are now being replaced. Also, the run queue and the wait/blocked queue now are reporting smaller numbers than previously.

```
vmtune: current values
-p              -P              -r              -R              -f              -F
minperm         maxperm         minpgahead      maxpgahead      minfree         maxfree
78618           209648          2               64              120             184

-N              -W              -M              -w              -k              -c
pd_npages       maxrandwrt      maxpin          npswarn         npskill         numclust
524288          0               419296          24576           6144            16

-b              -B              -u
numfsbufs       hd_pbuf_cnt     lvm_bufcnt
93              1632            9

number of valid memory pages=524120          maxperm=40.0% of real memory
maximum pinable=80.0% of real memory         minperm=15.0% of real memory
number of file memory pages=277551           numperm=53.0% of real memory
```

Figure 13. The vmtune output after the maxperm and minperm change

kthr		memory		page						faults			cpu			
r	b	avm	fre	re	pi	po	fr	sr	cy	in	sy	cs	us	sy	id	wa
1	1	301000	176	0	0	0	1033	1110	0	1863	3877	2198	3	17	0	81
1	1	301006	213	0	0	0	1350	1957	0	1876	4774	2103	4	18	0	78
1	1	301008	185	0	0	0	1340	2824	0	1878	4034	2090	4	16	0	81
1	1	301084	195	0	0	0	1014	1089	0	1868	3850	2048	3	16	0	81
1	1	301060	152	0	0	0	1311	2282	0	1877	4708	2180	4	17	0	79
1	0	301084	169	0	0	0	1339	1993	0	1869	3945	2063	3	15	0	82
1	1	301086	153	0	0	0	1353	2853	0	1879	4071	2082	4	16	0	80

Figure 14. vmtune output after maxperm and minperm changes

Case 2

In this second example, another customer reported a performance problem. The output of `vmstat` looked like Figure 15.

The fre column shows that the number of free frames was consistently low—even 0 at times. The pi and po values also are constantly nonzero, which indicates that continuous paging of working storage pages to/from the paging space devices is occurring. The number of threads on the VMM wait queue (this is the second column b) indicates that nearly 200 threads on average were waiting for a page.

Note the I/O wait values under the wa column. Is the system paging because of too many file pages in RAM or simply because there is not enough RAM to satisfy the working storage requirements of the workload the customer was running? To begin, we checked the amount of RAM on the system. The configuration data showed that this system had 1GB of RAM. If we take a typical value from the `vmstat` avm column and multiply it by 4 KB, the answer is about 1.5 GB. So the working storage requirements for this system are at least 1.5 GB, but it has only 1 GB of RAM.

Figure 16 shows the output of the `vmtune` command, which shows how many file pages are in memory.

Note that the customer had already tuned `maxperm` and `minperm` much lower than the default values; `numperm` is at 19%, so the system was paging even though not much RAM was used by files.

Since `numperm` is higher than `maxperm`, only file pages should have been stolen. Since many working storage pages are being replaced or stolen, the file pages in memory were being constantly accessed (their reference bits must have been set each time the page replacement algorithm examined the page frame table). In this case, we conclude that the system does not have enough RAM for this particular workload. The customer should either reduce the workload or add more memory. The result: The customer added another 1 GB of RAM for a total to 2 GB. The `vmstat` data from that point showed the results in Figure 17.

Although page replacement is still occurring and threads are on the VMM wait queue, the numbers look much better. Instead of nearly 200 threads on the wait queue, there are about six. The number of

kthr		memory		page						faults			cpu			
r	b	avm	fre	re	pi	po	fr	sr	cy	in	sy	cs	us	sy	id	wa
1	195	381837	41	0	77	159	241	2692	0	748	8054	573	12	24	0	65
1	173	382670	146	0	30	172	199	2455	0	864	3719	326	12	17	0	71
1	167	385751	0	0	16	172	361	3445	0	584	1323	1263	2	27	0	71
1	188	390009	0	0	9	321	463	1834	0	626	3563	1416	4	29	0	67
2	204	393852	40	0	10	308	426	1642	0	599	2149	1080	3	27	0	71
0	185	395703	136	0	25	152	239	2397	0	633	3906	543	5	20	18	57

Figure 15. vmstat output before tuning

```
vmtune: current values

-p              -P              -r              -R              -f              -F
minperm         maxperm         minpgahead      maxpgahead      minfree         maxfree
7864            20970           2               8               120             128

-N              -W              -M              -w              -k              -c
pd_npages       maxrandwrt      maxpin          npswarn         npskill         numclust
524288          0               209708          12288           3072            1

-b              -B              -u              -l                              -s
numfsbufs       hd_pbuf_cnt     lvm_bufcnt      lrubucket       defps           sync_release_ilock
93              336             9               131072          1               0

number of valid memory pages=262135     maxperm=8.0% of real memory
maximum pinable=80.0% of real memory    minperm=3.0% of real memory
number of file memory pages=49849       numperm=19.0% of real memory
```

Figure 16. vmtune output showing numperm and other data

```
kthr     memory              page                                    faults              cpu
-------  ------------------  --------------------------------------  ------------------  -----------
r  b     avm     fre         re  pi  po  fr   sr     cy              in   sy     cs      us sy id wa
1  11    477133  1031        0   7   0   452  21634  0               1303 12183  1391    13 18 0  70
1   6    476025  1078        0   6   0    57   1492  0               1244 14202  1560    11 16 0  72
1   5    476278   689        0   3   0   155   5008  0               1149 12914  1466    15 17 0  68
1   7    475288   506        0   2   0    41   9132  0               1220 13419  1415    14 17 0  69
0   7    475816   553        0   4   0   215  11348  0               1262 11937  1363     9 15 0  76
1   6    475127   627        0   6   0   160   3437  0               1274 12612  1539    14 15 0  71
2   6    476628   553        0   2   0   341  21787  0               1283 16573  1580    19 18 0  63
1   6    474701  1824        0   4   0    79   1914  0               1180 14965  1505    16 18 0  67
```

Figure 17. vmstat output after adding 1 GB of memory

page-ins is very small and there are no page-outs to paging space.

By multiplying the avm value by 4 KB, the value is close to 2 GB. This leaves little for file pages, so it is still not enough RAM, yet much better.

It is still necessary to tune `maxperm` and `minperm` to be low numbers; this ensures that only file pages are replaced. In some cases, memory is so constrained that `maxperm` and `minperm` are set to 1%.

Case 3

Another customer reported a paging problem. Although the data is no longer available, we can describe our observations.

The `vmstat` data was collected for a 24-hour period. During this time, the avm column continued to increase and never decreased in value. The system was paging heavily because the free list had reached the `minfree` value.

It seemed odd that the working storage requirements continued to grow rather than staying steady; this implies that a memory leak may exist. The next step was to look at `svmon` data. The performance data collected was obtained by running a tool called `perfpmr.sh`. This tool collects `svmon` data at the beginning and at the end of the collection period.

This data showed that a process whose count of memory pages in-use and in

paging space had increased substantially from the beginning of the collection period to the end. By viewing the segment, we could determine if it was a kernel, shared library, or application problem.

In this case, the private segment had grown. In fact, the segment had grown by about 80 MB in 24 hours. The growth of a private segment showing this type of growth indicates an application problem. Fortunately, the customer had the source code for this application since it was an in-house application. By adding some instrumentation to their code that allocates memory, they found a memory leak in their application. Once this was fixed, the system no longer paged constantly and the system performance improved.

***Lee Cheng**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Ms. Cheng currently works as a senior consultant for RS/6000 and AIX software vendors. She provides support in the areas of application benchmarks, performance tuning, application porting, and internationalization. Before joining the RS/6000 ISV Technical Support group, she was a developer for compilers and the AIX system management component. She holds an MS in Computer Science from the University of Kentucky.*

***Wayne Huang**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Huang is a senior consultant for RS/6000 and AIX systems with a focus on banking, finance, and securities industries. He provides AIX support to ISVs in the areas of application design, problem determination, system performance tuning, and application benchmarks. Mr. Huang holds a BS in Physics from National Taiwan University and an MS in Computer Science from the University of Texas at Austin.*

***Matt Accapadi,** IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Accapadi is a software engineer responsible for performance of AIX and Oracle on AIX, and works with other vendors to improve their application performance on AIX. He is also responsible for solving performance issues on customer systems, tuning benchmarks for optimal performance, and teaching courses on AIX performance tuning. Mr. Accapadi has a BS in Computer Science from Texas A&M University.*

Disk Mirroring Performance Trade-offs

By Bret R. Olszewski and David B. Whitworth

This article describes concepts of disk mirroring, a technique for achieving higher availability storage. Attributes of mirroring on AIX are described as well as how they affect disk I/O performance. Measurements are included on several mirrored configurations to help you determine the best configuration for your environment.

Disk mirroring is a mechanism to increase disk subsystem reliability through redundancy. Distilled to its simplest form, mirroring keeps exact copies of data on two or more disks simultaneously. Of course, in reality, there are a number of different mirroring options available in AIX 4.3.3, including, for the first time, striping and mirroring together. Knowing mirroring options and configuring them based on application behavior and throughput requirements is the most important factor in optimizing disk subsystem performance, and therefore, overall system performance.

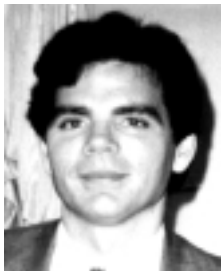
Mirroring is a property of logical volumes. Thus, mirroring works for disk I/O to file systems and raw logical volumes. AIX provides the following options when creating raw logical volumes for mirroring:

Number of copies of each logical partition: This is the number of unique copies of the data. This value may be one (not mirrored), two, or three. First, double mirroring protects against a single point of failure. Triple mirroring, while not frequently used, can be advantageous in a number of situations. Triple mirroring actually allows the I/O subsystem to tolerate multiple disk failures. Second, for online backups, triple mirroring protects against a single point of failure. Finally, triple mirroring can provide better performance for parallel scheduling policies if the data is extremely read-intensive.

Mirror write consistency (MWC): This mechanism ensures consistent writes when mirroring is enabled. For each volume group containing a mirrored logical volume (LV) with MWC enabled, a log is maintained. This log tracks the last 62 Logical Track Groups (LTGs) that were written. (An LTG is simply a 128 KB area of an LV). The log is kept in memory and on a single 512-byte block near the outer edge of each disk in the volume group.

The size of this log limits the maximum number of outstanding MWC log updates to 62 per volume group. The basic algorithm is that—before a mirrored write is initiated—the LVM checks the log for the LTGs being written. It they are already in the log, the write continues. If they are not in the log, the log is updated in memory and written to each disk containing part of the LV. Once the log writes are complete, the data is written. If a system failure occurs, then during reboot the LVM will copy all of the LTGs recorded

Bret R. Olszewski



David B. Whitworth

in the log from one copy to the others. This does not guarantee that the data is correct—a partial write could be applied, but it does guarantee that at system recovery the mirrors will be consistent. Perhaps most importantly, when MWC is enabled, some writes have the MWC log write precede them, causing disk seeks. Thus, turning off MWC usually benefits write performance. If MWC is disabled, it is possible to synchronize mirrors using the `syncvg` command.

Allocate each logical partition copy on a separate physical volume: This is a yes or no question which should normally be answered yes. Mirroring is effective at protecting against disk failures if there are two or more copies on physically different devices. Thus, should one disk fail completely, the others can be accessed for the data.

SCHEDULING POLICY for reading/writing: The standard policies are parallel, parallel/sequential, parallel/round robin, and sequential.

- The **parallel policy** balances reads between the disks. On each read, the system checks whether the primary is busy. If it is not busy, the read is initiated on the primary. If the primary is busy, the system checks the secondary. If it is not busy, the read is initiated on the secondary. If the secondary is busy, the read is initiated on the copy with the least number of outstanding I/Os. Writes are initiated concurrently.

- The **parallel/sequential policy** always initiates reads on the primary copy. Writes are initiated concurrently.

- The **parallel/round robin policy** is similar to the parallel policy except that instead of always checking the primary copy first, it "round robins" between the copies. This results in equal utilization for reads even when there is never more than one I/O outstanding at a time. Writes are initiated concurrently.

- The **sequential policy** results in all reads being issued to the primary copy. Writes happen serially, first to the primary disk; only when that is completed is the second write initiated to the secondary disk.

Enable WRITE VERIFY: This is another yes or no question. Normally, Write Verify is not enabled. When it is enabled, each write to a disk is followed with a read. The data returned by the read is compared with the original data to ensure that the disk image is correct. Because each write becomes two operations, performance is typically much lower when write verify is enabled. Write verify is typically not very useful when mirroring is in use. For example, in a double mirror case, if one write of the mirror pair succeeds, the application is not notified that the other write failed (although an error log entry is created). With mirroring, write verify would only be useful if both mirror writes failed.

Stripe Size: The number of contiguous bytes of data per disk. Striping data allows higher possible disk bandwidth by using multiple disks in parallel for reads or writes.

Performance Tests

We conducted the following performance tests for this article. *Note:* A full stripe is defined here as the LVM stripe size times the number of disks in the striped LV.

rseq: Sequentially reads 32 KB blocks from the beginning of the Journaled File System (JFS) data file.

rseq_1stripe: Sequentially reads full-stripe-size blocks from the beginning of the JFS data file. The `vmtune` command is used to increase `maxpgahead` to equal the full stripe size.

rseq_raw: Sequentially reads 128 KB blocks from the beginning of the raw data logical volume. This test uses much less CPU than the JFS read test because it uses much larger blocks and the more efficient raw I/O path. Typically, direct I/O on JFS has similar performance to I/O on raw logical volumes.

rseq_raw_1stripe: Sequentially reads full-stripe-size blocks from the beginning of the raw data logical volume.

rran_64mb: Randomly reads 16 MB from the first 64 MB of the data file. The 64 MB size was chosen to represent typical localized random I/O, such as paging space

size or a small database. It reads 4096 4 KB pages. This test also runs on JFS files.

rraw_16p_maxmb: A multiprocess test that creates 16 processes, each reading 1/16 of the 4096 blocks on a raw logical volume. This test exercises the disk device driver's reordering of I/Os for seek optimization. It also shows the benefit of command tagged queuing, which queues multiple commands to the disk. Queuing multiple requests to the disk optimizes performance by reducing latency for initiating disk requests.

wseq: Sequentially writes 32 KB blocks to the beginning of the JFS data file. The test overwrites the existing data file. The 32 KB writes are coalesced into larger physical I/Os by the file system. To ensure all data is on disk, the file is flushed with fsync at the end of the test.

wseq_raw: Sequentially writes 128 KB blocks to the beginning of the raw data logical volume. This test uses much less CPU than the JFS write test because it uses much larger blocks and the more efficient raw I/O path.

wseq_raw_1stripe: Sequentially writes full-stripe-size blocks to the beginning of the raw data logical volume.

wseq_raw_16stripe: Sequentially writes (16*full stripe size) blocks to the beginning of the raw data logical volume.

wran_64mb: Randomly writes 16 MB to the first 64 MB of the data file. It writes 4096 blocks, 4 KB each in size. Like the wseq test, the file is flushed with fsync at the end of the test.

The results reported from these tests were collected on a four processor RS/6000 Model H70 system. The system has four PCI buses, limiting the effective disk I/O bandwidth to approximately 360 MB/second with the supported configuration of four Advanced Serial RAID adapters. The disks were IBM 9.1 GB Serial Storage Architecture (SSA) drives with 7200 rpm rotational speed. Tests with mirrors measure the throughput observed by the test program. This does not measure the bandwidth to the disks, which may be two or three times higher for writes.

As always, measurements reflected in this article are created under laboratory conditions using artificial test cases. As such, they are no guarantee of performance achievable in production environments. Consider also that run-to-run variance exists for these tests, so small performance differences should not be considered significant.

Case 1: Results from Basic Tests

Case 1 consists of a set of simple mirroring performance tests on AIX 4.3.3. The results, shown in Figure 1, depend on the application of AIX 4.3.3 APAR IY05136.

The first test in the figure was run with a single disk with no mirroring. As expected, sequential reads and writes have much

	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6
Configuration						
Mirror	No	Yes	Yes	Yes	Yes	Yes
Disks	1	2	2	2	2	3
Policy	N/A	P	P	P	S	P
MWC	N/A	On	Off	On	On	On
Write-verify	Off	Off	Off	On	Off	Off
Results (MB/second)						
rseq	14.51	14.48	14.50	14.52	14.51	14.49
rran_64mb	0.54	0.54	0.54	0.54	0.54	0.54
rraw_16p_maxmb	0.66	1.27	1.27	1.27	0.65	1.69
wseq	13.71	12.54	13.87	3.71	12.55	11.83
wran_64mb	0.97	0.83	0.96	0.26	0.78	0.80

Figure 1. Mirroring measurements

higher performance than random I/O, as disk seeks are not necessary.

The second test, parallel mirrors with two disks, shows write performance is reduced over the non-mirrored case. This is a result of mirror write consistency.

The third test is like the second, except it turns off mirror-write consistency. The performance impact of mirror-write consistency is only observed on writes. Write performance becomes equivalent to the non-mirrored tests.

The fourth test modifies the second by enabling write verification. Write verification greatly reduces write performance because for every write, the data is subsequently read and compared to ensure the on-disk data is correct. Read performance is unaffected by write verification.

The fifth test modifies the second by using sequential mirrors. This case has the best read performance because the reads always go to the primary mirror, giving the same behavior as the unmirrored test. Write performance is again limited by MWC.

The sixth test modifies the second by having data mirrored on three disks. In this test, sequential read performance is the same as the other cases. Sequential write performance, using parallel mirrors, is slightly reduced from the two mirror case, as rotational latency for three disks becomes a factor.

Case 2: Scaling Up Mirroring Performance

The next case explores the performance characteristics of mirroring on larger disk configurations. First, consider that each MWC cache is one per volume group. This limits each volume group to a maximum of 62 outstanding MWC updates.

Figure 2 shows how increasing the number of volume groups while holding the number of disks constant can increase write performance. Notice that read performance, as expected, is unaffected. But write performance, both for sequential and random cases, is much improved with multiple volume groups. The throttling of outstanding writes within a volume group can seriously degrade overall disk performance.

	Test 1	Test 2
Configuration		
Mirror	Yes	Yes
Disks	32	32
SSA adapters	4	4
Volume Groups	1	16
Logical Volumes	16	16
Parallel/sequential	P	P
MWC	Yes	Yes
Write-verify	Off	Off
Results (MB/second)		
rseq	220.75	219.67
rran_64mb	8.63	8.62
wseq	77.96	93.99
wran_64mb	3.94	11.83

Figure 2: Effect on performance of the number of volume groups and MWC

Case 3: Striping and Mirroring

AIX 4.3.3 introduces function that allows striping and mirroring together on the same logical volume. This provides a convenient mechanism for high-performance redundant storage. Another method, striped RAID-5 devices, was described in the December 1998 issue of *AIXpert*.[1] Figure 3 shows results obtained by combining striping and mirroring. The measurements indicate that read and file system write performance of striping and mirroring is approximately equal to the unmirrored case, assuming you have twice as many disks.

The `rseq_1stripe` benchmark shows the benefit of larger I/Os and using `vmtune -R` to increase `maxpgahead` when reading JFS files on striped LVs. The `rseq_1stripe` performance still lags the `rseq_raw_1stripe` performance because of a CPU bottleneck. Even though these tests were run on a four-CPU machine, only one CPU was used to perform I/O,

[1]Olszewski, Bret R., and Whitworth, David B. "AIX Performance." *AIXpert* (December 1998).

which limits the throughput that one CPU can sustain.

File system writes benefit from caching in the file system, which allows considerable overlap of writes to disk with the program initiating the writes. The raw write performance suffers. Since it is synchronous, both writes must complete before control is returned to the initiating program. Performing larger writes increases raw write throughput. Also, MWC affects the performance of this case.

Case 4: CPU Utilization Impacts of Mirroring

Increased CPU utilization is observed when comparing mirrored storage to non-mirrored storage for data that is write intensive. The CPU consumption for reads should be similar, whether or not the disks are mirrored. Alternately, the CPU consumption for writes should be higher when disks are mirrored, because AIX must do more physical writes, each of which consumes CPU cycles in the logical volume manager and device drivers.

When we compared the disk throughput and CPU utilization between configurations, such as mirrored and unmirrored, it was difficult to get directly comparable results. Thus, we normalized the CPU utilization to the throughput to create the metric of cycles per byte. This is a metric of efficiency: lower cycles per byte mean lower CPU utilization for disk I/O.

Using this methodology, we observed that the CPU utilization for disk I/Os, comparing unmirrored to two mirrors, shows an increase of approximately 15 percent in overhead for sequential writes. The comparison for random writes is an increase in overhead of approximately 38 percent. These numbers represent a worst-case scenario. If the application is a database in which only 20 percent of the execution time is performing I/O with a high degree of reads to writes, the actual overhead of mirroring may be virtually immeasurable.

Conclusion

When comparing mirrored and non-mirrored configurations, mirroring at least doubles the number of disks to contain the same data. Mirroring scheduling policies, such as parallel and parallel/round-robin, can allow performance on read-intensive mirrored configurations to be equivalent to non-mirrored ones. Typically, performance on write-intensive mirrored configurations is less than non-mirrored, unless more disks are used.

Striping and mirroring allow redundant storage for very high-performance access. Normally, the CPU overhead of mirroring is not significant in application environments.

	Test 1	Test 2
Configuration		
Mirror	No	Yes
Disks	16	32
Adapters	4	4
Logical Volumes	1	1
Volume Groups parallel/sequential	N/A	Yes
MWC	N/A	Yes
Write-verify	Off	Off
Results (MB/second)		
rseq	26.27	27.26
rseq_1stripe	89.03	84.31
rseq_raw_1stripe	195.39	194.54
wseq	65.18	61.90
wseq_raw_1stripe	74.02	39.21
wseq_raw_16stripe	183.96	90.13

Figure 3. Striping and mirroring performance

Bret R. Olszewski, *IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Olszewski is a senior programmer working on MP performance. He joined IBM in 1989 and has worked on various aspects of AIX performance. He has a BS in Computer Science from the University of Minnesota.*

David B. Whitworth*, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Whitworth is a staff programmer in AIX Performance. He has a BA in Computer Science from the University of Texas.*

License Use Management in AIX 4.3: A Customer Perspective

By Veronica L. DeHart

License Use Management (LUM) is a mechanism for licensing and managing licensed-enabled products. To design a good license network, it is important to first consider the network's composition. LUM must then be installed and configured for the required setup on the server and the client. Once LUM is installed and configured correctly, the addition of products becomes a single command or a point-and-click with a Basic License Tool (i4blt). License Use Management is the single source for license administration in AIX 4.3.

License Use Management (LUM) from IBM is the strategic license management tool for all use-based, priced AIX® Licensed Program Products (LPPs), and is the single product for controlling licenses on AIX 4.3. The intent of LUM is to enable customers to audit software usage on their systems and to provide a mechanism for complying with their IBM contracts.

Through the use of encrypted keys, iFOR® technology (iFOR/LS or LUM) can monitor the type and number of licenses used by a stand-alone machine or by machines in a network. This enables a company to ensure its compliance with the terms of the licensing agreement. All new and future releases of products will require LUM configuration and enrollment of license files. Figure 1 shows IBM products that currently utilize LUM.

This article is intended to introduce customers to LUM characteristics on AIX. For a comprehensive review or for installation of LUM, see the documentation references noted at the end of the article.

Product licensing has evolved over the years. At one time, Apollo® Computer's NetLS was the standard for managing product licenses. Then iFOR/LS, from Gradient® and Hewlett-Packard®, enhanced the NetLS code. In 1997, ISOGON Corp. acquired iFOR/LS and renamed it License Power/iFOR.

Products Using License Use Management

- C for AIX Version 4.4
- XL Fortran Version 5.x (optional)
- XL Fortran Version 6.x (optional)
- VisualAge® C++ on AIX Version 4.x
- C/C++ Version 3.6.x (bundled with VisualAge C++)
- VisualAge for Java™ on AIX Version 2.x

Figure 1. License Use Management products

Veronica L. DeHart

IBM introduced its implementation of iFOR technology—LUM—as an alternative to iFOR/LS in AIX 4.2. Now, LUM is the only tool available in AIX 4.3; however, iFOR/LS commands are still compatible. Generally, the commands are symbolic links to equivalent LUM commands or shell scripts that format arguments before calling a LUM command. LUM commands can be reviewed in the *LUM Commands Reference* (see "License Use Management Resources" at the end of this article).

License Use Management offers a number of features that enable customers to manage their products effectively. LUM is currently supported on a variety of platforms, including OS/2® WARP™ Versions 3 and 4, Windows® 95, Windows NT™ 3.51 and 4.0, HP/UX 10.10 and 10.20, Sun Solaris® 2.5.1, and, of course, AIX. In heterogeneous environments, this means an AIX license server can provide licenses for all of these platforms. If the AIX platform is using licenses compatible with NetLS, it can even use one of these platforms as the license server. Platforms will continue to be added to the LUM support structure.

The License Use Management interface can be a Graphical User Interface (GUI), System Management Interface Tool (SMIT), or command line. This article will show command-line usage. See the LUM Runtime guide (See "License Use Management Resources" at end of this article) for the graphical equivalents and for SMIT usage. If you plan to install the License Use Runtime GUI filesets, you must first install IPF/X Runtime Support (`ipfx.rte`).

The Network License Configuration Plan

The first and most important step is to plan your network license environment. This involves attention to details that are often not given enough time or thought, especially when they involve a large network. The LUM Runtime guide includes a chapter, "Planning Your Network Licensing Environment," to assist in assessing these considerations. As indicated in the LUM documentation, when designing your network, you will need to determine:

- The number of network license servers to be set up
- Which machine(s) will be the network license server(s)
- How product licenses will be distributed among the network license servers
- Which clients will have access to which servers
- How clients will locate the servers
- Whether all servers will serve all clients, or whether several groups of servers and clients will be established
- Which machine will function as the central registry (if required)

License Use Management is the single source for license administration in AIX 4.3.

Once these questions are answered, review the LUM Runtime guide for the steps necessary to set up the LUM configuration. Next, install License Use Runtime (if needed) and additional filesets for other options, such as backward compatibility. Finally, configure and test the LUM configuration for the servers and the clients.

Direct Binding Example

The following example illustrates a node-lock network server and client using direct binding. This is a typical setup for a small, stable network environment with one or two servers. Direct binding is used because it is simple to set up. With direct binding, list the network license servers and the central registry. During configuration of servers and clients, specify the network addresses of all the servers on the list. They are stored on every server and every client in a local text file called the configuration file.

To handle products with network licenses, you must configure at least one network license server and configure each client as a network license client. If you use

products with customer-managed or reservable licenses (a reservable license is a network license reserved for the exclusive use of a user, a group, or a node), you must also configure one network license server as the central registry (that is, C/C++ 3.6.x). A single machine can perform more than one role in your licensing environment.

Configure the LUM Server

LUM must be configured before the product licenses can be accessed for compiler usage. A typical error message when using the compiler without the LUM configuration would read:

```
No licenses available. Contact your
program supplier to add additional
users.
```

The steps below describe the standard configuration procedure, which can be modified as appropriate. Chapter 8 in the LUM Runtime guide provides details. LUM can be configured from the GUI by invoking `i4cfg`. As root user, follow these steps.

1. Change directory to access commands: `#cd /var/ifor`
2. Execute the configuration tool: `#i4cfg -script`. If you see `i4cfg Version 4.0 AIX`, then you may find that some of the questions appear in a different order. Respond per question as shown to correctly configure LUM.
   If you see the message `Configuration cannot take place until all licensing daemons are stopped`, respond with **y** to stop the daemons so configuration can continue.
   If you get a message `default ini file not found, configuration ends` or any message mentioning the `ini` file, then delete the file `/var/ifor/i4ls.ini` (if it exists) and run `/usr/opt/ifor/ls/os/aix/bin/i4cnvini` to create a new `ini` file. Make sure this file has read permissions for everyone.
3. Select **4** (Central Registry [and/or Network and/or Nodelock] License Server) on the first panel.
4. Answer **Y** to Do you want this system to be a Network License Server, too?
5. Answer **Y** to Do you want this system to be a Nodelock License Server?
6. Answer **N** to Do you want to disable remote administration of this Network License Server?
7. Answer **N** to Do you want to disable remote administration of this Nodelock License Server?
8. Select **2** (Direct Binding only) as the mechanism to locate a license server.
9. Answer **N** to Do you want to change the Network License Server IP port number?
10. Answer **N** to Do you want to change the Central Registry License Server IP port number?
11. Answer **N** to Do you want to change the Nodelock License Server IP port number?
12. Select **1** "Default" as the desired server(s) logging level.
13. Press **Enter** to accept the default path for the default log file(s).
14. Answer **N** to Do you want to modify the list of remote Nodelock and/or Network License Servers? (*Note:* You can input remote administration servers or clients here if needed and continue with questions.)
15. Answer **Y** to Do you want the License Server(s) to automatically start on this system at boot time?
16. Answer **Y** to Do you want to continue?
17. Answer **Y** to Do you want the License Server(s) to start now?

Once these steps are complete, the LUM server should be configured. You should receive a message stating that the configuration is complete. To verify that the server is configured correctly, use the `i4tv tool`, which is discussed later in this article. The next step is to configure other machines in the network.

Configure a LUM Network Client

For clients to access the licenses, they also must be configured with LUM. Follow the next steps to do this for a direct binding network client. If you choose to use name-space binding as the binding type, review LUM documentation on its usage and LUM configuration parameters. This procedure can also be done using the GUI invoked from `i4cfg`. Follow these steps as root user.

1. Change to the LUM directory:
   `# cd /var/ifor`
2. Invoke the configuration tool command:
   `# i4cfg -script`.
3. Select **1** (Network License Client) on the first panel.
4. Select **2** (Direct Binding only) as the mechanism to locate a license server.
5. Select **3** (Create a new list) to the direct binding list menu. Enter the hostname, without the domain, of the system for which you are configuring LUM when prompted for the server network names. Enter the Central Registry License server name without the domain.
6. Answer **N** to Do you want to change the default IP port number [1515]?
7. Answer **N** to Do you want to change the default IP port number [10999]?
8. Answer **Y** to Do you want to continue?

You should see a note that the configuration file has been successfully updated and completed. Verify configuration with the `i4tv tool`, which is discussed in the next section.

LUM Tools

LUM has many options and commands available for use. It is important to know the commands or become familiar with the GUI to obtain the desired license administration information. A discussion of basic commands to verify the network setup follows. For more options and command usage, refer to the *LUM Commands Reference*.

LUM has a tool that allows you to check the configuration of your server and client: `i4tv`. It should be run on both the server and the client to ensure accuracy of input information. Figure 2 shows a sample command output.

LUM provides a great tool for checking statistics and status: Basic License Tool (`i4blt`). This tool can be used with its graphic interface or its command line to determine the number of licenses being used at that moment or over a given period. To see all available options, use the `i4blt -help` command to view flags and usage, or see LUM documentation for graphical equivalents. Figure 3 shows a sample output verifying that products are licensed.

```
# ./var/ifor/i4tv

i4tv Version 4.0 AIX - LUM Test and Verification Tool
(c) Copyright 1995-1997, IBM Corporation, All Rights Reserved
US Government Users Restricted Rights - Use, duplication or disclosure
restricted by GSA ADP Schedule Contract with IBM Corp.
(c) Copyright 1991-1997, Gradient Technologies Inc., All Rights Reserved
(c) Copyright 1991,1992,1993, Hewlett-Packard Company, All Rights Reserved

Completed license transaction on node 123456c running LUM 4.5 AIX
Active License Servers:
 ip:testmachine.ibm_US.com (IBM/AIX) running LUM 4.5 AIX

Active Central Registry License Server:
 ip:testmachine.ibm_US.com (IBM/AIX) running LUM 4.5 AIX
```

Figure 2. Sample command output from i4tv

Adding Product Licenses to LUM Configuration

Although LUM Version 4 is installed and configured on the machine, it is still necessary to enroll licenses to the LUM configuration for license-enabled products. For the license servers, concurrent nodelock and/or concurrent access (network) licenses can be enrolled, depending on the network setup. The product's `README` file generally details this information. The product should have two `.lic` files as specified in the `README` file. The `product_cn.lic` corresponds to the concurrent nodelock license; `product_c.lic` corresponds to the concurrent network license. Be sure to use the correct file for your setup. Figure 4 shows the location of `README` files.

To install the license, use the `i4blt` tool in either GUI or command-line form. Enabled products should provide the appropriate commands to install their licenses. For example, a concurrent nodelock license server installation command line for C/C++ 3.6.x would look similar to Figure 5.

This command can generally be accomplished by modifying the `.lic` file. For example, to enroll Fortran 6.1, locate your license key in `/usr/lpp/xlf/DOC`. Then enroll the concurrent nodelock license by following these steps:

1. `cd /usr/lpp/xlf/DOC`
2. `cp xlfaix_cn.lic /var/ifor`
3. `cd /var/ifor`
4. `chmod 744 xlfaix_cn.lic`
5. `vi xlfaix_cn.lic` (For concurrent it is: `f51aix_c.lic`).

   Enter `/var/ifor/` in front of the `i4blt` command and change

   `-T number_of_lics` —> `-T number of licenses that you purchased!`

   `-R "admin_name"` —> `-R " Your name here!"`

```
#./var/ifor/i4blt -lp
i4blt Version 4.5 AIX - LUM Basic License Tool
(c) Copyright 1995-1998, IBM Corporation, All Rights Reserved
US Government Users Restricted Rights - Use, duplication or disclosure
restricted by GSA ADP Schedule Contract with IBM Corp.
(c) Copyright 1991-1998 Gradient Technologies Inc., All Rights Reserved
(c) Copyright 1991,1992,1993, Hewlett-Packard Company, All Rights Reserved

Products
Vendor Name:  IBM Software Solutions Toronto
Vendor ID:  5ab23c553d4e.09.15.31.05.12.34.56
Product Name:  C and C++ Compilers
Product Version: 3.6.cn
Product ID:  67425
Licenses:  10
In Use Licenses: 0


Vendor Name:  IBM Software Solutions Toronto
Vendor ID:  5ab23c553d4e.09.15.31.05.12.34.56
Product Name:  VisualAge C++ Professional
Product Version: 4.0.cn
Product ID:  1099
Licenses:  5
In Use Licenses: 0


End of Product List
```

Figure 3. Verification of product licenses

File Name	Location of README File
C for AIX 4.x	/usr/vac/README.password
XL Fortran 5.x for AIX (LUM usage optional)	/usr/lpp/xlf/DOC/README.xlf
XL Fortran 6.x for AIX (LUM usage optional)	/usr/lpp/xlf/DOC/README.xlf
VisualAge C++ on AIX	/usr/vacpp/README.password
C/C++ 3.6.x (bundled with VisualAge C++)	/usr/ibmcxx/README.C++
VisualAge for Java on AIX	/usr/IBMVJava/README.password

Figure 4. README file locations

```
/var/ifor/i4blt -a -v "'IBM Software Solutions Toronto'
1ab23c456d7e.89.10.11.12.13.00.f9gh3ijk6lmno" -p "'C SET ++' '1.1.3.cn'
aabbcc0dd1eeffgghhii2jjkk34ii5 'Concurrent Nodelock feature'" -T 'number of
licenses purchased' -R "Your name here"
```

Figure 5. Command line to install concurrent nodelock license server

Delete everything including and below the line that reads:

```
LicenseCertificate!
```

Save the file.

6. Run the script by issuing the command:

```
./xlfaix_cn.lic
```

The product license should be enrolled and compilations can now be run. This same process is used to configure subsequent products; however, only the `.lic` file is modified and run. Running the `i4blt -lp` command will verify that the license is enrolled.

Conclusion

Designing an appropriate license network requires a thorough understanding of the network's composition. License Use Management, a mechanism for licensing and managing license-enabled products, must be installed and configured on both the server and the client. Once that step is completed, adding products requires only a single command or a point-and-click with the `i4blt` tool. License Use Management is the sole source for license administration in AIX 4.3.

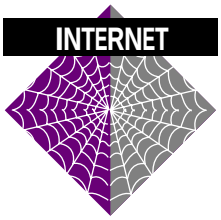
License Use Management Resources

- Visit the IBM License Use Management Web site for up-to-date information and to download LUM Version 4.5: http://www.software.ibm.com/is/lum/
- View the online LUM documentation from your browser if you have `ifor_ls.msg.en_US.base.cli` installed. Open the appropriate file required from `/usr/opt/ifor/ls/os/aix/doc/en_US/` (or appropriate language, if not English = en_US)
- View License User Runtime documentation with your Web browser. After installation, the following publications are available for viewing with your Web browser in the directory `usr/opt/ifor/ls/os/aix/doc`:
  - —*Using License Use Runtime* (lumusg.htm)
  - —*License Use Runtime Command Reference* (lumcmd.htm)
  - —*License Use Runtime Message Reference* (lummsg.htm)
- *Using License Use Management Runtime for AIX Version 4.0*. International Business

Machines Corporation. October 1997. This book is available in .HTM and .INF format, as well as in hard copy. A printable copy of this book in PostScript® format is available for downloading from the IBM FTP site *ftp://ftp.software.ibm.com* in the directory */software/lum*.

***Veronica L. DeHart**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. E-mail: vdehart@us.ibm.com. Ms. DeHart provides technical support for developers on the AIX platform. She graduated from the University of Utah with a Bachelor of Science in Electrical Engineering.*

The WebSphere Application Server: Focus on Servlets



By Vandana Kumar

The IBM WebSphere™ Application Server is software built on open, reusable technologies that leverage your existing resources, shorten development cycles, and ease your administrative burden. This article focuses on the capabilities of the Standard Edition of WebSphere Application Server and Java™ servlets.

IBM's WebSphere Application Server is a Java-based application environment for building, deploying, and managing Internet and intranet Web applications. It is an open and extensible solution that supports HyperText Transfer Protocol (HTTP), Internet Inter-ORB Protocol (IIOP), and Enterprise Java technology.

The WebSphere Application Server (Version 3.01) is available in three versions:

Standard Edition. A comprehensive Java-based Web application platform, which enables interactions with enterprise databases and transaction systems. This edition of WebSphere Application Server has the following major components:

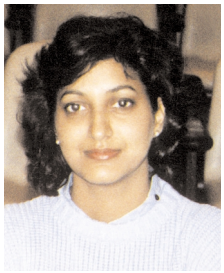
- Java-based servlet and JSP engine that supports major Web servers via plug-ins
- Deployment manager
- Connection manager
- Web site analysis tool
- Application services
- XML document structure services, including a parser and data transformation tools
- The IBM HTTP Web Server with additional Secure Sockets Layer- (SSL-) based security and performance features
- National language support

Advanced Edition. An enhanced version of the Standard Edition that includes server capabilities for applications built to the Enterprise JavaBeans™ (EJB) specifications. It provides relational database transaction management and monitoring based on EJB and CORBA components.

Enterprise Edition. An enhanced version of the Advanced Edition that combines TXSeries™, IBM's world-class transaction application environment, with the full distributed object business process integration capabilities of the Component Broker.

In addition to WebSphere Application Server, the WebSphere product line also includes:

- **WebSphere Performance Pack.** Web facilities management software that supports rapid growth of high-volume Web sites by combining caching, load balancing, and Web site replication in a single package.



Vandana Kumar

- **WebSphere Performance Pack Cache Manager**. An entry-level, proxy caching solution for small customer locations (10 to 50 user installations) running on Linux™ and Windows NT®.

- **WebSphere Studio**. A set of integrated Web development tools that makes it easy to create dynamic content and Java-based applications for the WebSphere Application Server. It includes wizards, a workbench, and other development tools for building new and powerful Web applications.

- **WebSphere Payment Manager.** Software that enables service providers to quickly and easily link merchants to payment processors to handle today's complex e-commerce transactions. This helps service providers to grow revenue from existing merchants while attracting new ones.

> ***Websphere is a Java-based servlet engine that is independent of both the Web server and the underlying operating system.***

This article focuses on the capabilities of the Standard Edition of WebSphere Application Server and Java servlets.

WebSphere runs on many operating systems including Windows NT, Sun Solaris®,

The WebSphere Application Server: Availability and Requirements (Version 3.01)

The trial version of the WebSphere Application Server (Standard Edition, Version 3.01) can be downloaded for free from the following site: http://www.software.ibm.com/websphere/appserv/download.html.

System requirements necessary before installing and running the WebSphere Application Server include:

- A minimum of 128 MB RAM, with 512 MB recommended and 40 MB free disk space. The Application Server installation CD includes Application Server, IBM HTTP Server, a Java Development Kit (JDK), and IBM DB2® Version 5.2
- AIX Version 4.3.2 or later
- One of the following Web servers:
  - — IBM HTTP Server Version 1.3.6 (available in the WebSphere Application Server package)
  - — Apache Server Version 1.3.6 (minimum) for AIX
  - — Domino™ Version 5.0 for AIX
  - — Lotus® Domino Go Webserver Version 4.6.2.5 for AIX
  - — Netscape Enterprise Server Versions 3.5.1 or 3.6 (recommended) for AIX
- Java Development Kit (JDK), JDK 1.1.6 with Fix Pack 9. JDK 1.1.6 ships with Application Server
- Web browser. The Application Server Manager is the user interface for managing servlets. An appletviewer or a browser that supports JDK 1.1.6, for example Netscape 4.06 or 4.07, is necessary to run the Application Manager
- Database. IBM DB2 Version 6.2 with Fix Pack 1a or Version 5.2 with Fix Pack 10 or 11 is installed with Application Server to support the Enterprise Java Services (EJS). If using EJS, ensure that the DB2 sample database has been created and can be accessed with a designated user ID

Note: All Fix Packs can be downloaded from http://www.software.ibm.com/websphere.

and AIX®. The Performance Pack add-on makes WebSphere extremely scalable. The WebSphere Performance Pack contains a powerful software solution called SecureWay® Network Dispatcher, designed specifically to enhance the scalability and reliability of e-business solutions.

WebSphere is designed to support the full Servlet API on several popular Web servers and operating systems. Additionally, it allows connections to most common databases and supports high-load transaction processing. Finally, it offers the choice of industry-standard Web servers, including IBM's HTTP Server based on Apache. This latter point is especially important, since Apache is the market-share leader in Web servers (see next section).

IBM's WebSphere is critical for organizations that have identified Java as a strategic technology in deploying moderate to highly complex applications requiring transactional processing (or high growth in traffic) and integration with legacy data.

Installing and Configuring the Web Application Server[1]

The Web server should be installed prior to installing the Application Server. See the article entitled "Quick Start Guide for the IBM HTTP Server" in this issue for more information on installing and configuring the IBM HTTP Web server.

The Application Server installation changes the httpd.conf file, making the Web server direct certain requests, such as servlet requests, to the Application Server. If the Web server is installed after the Application Server installation, these changes will not be made. JDK should also be installed before installation of the WebSphere Application Server. If a previous JDK is being replaced with a new one, be sure to clean up the `classpath` variables. The Application Server filesets can be installed using either the graphical user interface (GUI) or the system management interface tool (SMIT) administrative tool, although the GUI method is recommended.

Once all the files have been uncompressed and extracted from the tar file, run the `install.sh` script to install the software. If you install files from the CD, insert the installation CD, then go to the subdirectory AIX and run the setup executable `./install.sh`. A series of panels will guide you through the installation process. It is important to note that any previous installation of WebSphere must be de-installed prior to installing a new version of the software, and the Web server must be stopped before installing the Web Application Server.

Note that the previous WebSphere administration ID and password will continue to be valid for the new installation as well.

The Web server can be stopped by issuing the following command:

```
/usr/lpp/HTTPServer/sbin/apachectl stop
```

It can be restarted after WebSphere Application Server is completely installed by issuing the following command, which automatically starts the Application Server:

```
/usr/lpp/HTTPServer/sbin/apachectl start
```

The `/usr/lpp/HTTPServer/sbin/apachectl stop` command does not stop the Java processes that are started by Application Server. These processes should be stopped before restarting the Web server. To stop these processes, issue the `ps -ef | grep -i java` command to see their process IDs, then kill those processes.

Figure 1 shows a script that can stop the Web server, kill the Java processes, then restart the server in only one step.

Be sure that the super user has execute permission. The Application Manager interface can now be started by going to the Web site:

http://<hostname>:9527

where *hostname* is the name of the host machine on which the Web server and the Application Server are installed.

Execute a sample servlet to check the correct functioning of the Application Server:

[1] See http://www.software.ibm.com/websphere for details about installing WebSphere Application Server.

```
cleanstart.sh

#!/bin/ksh

echo "------------------------------------------------------------"
echo "Stop IBM HTTP server and Application server"
echo "------------------------------------------------------------"

echo "---------------> stopping IBM HTTP server"
/usr/lpp/HTTPServer/sbin/apachectl stop
kill -9 `ps -e | grep httpd | awk `{print $1}'`
ps -e | grep http
sleep 1

echo "------------------------------------------------------------"
kill -9 `ps -e | grep java | awk `{print $1}'`
kill -9 `ps -e | grep java | awk `{print $1}'`
sleep 1

echo "------------------------------------------------------------"
echo "DONE"
echo "------------------------------------------------------------"
```

Figure 1. Stopping the Java processes

http://<hostname>/servlet/snoop

where *snoop* is an alias for the servlet SnoopServlet.

Once the Application Manager interface is started, the help documentation can be accessed by clicking the "?" button in the upper right-hand corner of the window.

The `<as_root>` used in the rest of this article refers to the full pathname of the directory where the Application Server is installed.

Installing and Configuring DB2

Several examples require connections with the DB2 database to run properly. This section outlines the most basic and simple steps for installing and configuring DB2 to run the database servlet examples.[2]

Install the following filesets from the CD:

1. License support for DB2 Connect, Enterprise Edition
2. DB2 Client application enabler
3. Code Page conversion tables
4. DB2 Connect
5. DB2 communication support (TCP/IP, for example)
6. DB2 administration server
7. DB2 engine, RTE, and sample database source
8. License support for DB2 Universal Database (UDB), Enterprise Edition
9. Java support
10. Open Database Connectivity (ODBC) support
11. DB2 replication

After the filesets are installed on the system, use the DB2 installer program `db2setup` to install DB2. The DB2 filesets are installed in the `/usr/lpp/db2_05_00` directory. The `db2setup` script is located in the install directory under the DB2 root directory. Use this script to create a DB2 instance and the administration server. By default, this creates a DB2 user `db2inst1` and an administrator `db2as`. Their home directories will have a .profile file and a directory called `sqllib` in them. The .profile file sets the environment variables for the DB2 user, and

[2] A 60-day evaluation copy of DB2 Universal Database Version 6.1 is available free on the AIX 4.3.2 Bonus Pack CDs. It can also be downloaded from the following Web site: http://www.software.ibm.com/db2.

the sqllib directory has several files linked from the DB2 root directory.

Next, add the DB2 JDBC driver to the Application Server's CLASSPATH by opening the Application Server Manager (*http://<hostname>:9527*), then selecting "Setup" of the Java engine. On the Path tab, ensure that the DB2 sqllib/java/db2java.zip is in the Java classpath field.

An alternative way to do this is to edit the WebSphere's bootstrap.properties file <as_root>/properties/bootstrap.properties as follows:

```
java.classpath=/usr/jdk_base/lib/
classes.zip:<as_root>/classes:
<as_root>/web/classes:/home/db2inst/s
qllib/java/db2java.zip.
```

Next, edit the *.servlet files in the Xtreme Adventure and WebBank samples to enable the applications to access DB2. These files are in <as_root>/servlets/XtremeTravel/ for the XtremeAdventure example and in <as_root>/servlets/WebBank for the WebBank example.

Specify the database, the userID, and the password in these files. For example, if your DB2 user is db2inst1, then specify that as the userID and enter the password for db2inst1.

Copy the <as_root>/samples/login.properties into the <as_root>/servlets directory, and again enter the userID and password.

Open the DB2 command window on the DB2 server by typing db2. The prompt will change to the db2=> prompt. This must be done as a DB2 user or administrator so that all the environment variables are enabled. Populate the database tables needed for these examples by running the createdb.bat script in these directories: <as_root>/samples/XtremeTravel/database and <as_root>/samples/WebBank/database.

This completes the setup for the database examples.

Servlet Support in WebSphere

WebSphere is a Java-based servlet engine that is independent of both the Web server and the underlying operating system. The servlet engine enables the true implementation of write-once-use-anywhere code.

Among other important components, the software includes a Servlet Manager application that lets the WebSphere Application Manager's administrator manage the servlets from any location on the network. The Servlet Manager allows options to be set for loading the servlets, initializing parameters, setting classpaths and servlet aliases, monitoring system resource usage, and much more. It is said that servlets are to servers as applets are to browsers.

> ***Once a servlet is loaded, it stays in memory and can begin handling requests immediately after invocation, without spawning new processes or interpreters.***

Java Servlets

Java servlets are extended Java classes that can be loaded dynamically to expand the server's functionality. Servlets reside completely within the server environment and run within the Java Virtual Machine (JVM) on the server, making them safe and portable. Servlets are very portable, flexible, extensible, and elegant for dynamic Web content development. The servlets operate on the server side only; hence, no special requirements are imposed on the client side except basic Java support. It is also possible to handle Enterprise JavaBeans and JavaBeans™ from within the servlets to enable linkage with corporate legacy systems. The tight integration of the servlets with the servers allows close cooperation between them.

Once a servlet is loaded, it stays in memory and can begin handling requests immediately after invocation, without spawning new processes or interpreters. Since a separate thread handles the cache request, the servlets are highly scalable entities. Servlets can maintain their state and hold on to database connections with persistence.

Servlets can be used to generate entire Web pages, or they can be added to static pages by using the <SERVLET> tag (known as server-side include). They can also be used in cooperation with other servlets to filter the contents in servlet chains, similar to the UNIX® command piping mechanism.

It is now possible for snippets of servlet code to be embedded directly into the HTML pages by using a new technique called JavaServer Pages (JSP), which can have JavaBeans directly embedded in them via the <BEAN> tag. JavaBeans are reusable Java classes that follow specific naming conventions. They can perform well-defined tasks such as executing database queries or maintaining information about the client. They also can make the resulting information available to the JSP page through simple accessor methods.

Compiling Sample Servlets

The Java Servlet Development Toolkit is provided with the Application Server software as jsdk.jar file in the /usr/lpp/IBMWebAS/lib directory. Using the following command, extract the jsdk libraries:

```
jar xvf ./jsdk.jar
```

Next, set the CLASSPATH, and LIBPATH environment variables in the shell where compilations are going to occur, as shown in Figure 2.

```
export CLASSPATH=/usr/jdk_base/lib/classes.zip:/usr/
   lpp/ IBMWebAS/classes:/usr/lpp/IBMWebAS/lib

export LIBPATH=/usr/lpp/IBMWebAS/lib:/usr/
   jdk_base/lib/aix/native_threads
```

Figure 2. Setting the CLASSPATH and LIBPATH environments

Servlets can be compiled by issuing the javac command:

```
javac <my_servlet.java>
```

This produces a my_servlet.class file in the local directory. To use the Application Server Manager to manage new servlets, copy the servlets from their current location to the <as_root>/servlets directory. If you do not want to copy the servlets to this directory, use the Reloadable Servlet CLASSPATH field of the Java Engine page of the Manager interface to specify additional directories to be monitored. Use the Configuration page of the Manager interface to reconfigure existing servlets.

Note that for the Web server and the Application Server to read/write any file on the server, everyone must have read/write file permissions.

A Simple Servlet Example

Figure 3 shows a simple servlet example (FunDayServlet.java) that takes the input

```
FunDay.html

<html>
<head>
<title> PID Golf Sign-Up Form</title>
</head>

<body bgcolor="#FFFFFF">

       <p>The PID team is
       organizing a scramble golf tournament. It works like
        this:</p>
        <ul>
            <li>Divide the group into 4-person teams (each team
                to have at least one experienced
```

(continued on following page)

Figure 3. Simple servlet example

(continued from previous page)

```
                &quot;golfer&quot;)</li>
            <li>Each team will play 9 holes of golf in a scramble
                style</li>
        </ul>
        <p>To make teams, we need to know some information beforehand
        (please fill this information in ONCE AND ONLY ONCE):</p>
        <form action=http:/servlet/FunDayServlet method=POST>
            <p> </p>
            <p><strong><u>What is your First/Last name:</u></strong></p>
            <p>Last Name: <!—webbot bot="Validation"
            b-value-required="TRUE" i-minimum-length="1"
            i-maximum-length="40" —><input type="text" size="40"
            maxlength="40" name="strLastName"> </p>
            <p>First Name: <!—webbot bot="Validation"
            b-value-required="TRUE" i-minimum-length="1"
            i-maximum-length="40" —><input type="text" size="40"
            maxlength="40" name="strFirstName"></p>
            <p> </p>
            <p><strong><u>Please rank your golf playing ability:
            </u></strong></p>
            <p><input type="radio" checked name="rdoAbility"
            value="V1">I've never hit a golf ball</p>
            <p><input type="radio" name="rdoAbility" value="V2">I
            can barely hit a golf ball</p>
            <p><input type="radio" name="rdoAbility" value="V3">I
            can really hit a golf ball</p>
            <p><input type="radio" name="rdoAbility" value="V4">I
            hit golf balls in my sleep</p>
            <p> </p>
            <p><strong><u>Please let us know your preferred prize
            if you win:</u></strong></p>
            <p><input type="radio" checked name="rdoPrize"
            value="V1">I would like a 1 week trip to Maui</p>
            <p><input type="radio" name="rdoPrize" value="V2">I
            would like dinner in Paris for Year 2000 New Years</p>
            <p><input type="radio" name="rdoPrize" value="V3">I
            would like my mortgage/rent payed for 1 year</p>
            <p><input type="radio" name="rdoPrize" value="V4">I
            would like an RS/6000 T-Shirt</p>
            <p> </p>
            <p><input type="submit" name="btnSubmit"
            value="Submit"></p>
        </form>

<p>— Thanks, The RS/6000 PID fun day team</p>

<p> </p>

<p> </p>

<p> </p>
</body>
</html>


FunDayServlet.java
```

(continued on following page)

Figure 3. Simple servlet example

(continued from previous page)

```
// FunDayServlet
// Written by Vandana Kumar
// Date: June 1, 1999

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class FunDayServlet extends HttpServlet
{
  // set max class size as needed for your application
  public final int MAX_CLASS_SIZE = 2;

  // set ResourceBundle name
  public final String BASENAME = "samples";
  static ResourceBundle rb = null;

  public void init(ServletConfig config) throws ServletException
  {
    super.init(config);
    /* get the ResourceBundle for localization */
    try{
      rb = ResourceBundle.getBundle(BASENAME);
 }
    catch(Throwable e) {

      e.printStackTrace();
    }
  }

public synchronized void doPost(HttpServletRequest req, HttpServletResponse
res)
throws ServletException, IOException
{
      String SurveyName = null;

      // Response is html
      res.setContentType("text/html");

      //Get the response's PrintWriter to return text to the client.
      PrintWriter out = res.getWriter();

      // Get the input text
      String lastname = req.getParameter("strLastName");

      String firstname = req.getParameter("strFirstName");

      String level = req.getParameter("rdoAbility");

      // Write the result back to the client

if (lastname != null && firstname != null) {
        out.println("<html>");
        out.println("<h1>(This is a confirmation of your
participation...)</h1>");
        out.println("<br>");
```

(continued on following page)

Figure 3. Simple servlet example

(continued from previous page)

```
        out.println("<br><h3>");
        out.println("Thanks for responding " + firstname + " ...");
        out.println("</html>");

        // Write results to a file

        FileWriter golfFile = new FileWriter("/funday/survey_results/Survey.txt", true);
        PrintWriter toFile = new PrintWriter(golfFile);

        toFile.println( firstname + " " + lastname + " | Golf Skill level: " + level);
        toFile.flush();

        // We are done - close the writer
        out.close();
        }
    }
}
```

Figure 3. Simple servlet example

from the end-user's form and stores it in a file on the server.

Once the `FunDayServlet.java` compiles cleanly and results in a `FunDayServlet.class` file, the class file is copied to the `<as_root>/servlets` directory so that the `ACTION` attribute can directly refer to the servlet as http://servlet/FunDayServlet, and the WebSphere Application Manager can automatically reload it if it is changed or updated.

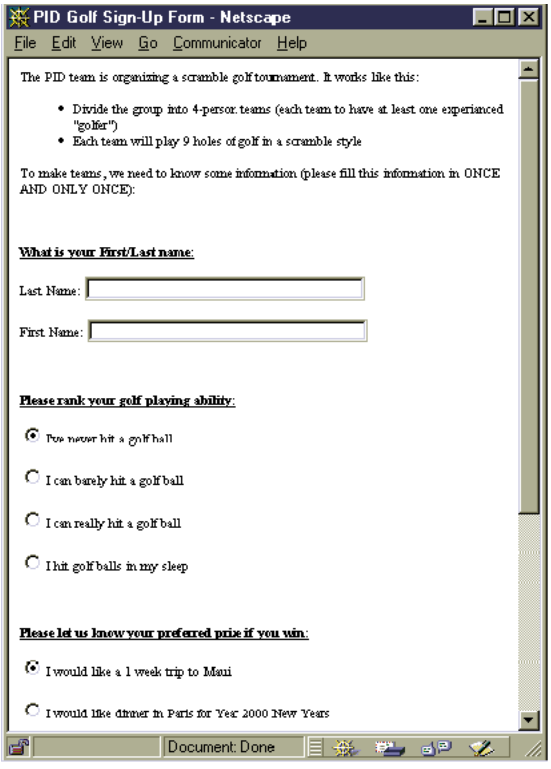


Figure 4. Funday example

When the browser displays the HTML file `FunDay.html`, the end user completes the information and presses the Submit button. The information submitted is sent to the FunDayServlet, which takes the input, stores it in the file `survey.txt` on the server, and displays a confirmation to the user in HTML format. See Figure 4.

This servlet was actually used to collect participation responses for organizing a departmental team-building event for employees who were located in several geographical regions.

A Simple JSP Example

Figure 5 shows a simple example of a Java Server Page.

The main difference between server-side includes and JSPs is that JSPs include actual code snippets. This can be useful when small pieces of dynamic content are located between static content throughout the pages. JSPs require support in the Web servers and browsers; therefore, the JSP code is enclosed inside `<%, and %>` tags so it is hidden for servers that do not support JSPs.

Behind the scenes, JSPs are automatically created, compiled, loaded, and run via a special servlet by the Application Server.

```
name.html

<html>
<head>
<title>Introductions</title>
</head>
<body>
<p>Please enter your name here
<br>
<form method=GET ACTION="/IBMWebAS/samples/vandana/printname.jsp">
Name
<input type=text name="name">
<input type=submit>
</form>
</body>
</html>


printname.jsp

<html>
<head><title>Hello</title></head>
<body>
<h1>
<%
    if (request.getParameter("name") == null) {
        out.println("Hello World");
    }
    else {
        out.println("Hello, " + request.getParameter("name"));
    }
%>
</h1>
</body></html>
```

Figure 5. Java Server Page example

Thus, the page containing JSP will load slowly the first time, but will be very fast for subsequent requests since the servlet is already loaded.

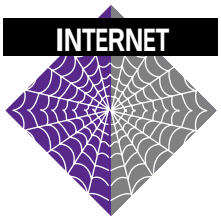
References

The Help files of the Application Manager provide much information once the Application Server is installed and the Application Manager is running. See additional information on "Getting Started" at http://www.software.ibm.com/webservers/appserv/library.html under the WebSphere Server documentation.

The following Web sites contain additional information about specific products mentioned in this article:

- www.ibm.com/aix
- www.software.ibm.com/websphere
- www.software.ibm.com/db2
- www.rs6000.ibm.com

Vandana Kumar, *IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Ms. Kumar is an e-business technologist in the IBM Server Group. She has worked with several AIX application vendors in porting, enabling, and tuning their software on the RS/6000 platform. Ms. Kumar has a MS in Electrical Engineering from Wayne State University.*



Quick Start Guide for the IBM HTTP Server

By Vandana Kumar and Andrew Taylor

This Quick Start Guide is designed to lead you through the steps necessary to quickly download, install, configure, and run the IBM HTTP Server powered by Apache on your RS/6000®. Also included are steps and hints needed to successfully compile and extend the features of the IBM HTTP Server.

The HTTP Server included in IBM's WebSphere family of Application Server products is based on the Apache HTTP Server. Developed by the Apache Software Foundation, formerly known as the Apache Group, it is the most widely used HTTP Server on the Internet. The "official" Apache distribution is available from the Apache Web site, http://www.apache.org, in both source form and precompiled binaries for almost any platform, including IBM AIX, HP-UX, Silicon Graphics Inc. (SGI) Irix, Sun Solaris, Linux, Windows NT, and others.

The IBM HTTP Server powered by Apache has several additional significant features and offerings over the standard Apache server. These enhancements include:

- Easy installation via the standard AIX install process (`installp` and `smit`)
- Support for Secure Sockets Layer (SSL) Versions 2 and 3 for secure connections between browser-based clients and the HTTP Server
- Support and service offerings directly from IBM

Downloading and Installing

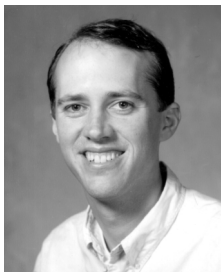
The IBM HTTP Server 1.3.3.1 is available on a complimentary basis from the following URL: http://www.software.ibm.com/webservers/httpservers/download.html. When this article was written, the most current version available on the IBM site was Version 1.3.6. This download is fully functional and not time-disabled.

Follow the instructions on this Web page to successfully download the proper version of the HTTP Server (RS/6000 system requirements are noted on the Web site). Based on your country of origin and your encryption needs, your choices include 40-bit, 56-bit, and 128-bit versions of the software. You must register to gain access to the download page, but registration is quick, painless, and free!

After downloading the IBM HTTP Server, you will have one of the following files:

- HTTPServer.aix.128.tar.Z
- HTTPServer.aix.56.tar.Z
- HTTPServer.aix.40.tar.Z

Vandana Kumar



Andrew Taylor

The exact filename will depend on the encryption level you select when downloading.

Once you reach this point, follow the instructions in Figure 1 for quick installation (These instructions assume you have downloaded HTTPServer.aix.128.tar.Z and that this file is in `/tmp`).

Testing Your HTTP Server Installation

The easiest way to test your new HTTP Server installation is to try it. But first, you need to start the actual `httpd` daemon processes to activate the HTTP Server. This is best accomplished by using the `apachectl` command:

```
Export PATH=/usr/lpp/HTTPServer/
   sbin:/usr/lpp/HTTPServer/bin:$PATH
apachectl start
```

Next, use the following command to confirm that the `httpd` processes (There should be six `httpd` processes) started successfully:

```
ps -e | grep httpd | grep -v grep
```

Finally, you are ready to actually try connecting to your HTTP Server through your Web browser. At your Web browser, type in the URL of your newly installed HTTP Server. For example, if the hostname of your test machine is abc.austin.ibm.com, type *http://abc.austin.ibm.com*. Figure 2 shows the default home page that is displayed after the IBM HTTP Server is installed.

Important Files in Your HTTP Server Installation

When you install the IBM HTTP Server, approximately 400 files are installed into the `/usr/lpp/HTTPServer` directory. The following list provides some important files found in the installation directory:

- `/usr/lpp/HTTPServer/etc/httpd.conf` —main configuration file for the HTTP Server (defines document root directory, number of initial `httpd` processes, error log directory, and so on)

```
log in as root
cd /tmp
uncompress HTTPServer.aix.128.tar.Z
tar -xvf HTTPServer.aix.128.tar
cd images
installp -acF -d. all
```

Add `/usr/lpp/HTTPServer/sbin` and `/usr/lpp/HTTPServer/bin` to your PATH (optional)

Note: When the `installp` command is issued, the HTTP Server will be installed in the `/usr/lpp/HTTPServer` directory.

Figure 1. Quick installation instructions

Figure 2: The Default home page of the IBM HTTP Server

- `/usr/lpp/HTTPServer/sbin/apachectl` —used to start/stop/restart the HTTP Server
- `/usr/lpp/HTTPServer/var/run/httpd.pid`—contains the process ID for the parent `httpd` process (this is the process that forks the others)
- `/usr/lpp/HTTPServer/var/log/*`—contains log files that are automatically generated by the `httpd` processes
- `/usr/lpp/HTTPServer/share/htdocs`—default document root directory

```
cd /usr/lpp/HTTPServer/apachesrc/apache-1.3
./configure –prefix=/usr/lpp/HTTPServer1 –enable-module=so
make
make install
```

Figure 3. Install HTTP Server into the directory

- `/usr/lpp/HTTPServer/share/htdocs/index.html`—default HTML file for the HTTP Server (such as *http://yourhostname*)
- `/usr/lpp/HTTPServer/share/cgi-bin`—default cgi-bin directory

Compiling the IBM HTTP Server

You may need to recompile the IBM HTTP Server, for example, to add third-party modules. This can be done quickly and easily. However, before proceeding, it is important to make sure you have at least Version 3.6.6 of the IBM C and C++ Compiler on your system. This can be checked with the following command:

```
lslpp -L | grep ibmcxx
```

The source code to Version 1.3.3 of the Apache HTTP Server is stored in a `gzip` file called `/usr/lpp/HTTPServer/apachesrc/apache_tar.gz`. Prior to compiling, you must extract the actual source code using the following steps:

```
cd /usr/lpp/HTTPServer/apachesrc
gunzip apache_tar.gz
tar -xvf apache_tar
```

Note: `gzip` and `gunzip` do not ship with AIX, but you can download AIX binaries for these programs from *http://www.gzip.org*.

The extraction process will create the directory `/usr/lpp/HTTPServer/apachesrc/apache-1.3`. The steps shown in Figure 3 will configure, build, and install the HTTP Server into the directory `/usr/lpp/HTTPServer1`.

Note: If you have the gnu compiler (gcc) installed, you may have problems building. If so, edit the `src/Configuration.tmpl` file and change the `#CC=3D` line to be `CC=3Dcc`. This will ensure that the IBM C and C++ compiler is used.

To test the new installation, make sure that another instance of the HTTP Server is not already running. Then, start the new installation by using the `apachectl` command:

```
/usr/lpp/HTTPServer1/sbin/
   apachectl start
```

Dynamic Shared Object Support

The IBM HTTP Server can load third-party modules into its address space at runtime via the AIX Dynamic Shared Object (DSO) mechanism, which uses the `dlopen/dlsym` system calls. In this way, the third-party shared modules do not have to be linked into the HTTP Server executable. Instead, they reside in the file system and can be dynamically loaded at runtime as needed.

> ***The IBM HTTP Server can load third-party modules into its address space at runtime via the AIX Dynamic Shared Object (DSO) mechanism.***

The DSO support for loading these shared modules is based on an Apache module called `mod_so.c`, which must be statically linked into the HTTP Server executable. All other distributed Apache modules can then be placed into a DSO by first building the actual shared object, then specifying the HTTP Server to load the new shared modules. Details on this process can be found in the following HTML file: `/usr/lpp/HTTPServer/apachesrc/apache-1.3/htdocs/manual/dso.html`.

Frequently Asked Questions

The following questions may occur while downloading, installing, testing, and building the IBM HTTP Server based on Apache for RS/6000.

What does it mean when I get "Expected </Directory> but saw <?Directory>" when I try to start Apache?
A previous level of the xlC compiler causes this message. To avoid this, please download and install the following PTFs from the IBM support site (http://service.software.ibm.com/support/rs6000):

- U462005
- U462006
- U462007
- U462023—U462030

Why do I get compilation errors?
Check that the gcc compiler is not inadvertently getting called. Also ensure that the compiler filesets are a consistent level and the C header files are in sync with the compiler and the operating system.

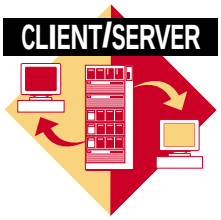
Additional Information

The following Web sites provide the most recent updates to this information and papers in the RS/6000 e-business Technical Series:

- http://www.software.ibm.com
- http://www.rs6000.ibm.com
- http://www.developer.ibm.com

Vandana Kumar, *IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Ms. Kumar is an advisory engineer in the IBM Server Group. She has worked with several AIX application vendors in porting, enabling, and tuning their software on the RS/6000 platform. Ms. Kumar has a MS in Electrical Engineering from Wayne State University.*

Andrew Taylor, *IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Taylor currently works in the IBM Server Group in the RS/6000 Division. He focuses on e-business technologies and helps IBM Business Partners to port, tune, and deliver their solutions on AIX and the RS/6000.*



Distributed Computing with Java RMI

By Greg Flurry

Java offers a wide range of mechanisms for creating vendor-neutral, object-based applications. One mechanism, Remote Method Invocation, gives programmers the ability to distribute pieces of an application across different machines. This article examines the architecture of the Java Remote Method Invocation mechanism, discusses the programming model and some alternative technologies, and finally presents an example on AIX®.

The Java™ platform designers realized that situations occur in which parts of an application must be distributed across multiple physical systems. There are many possible reasons for distributing an application. For example, the natural design of an application can require some pieces to run on different systems, such as a groupware application. An application might leverage special processing capabilities on a particular system, such as a vector processor. Or an application might take advantage of unused processing capacity from systems available only at certain times, such as personal systems only in use during normal business hours. Finally, an application might require increased availability or fault tolerance.

In Java, distributed computing means interaction between objects instantiated on different Java Virtual Machines (JVMs), which may be running on the same or different physical machines. The Java designers created the Remote Method Invocation (RMI) mechanism to support a distributed object model for Java.

According to the Java RMI Specification, RMI "is the action of invoking a method of a remote interface on a remote object. Most importantly, a method invocation on a remote object has the same syntax as a method invocation on a local object."[1]

Java RMI System Architecture

Figure 1 shows the basic architecture of the Java RMI runtime system, which comprises three independent layers: the stub/skeleton layer, the remote reference layer, and the transport layer. The boundaries between layers are defined by specific interfaces and protocols, so any layer can be replaced by an alternate implementation without impacting the other layers. The Java Development Kit (JDK) provides default implementations of all three layers.

[1]For an overview of RMI, see *http://www.javasoft.com/marketing/collateral/javarmi.html*. See the full RMI specification for Java Development Kit (JDK) 1.1, which is the latest version available for AIX, at *http://www.java.sun.com/products/jdk/rmi/index.html*. The Web page has links to other useful RMI-related resources as well, including RMI for JDK 1.2, which is somewhat different from RMI for JDK 1.1.

Greg Flurry

Stub/Skeleton Layer

As shown in Figure 1, an object invoking a method on a remote object actually calls a *stub* (or proxy) for the remote object to interact with the remote object. This stub is an implementation of the remote interfaces of the remote object. When called, the stub does the following:

- Marshals the arguments onto a *marshal stream*, which it obtains from the remote reference layer. Primitive types, such as `int` and `boolean`, are "passed by value" as with local Java method invocation. Local objects are marshaled using object serialization; therefore, they are also "passed by value" because a copy of those objects will run on the remote server. Remote objects are "passed by reference;" that is, a remote object reference is placed in the marshal stream.
- Informs the remote reference layer that the remote object should be invoked.
- Unmarshals any returned object or exception from the marshal stream.

A *skeleton* (or proxy) for a remote object is a server-side entity that dispatches calls to the actual remote object implementation. The skeleton, called by the remote reference layer, does the following:

- Unmarshals arguments from the marshal stream created by the stub.
- Calls the actual remote object implementation.
- Marshals the return value of the call or exception onto the marshal stream.

Remote Reference Layer

The *remote reference layer* implements the desired remote reference semantics between the client and server objects via a specific remote reference protocol. This protocol is independent of the client stubs and server skeletons. Each remote object implementation chooses its own remote reference semantics. For example, it may choose to run on a single server or to behave as a replicated object running on multiple servers.

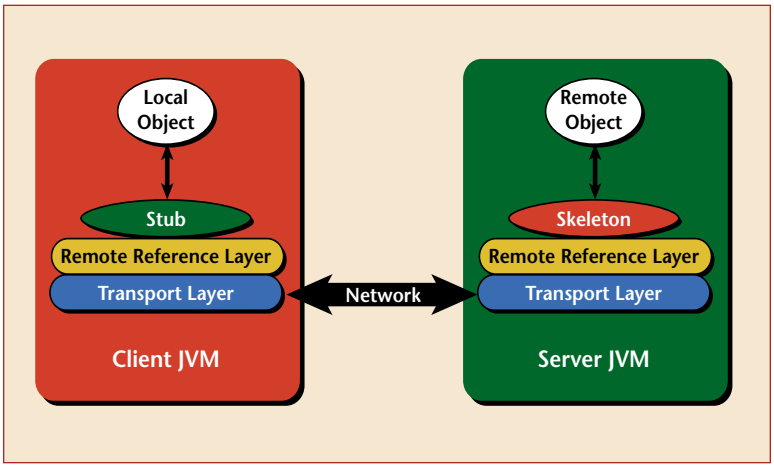


Figure 1. Java RMI architecture

The remote reference layer also abstracts the reference semantics for the server. For example, it hides the different ways of referring to remote objects that reside on servers that are always running, and servers that are run only when a client invokes a method on a remote object served by the server.

Transport Layer

The remote reference layer communicates with the transport layer of the RMI system via an abstraction called a stream-oriented *connection*. The *transport layer*, which actually implements connections, is responsible for the following:

- Setting up and tearing down connections to remote address spaces.
- Monitoring connection activity.
- Listening for incoming calls to remote objects in an address space.
- Maintaining a table of remote objects that reside in a particular address space.
- Dispatching the remote call to the target remote object by forwarding the remote call up to the remote reference layer; the transport layer uses an RMI-unique wire protocol called the Java Remote Method Protocol (JRMP) to communicate between JVMs.

In summary, when a local object calls a method on a remote object, it actually calls

a stub representing the remote object. The stub marshals the arguments for the method, serializing as necessary. The stub forwards the remote invocation request to the remote reference layer.

The client side of this layer handles any client-specific behavior, then forwards the remote request to the transport layer. The client-side transport layer creates a network connection between the client and server JVMs and sends the remote request over the connection to the server-side transport layer.

The server-side transport layer looks up the requested remote object, then forwards the remote call to the server-side remote reference layer. The server side of this layer handles any necessary server-side behavior and hands off the request to the skeleton. The skeleton for the remote object unmarshals the arguments and calls the remote object implementation, which carries out the actual processing requested.

The return value of a call is marshaled by the skeleton, sent down through server-side reference layer and transport layer, over the connection, up through the client-side transport layer and remote reference layer, and then to stub on the client side, where it is unmarshaled and returned to the client object.

Of course, before a client object invokes a method on a remote object, it must get a reference to an implementation of the remote object. Clients use a *name server* to acquire remote object references. Remote object servers must register the objects they serve with the name server. Clients can then access the name server to get the remote object references.

RMI comes with a simple name server, the `rmiregistry`, which must reside on a known machine. It provides a flat name space and does not support automatic instantiation of remote objects. Applications can define and use other, more sophisticated name server mechanisms that are beyond the scope of this article.

Dynamic Class Loading with RMI

One major consideration for distributed computing in the Java environment is the location of the classes required for successful method invocation on a remote object. These classes include the remote objects and their interfaces, the stubs and skeletons that serve as proxies for remote objects, and other classes used as parameters to, or return values from, remote method invocations.

RMI uses a mechanism called *dynamic class loading* to load at runtime the classes required to handle method invocations on a remote object. In Java, the class loader used to load the initial Java class is used to load all the interfaces and classes that are used directly by the class. For example, the default class loader will load the class whose `main()` method is run using the `java` command from the directories listed in the local `CLASSPATH` environment variable; all classes used directly by that class are also loaded by the default class loader from directories in the local `CLASSPATH`.

RMI uses a mechanism called dynamic class loading to load at runtime the classes required to handle method invocations on a remote object.

When the `AppletClassLoader` is used to download a Java applet from the location indicated by the codebase attribute, all classes used directly in the applet are subsequently loaded by the `AppletClassLoader`.

RMI provides its own class loader, `RMIClassLoader`, which loads classes not directly used by the client-side or server-side class; that is, the stubs and skeletons of remote objects, and the classes of arguments and return values for remote method calls. The `RMIClassLoader` searches for these classes in the following locations:

- In the directories listed in the local `CLASSPATH`
- At the Uniform Resource Locator (URL) encoded in the marshal stream (that of a non-default class loader if one is used), or the URL specified by the local property `java.rmi.server.codebase` if defined, for both local and remote parameters or returned values

- At the URL specified by the local `java.rmi.server.codebase` property for stubs and skeletons of remote objects created in the local JVM

Typically, classes are downloaded from either the client or the server using a Web server that is already running. As shown in Figure 2, after a remote object server registers a remote object with a name server, the client can find the remote object using the name server. When the client calls a method on the remote object, it can download the necessary stub from the server. When the remote object executes, the server can download the classes for the parameters and any other necessary classes from the client. Similarly, the client can download any required returned classes from the server.

Security is obviously a concern with dynamic class loading, since classes from untrusted sources can be loaded from the network. Thus, when the `RMIClassLoader` attempts to load classes from the network, a security manager must be in place or the RMI system throws an exception. The security manager must ensure that classes loaded from the network adhere to the standard Java safety guarantees; for example, classes are loaded from "trusted" sources and there is no attempt to access potentially dangerous resources.

Applications can use either the standard `RMISecurityManager` that comes with the JDK or their own security manager. In either case, starting the security manager must be the first action by an application. A security manager must be running for classes to be loaded from network sources.

Distributed Garbage Collection

Local JVMs automatically delete unreferenced local objects to reduce application complexity and to make the developer's job easier. Similarly, RMI can automatically delete remote objects that are no longer referenced by any client. RMI uses a reference-counting garbage-collection scheme that tracks all live client/server connections.

The distributed garbage-collection algorithm interacts with the local JVM's garbage collector in the usual way. As long as remote (or local) reference to a remote object exists, the remote object will not be garbage-collected; otherwise, the garbage collector calls the objects' `finalize()` method and then removes the object.

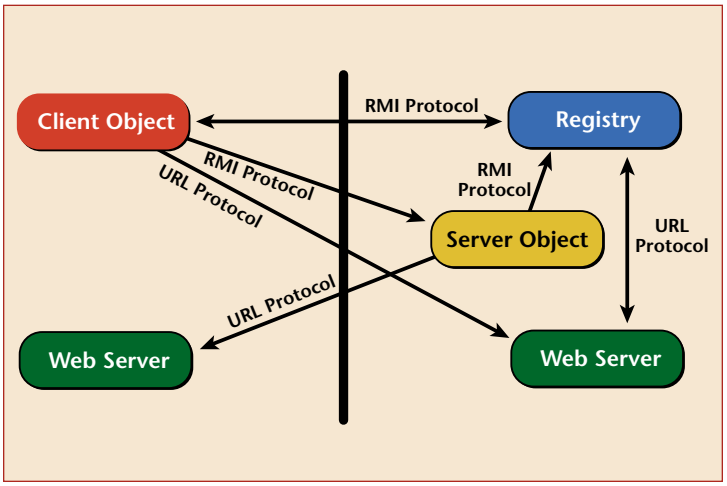


Figure 2. RMI dynamic class loading

Programming with RMI

Although the RMI mechanisms sound complex, fortunately, the vast majority of RMI users need to understand little about the details of the mechanism. Instead, they must understand only how to build clients, remote objects, and their servers.

A few general factors to consider when building RMI-based applications include the following:

- Any local or remote method invocation can pass a reference to a remote object as an argument or can return a reference to a remote object as a result.
- A remote object can be cast to any of the remote interfaces supported by the implementation.
- Clients of remote objects interact with remote interfaces, not with the implementation classes of those interfaces.
- Non-remote arguments to and results from a remote method invocation are passed by copy rather than by reference, while remote objects are passed by reference.

- The semantics of some methods defined by class `Object` are specialized for remote objects.
- The failure modes of invoking remote objects are more complicated than those of invoking local objects, so clients must deal with additional exceptions that can occur during a remote method invocation.

The Client Side

The `java.rmi.Remote` interface serves to identify all remote objects. All client-side remote interfaces extend—either directly or indirectly—the remote interface, which defines no methods. All remote objects must directly or indirectly implement this interface. All remote interfaces must be declared public.

Any method in a remote interface must declare the `java.rmi.RemoteException` in its `throws` clause, along with any application-specific exceptions. That class is the superclass of all exceptions that can be thrown by the RMI system, and it distinguishes those exceptions that occur when a remote method invocation fails from other types of Java exceptions. For example, the RMI system throws a `RemoteException` when the network fails or the server for the method call cannot be reached.

The methods in a remote interface must pass remote objects—used either as an argument or a return value—as a remote interface, not as an implementation class.

Both the client and the server use the `java.rmi.Naming` class to interact with a name server. The default name server is the `rmiregistry`. The name server allows clients to acquire remote object references. Remote objects are identified using the standard Uniform Resource Locator syntax consisting of protocol, host, port, and name fields. The protocol used is `rmi`, although it can be omitted. The client uses the `lookup()` method to obtain a remote object reference from the name service.

Both the client and the server require a security manager. If the client is an applet, the security manager provided by the Web browser is adequate. The `java.rmi.RMISecurityManager` class is a simple security manager in the JDK that can be sufficient in many applications. It disables all functions except class definition and access, so that other classes for remote objects, their arguments, and returns can be loaded as needed. It allows a downloaded class to make a connection only if the connection is initiated via the RMI transport. An application can implement specialized security managers to replace the `RMISecurityManager`, but these are beyond the scope of this article.

The Server Side

The server-side implementation of a remote object must extend `java.rmi.server.RemoteObject` or its subclasses: `java.rmi.server.RemoteServer` and `java.rmi.server.UnicastRemoteObject`. The `RemoteObject` class provides the remote semantics of `java.Object` by overriding the `hashcode()`, `equals()`, and `toString()` methods in that class.

Both the client and the server use the java.rmi.Naming class to interact with a name server.

`RemoteServer` extends `RemoteObject` and provides abstract definitions for the methods needed to create objects and make them available remotely over a wide range of remote reference semantics. Subclasses of `RemoteServer` provide the concrete semantics of a remote reference, such as whether the remote service is a single or a replicated object.

Currently, `java.rmi.server.UnicastRemoteObject` is the only `RemoteServer` subclass supported by the standard RMI implementation. `UnicastRemoteObject` defines a non-replicated point-to-point remote object whose references are valid only while the server process supporting that object is running. It also assumes a TCP connection-based transport.

The remote object server also uses the `java.rmi.Naming` class to interact with a name server. The server uses the `bind()` or

rebind() methods to register as a server of remote objects with the name server.

The bind() method creates an association between a name and an object instance. The association can be undone using unbind(), but another bind() on the same name before an unbind() causes an exception.

Using rebind() essentially unbinds the old association and binds the new one. This is especially convenient for debugging, because it allows the name server to continue running without throwing exceptions. There are other server-side classes and interfaces that are beyond the scope of this article. See the RMI specification (http://www.java.sun.com/products/jdk/rmi/index.html) for additional details.

A Simple Example

In general, the steps for creating a distributed application using RMI include the following:

1. Define the remote interface.
2. Implement the remote interface.
3. Implement a server for the remote object.
4. Implement the client that uses the remote interface.
5. Create the stub and skeleton.
6. Start the HTTP servers that serve up the remote class files.
7. Start the name server.
8. Start the server of the remote object.
9. Run the client of the remote object.

These steps are illustrated with a simple example: a form of the ubiquitous "Hello World" used to prove a minimal level of capability in many situations. In the example, a client will use a remote object to return a string, then display the string. For simplicity, we will use only one physical machine running AIX 4.3.2 and using the JDK 1.1.6.

In Step 1, we define an interface called HelloTime, shown in Figure 3. HelloTime demonstrates the important characteristics of a remote interface: it is declared public ❶, which allows clients outside

```
package hello;
import java.rmi.*;

public ❶ interface HelloTime extends Remote { ❷
   String tellMe() throws RemoteException;
} ❸
```

Figure 3. The HelloTime remote interface

the package to access the interface, and it extends java.rmi.remote ❷ for the reasons described above. The method tellMe() throws a java.rmi.RemoteException,❸ as indicated above; in this case, the method tellMe() returns a String that is serialized as part of the Java language definition. We place the file HelloTime.java in $HOME/scr/rmi/hello.

In Step 2, we create an implementation of the HelloTime interface to run on the server. Figure 4 shows class HelloTimeImpl, which demonstrates the important characteristics of a remote object implementation: It must extend some subclass of java.rmi.server.❶ It must declare that it implements a remote interface.❷ It must define at least one constructor for the remote object.❸ It must also provide an implementation for all methods in the remote interfaces it implements.❹

The remote object implementation can define methods not in the remote interface, but such methods can be called only locally. We also place the file HelloTimeImpl.java in $HOME/scr/rmi/hello.

The tellMe() method implementation shows a slight twist on the standard "Hello World" demonstration. Instead of returning a constant string, it determines whether it is morning, afternoon, or evening in the current time zone, and responds appropriately.

In Step 3, the main() method of HelloTimeImpl provides the server for the remote object implementation. The server could be implemented entirely in another class, as long as it implements the important characteristics of a remote object server. The server must instantiate and install a security manager,❺ as discussed above. It must create at least one instance of the remote object.❻ The server must register all instances of the remote object with the name

```
package hello;
import java.util.*;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class HelloTimeImpl extends UnicastRemoteObject ❶
                    implements HelloTime ❷  {

  public HelloTimeImpl() ❸ throws RemoteException {
    super();
  }

  public String tellMe() throws RemoteException { ❹
    String time;
    // get the time
    Calendar cal = Calendar.getInstance();
    if (cal.get(Calendar.HOUR_OF_DAY) >= 18)
      time = "evening";
    else if (cal.get(Calendar.HOUR_OF_DAY) >= 12)
      time = "afternoon";
    else
      time = "morning";

    return ("Good " + time + ", World!");
  }

  public static void main(String args[]) {
    // set up security manager
    System.setSecurityManager(new RMISecurityManager()); ❺

    // create a remote object and register with the name server
    try {
      HelloTimeImpl ro = new HelloTimeImpl();' ❻
      Naming.rebind("//<hostname>/HelloTime", ro);' ❼
    } catch (Exception e) " { ❽
      System.out.println("Error: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

Figure 4. The HelloTimeImpl class

server; in this example, we use `rebind()` as an aid for debugging.❼ In the URL for the remote object, `<hostname>` should be replaced by the name of the machine upon which the example is run. Note the potential for remote exceptions that must be caught.❽

In Step 4, we implement a client that uses the remote interface. The client, shown in Figure 5, demonstrates the typical requirements for a client that uses a remote interface. The client instantiates and installs a security manager;❶ this is not necessary if the client is running as an applet because of the browser's security manager. The client gets a reference to the remote object from the name server.❷ The name used in the lookup operation must match that used in the bind or rebind operation on the server. The client must then cast the `Object` returned from the lookup method to the correct remote interface.❸ Finally, the client can invoke methods on the remote object.❹ Note the need to catch remote exceptions.❺ We also place the file `RemoteHello.java` in `$HOME/scr/rmi/hello`.

In Step 5, to create the stub and skeleton for the remote object, move to the directory `$HOME/scr/rmi`. Then run the RMI compiler on the remote object using the command `rmic hello.HelloTimeImpl`.

```
package hello;
import java.rmi.*;

public class RemoteHello {
  public static void main(String[] args) {
    // set up security manager
    System.setSecurityManager(new RMISecurityManager()); ❶

    // invoke the remote object
    try {
      HelloTime ri = (HelloTime) ❸
          Naming.lookup("//<hostname>/HelloTime"); ❷
      System.out.println(ri.tellMe()); ❹
    } catch (Exception e) { ❺
      System.out.println("Error: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

Figure 5. The RemoteHello class

This creates the stub and skeleton, `HelloTimeImpl_Stub.class` and `HelloTimeImpl_Skel.class`, respectively, in the directory `$HOME/scr/rmi`.

For Step 6, copy all relevant `.class` files to the directory from which the Web server will serve up the `.class` files. In this example, copy the following files to `$HOME/public_html/demo/hello` (this example assumes the Web server has been configured to serve files from the directory `public_html`):

- `$HOME/scr/rmi/HelloTimeImpl_Stub.class`
- `$HOME/scr/rmi/HelloTimeImpl_Skel.class`
- `$HOME/scr/rmi/hello/HelloTimeImpl.class`
- `$HOME/scr/rmi/hello/HelloTime.class`

Then, if it is not already started, start the Web server, using the appropriate command. Since this example only uses one machine, there is no need to start another Web server.

Step 7 uses the default name service, the `rmiregistry`. Start the name server using the command `rmiregistry`. Typically the name server is run in the background. Before starting the `rmiregistry`, make sure that no `CLASSPATH` environment variable is set or that `CLASSPATH` does not include the path of any classes that you want downloaded to your client, including the stub for the remote object implementation. If the `rmiregistry` can find the stub class from its `CLASSPATH`, it will ignore the server's `java.rmi.server.codebase` property and the client cannot download the stub code for the remote object.

For Step 8, start a new `dtterm` and change to the `$HOME/scr/rmi` directory. Either copy the stub and skeleton classes to the directory where server class resides (`$HOME/src/rmi/hello`) or make sure that the `$HOME/public_html/demo` directory is in `CLASSPATH`. Start the server for the remote object in this example using the following command:

```
java -Djava.rmi.server.codebase
=http://<hostname>/~<user>/demo/
hello.HelloTimeImpl
```

The option `-Djava.rmi.server.codebase` sets that property to the location of the `.class` files to be accessed by the server. For this example, the property corresponds to the `$HOME/public_html/demo/hello` directory. Note that the terminating slash "`/`" is important; without it, the `.class` files cannot be located.

In Step 9, start a new `dtterm` and change to the `$HOME/scr/rmi` directory. Run the

client of the remote object in this example using the following command:

```
java -Djava.rmi.server.codebase
=http://<hostname>/~<user>/demo/
hello.RemoteHello
```

As with the server, the option `-Djava.rmi.server.codebase` sets that property to the location of the `.class` files, in this case for the client. When you run the client, you should see the following, depending on the time of day: Good morning, World!, Good afternoon, World!, or Good evening, World!

Alternatives to RMI

Alternatives to RMI exist. For general communication, Java supports sockets. Socket-based communication, while quite flexible, requires the definition and implementation of application-specific protocols for message exchange. The definition of such protocols can be difficult, the implementation can be error-prone, and it is not consistent with object-oriented programming techniques.

Java IDL allows Java to work in the Object Management Group's Common Object Request Broker Architecture (CORBA). Since CORBA was designed to support a heterogeneous, multi-language environment, it has a language-neutral object model that can be used in the Java environment. Although the details are beyond the scope of this article, it is more work to use Java IDL than RMI because RMI assumes a homogeneous environment and can therefore take advantage of the Java object model.[2]

A final alternative, RMI over IIOP (RMI-IIOP) is perhaps the most interesting because it offers the simplicity of RMI and the heterogeneity of Java IDL. At the heart of RMI-IIOP, developed jointly by IBM and Sun Microsystems®, is the use of CORBA's Internet Inter-ORB Protocol (IIOP) instead of RMI's JRMP. This permits communication between Java objects and non-Java objects.

RMI-IIOP also includes a set of tools that permit programmers to create distributed applications with little additional effort beyond standard RMI, but reap the benefits of interoperability with non-Java objects.[3, 4]

A Remarkable Tool for Distributed Computing

The Java Remote Method Invocation is another important part of Java technology. RMI, as an extension of the Java object model, enables programmers to leverage important aspects of the Java environment, such as security, automatic garbage collection, and "write once, run anywhere" in distributed applications. RMI also allows programmers to perform tasks not possible with non-distributed applications, such as migrate function from client to server, or vice versa, and to do so very easily.

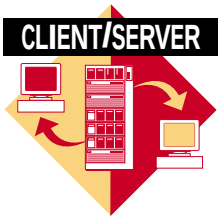
Extensions of RMI even allow distributed Java objects to communicate with non-Java objects. This makes RMI a remarkable tool for distributed computing and a valuable addition to a programmer's skill base.

***Greg Flurry**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Flurry is a senior technical staff member in the Software Solutions Division. His responsibilities, as part of the Software Group Technical Center, include developing and fostering IBM's e-Business technologies. He has a BS in Electrical Engineering from Vanderbilt University and an MS in the same field from the University of Kentucky.*

[2]For more information about CORBA's relationship to Java, see *Client/Server Programming with Java and CORBA* by Robert Orfali and Dan Harkey, John Wiley and Sons, 1997.

[3]Flurry, Greg. "RMI-IIOP—Enterprise-class Distributed Computing for Java." *AIXpert* (December 1999).

[4]Additional information on RMI-IIOP can be found on the Web at http://www.ibm.com/java/jdk/rmi-iiop/.



RMI-IIOP: Enterprise-class Distributed Computing for Java

By Greg Flurry

Java provides several options for creating distributed applications that offer different distribution capabilities and considerations during development. This article examines the latest option, RMI-IIOP, which offers the development-time simplicity of Java's RMI and the runtime interoperability of CORBA's IIOP. In addition, IIOP can propagate a transaction context and a security context between the client and server, which makes IIOP very attractive for enterprise-class applications and middleware.

Distributed Computing in Java

Java™ provides a range of options that allow programmers to implement various forms of distributed applications. These options include sockets, Java Remote Method Invocation, Java Interface Definition Language (IDL), and Remote Method Invocation over the Internet Inter-ORB Protocol (RMI-IIOP).

Sockets

Sockets provide a low-level, but ubiquitous technique for distributed computing. Sockets can be used for general communication purposes, but socket designs can be complex, are rarely consistent with object-oriented programming techniques, and are generally not reusable.

Sockets are available in Java Development Kit (JDK) 1.0 and later versions. Most general Java books offer some guidance for using sockets.[1]

Java Remote Method Invocation

Java Remote Method Invocation (RMI) allows programmers to create distributed applications in a manner consistent with traditional Java object-oriented programming techniques, thereby promoting reuse. RMI, based on the concepts of distributed Java objects, enables programmers to produce a distributed application with little additional effort beyond that required to produce a non-distributed application. For example, RMI allows passing local objects by value using Java serialization as well as passing remote objects by reference, which minimizes the differences between distributed and non-distributed applications. In addition, programmers simply define the remote interfaces used in the application and use the `rmic` tool to produce the additional classes required to support distributed operation.

Greg Flurry

[1]Harold, Elliott Rusty. *Java Network Programming*. O'Reilly and Associates, 1997.

RMI also supplies a simple name server, the `rmiregistry`, which supports running distributed applications. However, distributed applications implemented with RMI do have some limitations. RMI assumes a homogeneous distributed environment, so both the client and server objects must be written in Java. The standard RMI naming service supports only nonpersistent servers. Limitations also exist on the scalability and transactional characteristics of RMI-based applications because of its use of the simplistic Java Remote Method Protocol (JRMP).

RMI is available in JDK 1.1 and later versions. *Note:* See "Distributed Computing with Java RMI" by Greg Flurry in this issue of *AIXpert* for additional information about RMI and an example on AIX®.

Java IDL

Java IDL provides standards-based connectivity and interoperability in a heterogeneous distributed environment using the Common Object Request Broker Architecture (CORBA). CORBA, an open industry standard for heterogeneous distributed computing defined by the Object Management Group® (OMG®), allows objects to interoperate without any consideration of location, programming languages, operating systems, or any other system aspects that are not part of an object's interface.

Figure 1 shows a simplified view of the CORBA runtime environment. The client-side runtime includes two parts: an Object Request Broker (ORB) through which a client-side object intercommunicates with a server-side object, and a stub that is the client-side object's view of the server object and used by the client to interface with its ORB. The server-side runtime includes an ORB used to intercommunicate with the client-side object, an object adapter that instantiates the server-side object and provides it an identity, and a skeleton, the server-side object used to interface to its ORB.

The ORBs interact using the Internet Inter-ORB Protocol (IIOP). To use CORBA, a programmer must define remote network

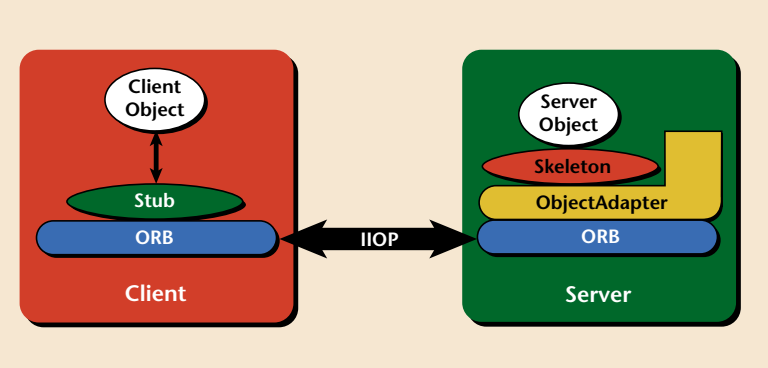


Figure 1. CORBA runtime architecture

services using the OMG's Interface Definition Language. The actual object implementations can be in any of the many languages supported by CORBA.

CORBA also defines several services available to CORBA-compliant objects. One of the most important is a name server with which objects can locate other objects. Additional information about Java IDL and CORBA is available at http://java.sun.com/products/jdk/1.2/docs/guide/idl/index.html and other references.[2]

Java IDL allows Java objects to participate in a heterogeneous CORBA environment. Java IDL includes the `idltojava` compiler that generates, from IDL, portable client stubs and server skeletons in Java. These stubs and skeletons work with any CORBA-compliant ORB implementation. Java IDL also includes a CORBA-compliant ORB that interacts with other CORBA-compliant ORBs using IIOP and a CORBA-compliant name server, `tnameserv`. The name server offers capabilities similar to RMI's `rmiregistry` in that it supports only nonpersistent objects and does not scale well. See http://java.sun.com/products/jdk/idl/index.html and http://java.sun.com/docs/books/tutorial/idl/index.html for additional information.

Java IDL offers solutions to some problems with RMI. It clearly allows Java objects to use services implemented in other languages, which can be especially important for accessing legacy applications. CORBA offers enterprise-class robustness, scaling,

[2] Orfali, Robert and Harkey, Dan. *Client/Server Programming with Java and CORBA*. Wiley & Sons, 1997.

security, and transactional capabilities. The tool `tnameserv` can be replaced with a full implementation of the CORBA naming services that supports nonpersistent servers, increasing the efficiency of the runtime environment.

Java IDL also suffers from some drawbacks. Programmers must define interfaces in IDL rather than Java, which introduces extra work during the construction of applications. Also, IDL does not support passing parameters by value, which introduces some inconsistencies in the Java IDL programming paradigm compared to non-distributed (or RMI) Java applications.

Java IDL is available as part of Enterprise Java in Java 2 (also known as JDK 1.2). The JDK includes the ORB and `tnameserv`. Currently, `idltojava` must be downloaded separately from http://developer.java.sun.com/developer/earlyAccess/jdk12/idltojava.html. Although Java 2 is not yet supported on AIX, it is available for a preview on AIX and for downloading at IBM's alphaWorks™ site (http://www.alphaWorks.ibm.com).

RMI-IIOP

RMI-IIOP (pronounced RMI over IIOP) is the latest option for distributed computing in Java. Jointly developed by IBM and Sun Microsystems®, RMI-IIOP offers the usability of RMI and the interoperability of CORBA, giving distributed application developers the best of both worlds.

RMI-IIOP is an improved version of RMI that, like standard RMI, supports defining object interfaces in Java rather than IDL, and parameter passing consistent with non-distributed Java practices. Like Java IDL, however, RMI-IIOP permits Java objects to interoperate with CORBA ORBs and CORBA-compatible objects implemented in other languages.

RMI-IIOP builds on recently adopted CORBA standards for passing objects by value through IIOP and for mapping Java to IDL, and works with any CORBA ORB that supports these recent standards. It does require slight modifications to the RMI programming application programming interfaces (APIs), but as a result, a single RMI programming model operates with equivalent semantics over both JRMP and IIOP.

RMI-IIOP includes a new version of the `rmic` compiler that can generate IIOP-compatible stubs and skeletons (called *ties* in RMI-IIOP documentation). It can also produce IDL from the input Java interfaces. RMI-IIOP also provides a new `idltojava` compiler, which supports the new CORBA Objects-By-Value standard required for interoperation with RMI-IIOP. RMI-IIOP supports a CORBA-compliant ORB and CORBA-compliant name server, `tnameserv`.

RMI-IIOP is currently available only in beta form for JDK 1.1.6 and above. A beta version for Microsoft® Windows® and Sun Solaris® is available to download from http://www.ibm.com/java/jdk/rmi-iiop/. The beta form for AIX is not yet available. The beta download includes extensive documentation on the RMI-IIOP, tools, and APIs.

RMI-IIOP vs. RMI

The steps for creating a distributed application using RMI-IIOP are very similar to the steps for creating a plain RMI application.[3] Those steps are repeated here, along with the differences between RMI and RMI-IIOP programming.

1. Define the remote interface. This step is exactly the same because the interface can be defined in Java.

2. Implement the remote interface. The remote implementation class must now inherit from `javax.rmi.PortableRemoteObject`, or explicitly export implementation objects after creation by calling `PortableRemoteObject.exportObject()`.

3. Implement a server for the remote object. The server needs a security manager, as with RMI. The application must use Java Naming and Directory Interface (JNDI) through the CORBA-compliant name server plug in, which means using the `javax.naming.*` package. If CORBA naming is not appropriate, then you

[3]Flurry, Greg. "Distributed Computing with Java Remote Method Invocation." *AIXpert* (December 1999).

must use other techniques. RMI's `rmiregistry` and associated access methods cannot be used.

4. Implement the client that uses the remote interface. The client must still invoke a security manager (unless it is an applet). The client must also make the changes for the naming service outlined in Step 3. In addition, instead of simply casting the result of looking up a remote object to the remote interface, the client must use `javax.rmi.PortableRemoteObject.narrow()`, the mechanism that checks to ensure the RMI-IIOP object can be cast to the desired remote interface type.
5. Create the stub and skeleton. It is now necessary to use the new RMI-IIOP `rmic` command with the `-iiop` option to produce the client-side stub and server-side tie (skeleton) class files.
6. Start the HTTP servers that serve up the remote class files. This step is the same when creating the plain RMI application.
7. Start the name server. For RMI-IIOP, you must use the `tnameserv` instead of the `rmiregistry`, or another appropriate name server mechanism.
8. Start the server for the remote object. When starting the server and running the client, use a different set of system properties on the `java` command. Figure 2 shows the properties that indicate the class of the name server factory and the URL of the name server.
9. Run the client of the remote object. See the description of the previous step.

Restrictions for RMI Applications Running over IIOP

RMI-IIOP applications have several restrictions:

- All constant definitions in remote interfaces must be primitive types or `String` and evaluated at compile time.
- Applications cannot use Java names that conflict with the "mangled" names generated by the Java-to-IDL mapping rules from the OMG's Java Language-to-IDL Mapping specification.
- Classes cannot inherit the same method name into a remote interface more than once from different base remote interfaces.
- Using names that differ only in case is generally not sufficient to distinguish methods or objects.
- It is not possible to share an "object reference" object across IIOP. Sharing other objects works correctly.
- Applications cannot use the following features of RMI: `RMISocketFactory`, `UnicastRemoteObject`, and `Unreferenced`.
- Applications cannot depend on RMI's distributed garbage collection, nor can an RMI-IIOP server use any RMI interfaces associated with distributed garbage collection.

For the vast majority of applications, none of these restrictions present real problems.

An RMI-IIOP Example

The following example converts the RMI example in "Distributed Computing with Java RMI" (this issue of *AIXpert*) to RMI-IIOP.

Installing and Setting Up RMI-IIOP

To follow the example, you must first install the RMI-IIOP code. As this article goes to press, RMI-IIOP is not yet available for AIX, so we will use Microsoft Windows, which also illustrates the portability of Java applications. You can download the beta version of RMI-IIOP for Microsoft Windows from http://www.ibm.com/java/jdk/rmi-iiop/. You must have installed JDK 1.1.6 or above to run the RMI-IIOP environment.

```
-Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
-Djava.naming.provider.url=iiop://<hostname>:<port>
```

Figure 2. Properties of the name server

Follow the instructions on the download page to unzip the downloaded file and execute the installer application, `rmiiiopbeta`. The installer deposits all the documentation, jar, sample, and executable files in the chosen directory (the rest of the example assumes the directory is `c:\rmi-iiop-beta`). You should then browse `file:///c:/rmi-iiop-beta/readme.html` for setup instructions regarding environment variables.

We found a problem with the `c:\rmi-iiop-beta\rmi-iiop.bat` file used to establish the right environment for building and running RMI-IIOP applications. The batch file has two branch target names that are too long; they must be shortened to eight characters or less, and of course, remain unique.

After modifying `rmi-iiop.bat` as described above, you can run it. Then run `checkme.bat` in the same DOS shell to confirm the installation and setup. Assuming everything checks out correctly, you can proceed to the next steps.

Modifying an RMI Application to Work with RMI-IIOP

The RMI example from the reference is a variation of the simple, but ubiquitous, "Hello World" used to demonstrate many technologies. Figure 3 shows the original remote interface for the remote object; it does not need to change to port the application to RMI-IIOP.

Figure 4 shows the original implementation of the remote object and its server for an RMI environment. Figure 5 shows the implementation of the remote object and server for the RMI-IIOP environment, with the differences highlighted. Note that the new implementation of `HelloTimeImpl` imports the CORBA-compliant `javax.rmi.server.PortableRemoteObject` ❶ rather than RMI `javax.rmi.server.UnicastRemoteObject`; it also extends `PortableRemoteObject` ❷ rather than `UnicastRemoteObject`.

To use the CORBA-compliant JNDI naming service rather than the `rmiregistry`, the new `HelloTimeImpl` server imports the

```
package hello;
import java.rmi.*;

public interface HelloTime extends Remote {
    String tellMe() throws RemoteException;
}
```

Figure 3. The HelloTime remote interface for both RMI and RMI-IIOP

```
package hello;
import java.util.*;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;

public class HelloTimeImpl extends UnicastRemoteObject
                    implements HelloTime {

  public HelloTimeImpl() throws RemoteException {
    super();
  }

  public String tellMe() throws RemoteException {
    String time;

    // get the time
    Calendar cal = Calendar.getInstance();
    if (cal.get(Calendar.HOUR_OF_DAY) >= 18)
      time = "evening";
    else if (cal.get(Calendar.HOUR_OF_DAY) >= 12)
      time = "afternoon";
    else
      time = "morning";
```

(continued on following page)

Figure 4. The HelloTimeImpl class for RMI

(continued from previous page)

```
    return ("Good " + time + ", World!");
  }

  public static void main(String args[]) {

    // set up security manager

    System.setSecurityManager(new RMISecurityManager());

    // create a remote object and register with the naming service
    try {
      HelloTimeImpl ro = new HelloTimeImpl();
      Naming.rebind("//<hostname>/HelloTime", ro);
    } catch (Exception e) {
      System.out.println("Error: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

Figure 4. The HelloTimeImpl class for RMI

```
package hello;
import java.util.*;
import java.rmi.*;
import javax.rmi.PortableRemoteObject; ❶
import javax.naming.*; ❸

public class HelloTimeImpl extends PortableRemoteObject ❷
                    implements HelloTime {

  public HelloTimeImpl() throws RemoteException {
    super();
  }

  public String tellMe() throws RemoteException {
    String end;

    // get the time
    Calendar cal = Calendar.getInstance();
    if (cal.get(Calendar.HOUR_OF_DAY) >= 18)
      time = "evening";
    else if (cal.get(Calendar.HOUR_OF_DAY) >= 12)
      time = "afternoon";
    else
      time = "morning";
    return ("Good " + time + ", World!");
  }

  public static void main(String args[]) {

    Context iNC = null; ❹
```

(continued on following page)

Figure 5. The HelloTimeImpl class for RMI-IIOP

(continued from previous page)

```
    // fire up the JNDI name server
    try {
      iNC = new InitialContext(); ❹
    } catch (Exception e) {
      System.out.println("Error: " + e.getMessage());
      e.printStackTrace();
    }

    // set up security manager
    System.setSecurityManager(new RMISecurityManager());

    // create a remote object and register with the naming service
    try {

      HelloTimeImpl ro = new HelloTimeImpl();

      iNC.rebind( ❺
        "iiop://<hostname>/HelloTime", ro); ❻

      System.out.println("HelloTime bound");
    } catch (Exception e) {
      System.out.println("Error: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

Figure 5. The HelloTimeImpl class for RMI-IIOP

JNDI naming package `javax.naming.*` ❸, then creates an initial naming context ❹ and uses that naming context to register the remote object instance with the name server ❺ . Note also that `iiop` ❻ is used as the protocol for the URL of the remote object.

Figure 6 shows the original client of the remote object for an RMI environment. Figure 7 shows the implementation of the client of the remote object for the RMI-IIOP environment, with the differences highlighted. Note that the new implementation of `RemoteHello` imports the CORBA-compliant `javax.rmi.server.PortableRemoteObject` ❶ rather than RMI `javax.rmi.server.UnicastRemoteObject`; it also replaces the simple cast of the remote interface with a call to `javax.rmi.PortableRemoteObject.narrow()` ❷ .

To use the CORBA-compliant JNDI naming service rather than the `rmiregistry`, the new `RemoteHello` imports the JNDI naming package `javax.naming.*` ❸, creates an initial naming context ❹, and uses that naming context to look up the remote object on the name server ❺. Note again that `iiop` ❻ is used as the protocol for the URL of the remote object.

Once the Java files in Figures 5 and 7 are compiled, the first four steps in creating an RMI-IIOP application are complete. For Step 5—creating the stub and tie for the remote object—run the new RMI compiler on the remote object using the command `rmic -iiop hello.HelloTimeImpl`. This creates the stub and tie: `HelloTimeImpl_Stub.class` and `HelloTimeImpl_Tie.class`.

In Step 6, copy all of the relevant `.class` files to the directory from which the Web server will serve up the `.class` files, then start the Web server. For Step 7 (assuming you wish to use the RMI-IIOP default name server), start the name server using the command `tnameserv`. Run the name server in the background.

For Step 8, begin a new DOS shell and start the server for the remote object using the command in Figure 8. During testing, we found that the DOS shell would not

```
package hello;
import java.rmi.*;

public class RemoteHello {

  public static void main(String[] args) {
    // set up security manager
    System.setSecurityManager(new RMISecurityManager());

    // invoke the remote object
    try {
      HelloTime ri =
          (HelloTime)Naming.lookup("//<hostname>/HelloTime");
      System.out.println(ri.tellMe());
    } catch (Exception e) {
      System.out.println("Error: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

Figure 6. The RemoteHello class for RMI

```
package hello;
import java.rmi.*;
import javax.rmi.PortableRemoteObject; ❶
import javax.naming.*; ❸

public class RemoteHello {

  public static void main(String[] args) {
    // set up security manager
    System.setSecurityManager(new RMISecurityManager());

    Context iNC = null; ❹

    // access the JNDI name server
    try {
      iNC = new InitialContext(); ❹
    } catch (Exception e) {
       System.out.println("Error: " + e.getMessage());
       e.printStackTrace();
    }

    // invoke the remote object
    try {
      HelloTime ri = (HelloTime) PortableRemoteObject.narrow( ❷
         iNC.lookup( ❺
         "iiop://<hostname>/HelloTime"), ❻
         HelloTime.class);
    System.out.println(ri.tellMe());
    } catch (Exception e) {
      System.out.println("Error: " + e.getMessage());
      e.printStackTrace();
    }
  }
}
```

Figure 7. The RemoteHello class for RMI-IIOP

```
java -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
     -Djava.naming.provider.url=iiop://<hostname>
     hello.HelloTimeImpl
```

Figure 8. Command to start the remote server

```
java -Djava.naming.factory.initial=com.sun.jndi.cosnaming.CNCtxFactory
     -Djava.naming.provider.url=iiop://<hostname>
     hello.RemoteHello
```

Figure 9. Command to run the client of the remote object

accept the complete command, so we created a batch file containing the command.

For Step 9, start a new DOS shell and run the client of the remote object using the command in Figure 9. As with Step 8, we had to create a batch file for the command.

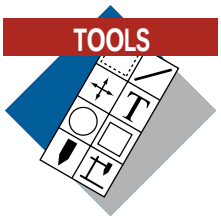
When you run the client, you should see "Good morning, World!" or "Good afternoon, World!" or "Good evening, World!," depending on the time of day.

The Future of RMI-IIOP

RMI-IIOP offers the development-time simplicity of Java's RMI and the runtime interoperability of CORBA's IIOP. In addition, IIOP, unlike RMI's simple JRMP, can propagate a transaction context and a security context between the client and server. This makes IIOP very attractive for enterprise-class applications and middleware—the reason that IBM and Sun chose to make IIOP a part of the Enterprise JavaBeans™ specification.

Enterprise JavaBeans are the foundation of Sun's upcoming Java 2 Enterprise Edition, which is expected to provide the environment for the development and deployment of cross-platform enterprise-class applications in the next decade. RMI-IIOP is at the heart of this upcoming revolution in enterprise-class application development and deployment.

***Greg Flurry**, IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Mr. Flurry is a senior technical staff member in the Software Solutions Division. His responsibilities, as part of the Software Group Technical Center, include developing and fostering IBM's e-business technologies. He has a BS in Electrical Engineering from Vanderbilt University and an MS in the same field from the University of Kentucky.*



Native AIX Applications from Windows Source Code

By Marius van Handel

MainWin allows you to recompile Windows source code to create native AIX applications. No development is required. Instead of taking months to write code, you can rehost existing Windows code on AIX (and other UNIX platforms) in days. MainWin is not an emulator: MainWin-rehosted applications are native AIX applications with native AIX performance and full Windows functionality. This article describes MainWin for AIX.

MainWin allows you to create native AIX® applications from Windows® source code. When your Windows application is ready, you take the application source code, move it to your AIX host where MainWin is installed, and use MainWin SDK and the AIX C/C++ compiler to rebuild the application on AIX. You can then test and debug it; and when you are happy with the results, package your application with MainWin Run-Time and release it.

MainWin consists of two parts:

MainWin SDK—provides all the tools required to rebuild a Windows C/C++ application on AIX. The SDK includes the Windows NT® 4 system header files and a set of Windows build tools, such as the Windows Resource Compiler, the MIDL Compiler for DCOM, and the Message Compiler, which were rehosted by Mainsoft to AIX. The MainWin SDK also includes an automatic makefile generator to help you set up the application build on AIX. Mainsoft's access to Windows NT 4 source code guarantees the authenticity of both the Windows system header files and the build tools provided in the MainWin SDK.

MainWin Run-Time—provides the Windows system dynamic link libraries (DLLs) and services necessary for Windows applications to execute successfully on AIX. MainWin Run-Time implements basic Windows functionality, such as the Win32 subsystem (User, GDI, and Kernel) and extended Windows system services, such as WGL, MFC, DCOM, ATL, OLE/ActiveX™, HTML, Dynamic HTML, HTML-Help, WinSock, WinInet, URLMon, Common Dialogs, Common Controls, RichEdit, Shell32, and advapi.dll. MainWin Run-Time also includes Microsoft's HTML-Help and WinHelp as AIX executables. They allow you to read and display unmodified Windows HTML-Help and WinHelp files on AIX.

MainWin offers both Motif® and Windows look-and-feel (or "GUI mode") for the AIX version of your application, giving you the freedom to choose the GUI mode most appropriate for the end user and the application. The Motif look is generated by MainWin. To ensure maximum Windows compliance and optimum performance,

Marius van Handel

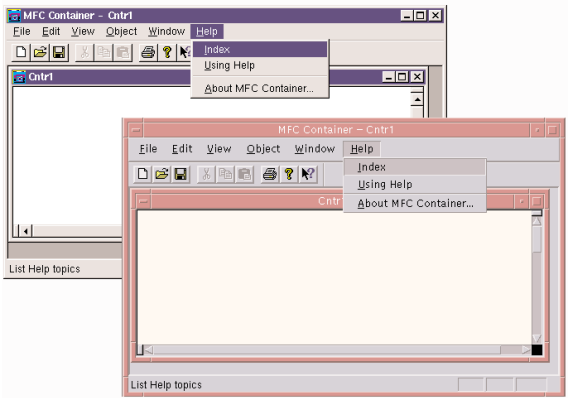


Figure 1. The same rehosted application: Motif look and Windows look

MainWin is completely independent of Motif and Xtoolkit.

How MainWin Works

Taking advantage of our source-code licensing agreement with Microsoft®, Mainsoft has spent more than 120 staff-years rewriting the operating-system implementation layer of the Win32 API for UNIX®. The result is a full Windows on AIX. Your program code does not even need to know that you are running on AIX.

In addition to the Win32 API, MainWin brings the latest Microsoft extended services, such as MFC, ATL, OLE/ActiveX, WGL, DCOM/COM, WinInet, Dynamic HTML, Common Controls, and Common Dialogs to AIX.

Figure 2 compares the Windows architecture on Windows to the MainWin Windows architecture on AIX.

The MainWin Run-Time

The MainWin Windows on AIX runtime comprises the Win32 subsystem and the Windows Extended Services. These two layers sit directly on top of the AIX operating system and provide the runtime support required for applications to run on AIX systems with full Windows functionality. MainWin Run-Time is distributed with the AIX version of your application and will reside on the end-user's computer.

Figure 3 is a schematic view of the MainWin Windows runtime on AIX.

MainWin Win32 Subsystem

The Win32 subsystem is a low-level implementation of the Windows 32-bit interface (Win32 API) on AIX. This thin and efficient layer sits close to the low-level AIX service layers such as the kernel, Xlib, and OpenGL™. While MainWin's Win32 API consists of a large amount of Windows source code, the operating-system-dependent implementation layer has been rewritten by Mainsoft for AIX.

MainWin Windows Extended Services

The Windows Extended Services, which consists of millions of lines of Windows NT 4 source code, has been rehosted on AIX. The MainWin Win32 layer has allowed us to rehost large portions of Windows NT 4 runtime support with minimal code modifications. Having the actual Windows source code running on AIX assures you of the highest level of Windows compatibility for your applications, and allows the same source code to run correctly on both UNIX and Windows. These Windows services enable AIX developers to develop in the Windows environment and deploy on AIX.

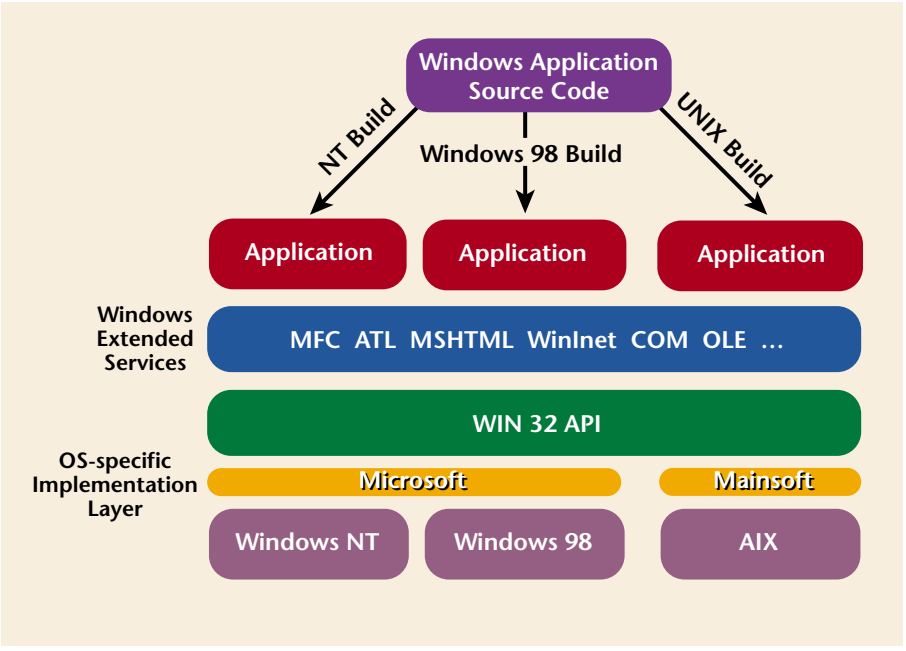


Figure 2. Windows on Windows and Windows on AIX

MainWin: True Windows on AIX

MainWin delivers the exact Windows look-and-feel, Windows event loop, Windows bitmap representation, Windows Region management and clipping algorithms, Windows Resource Compiler, MIDL compiler, and Windows messaging.

As a result of this Windows-compliant architecture, applications rehosted on AIX using MainWin get to market quickly, with a minimum of code modification. Our historical experience has been that porting an application from Windows NT to AIX via MainWin requires changing, on the average, only 0.01–0.1% of source code. When we rehosted Microsoft's Visual SourceSafe to AIX, we needed to modify less than 200 out of 400,000 lines (less than 0.05%) of source code.

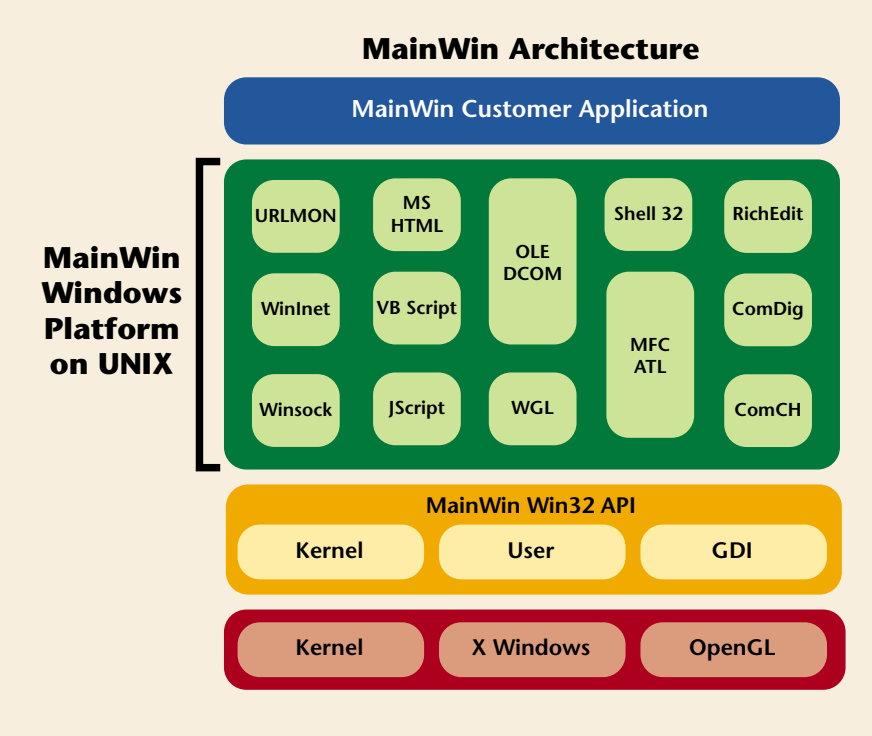


Figure 3. MainWin Run-Time architecture

Performance

Since MainWin is a thin layer sitting directly on top of Xlib, MainWin-rehosted applications perform better than ported applications relying on toolkits or other translation techniques. In fact, applications rehosted by MainWin are native AIX applications that take full advantage of the performance, scalability, and robustness of IBM's AIX operating system.

> ***MainWin offers both Motif and Windows look-and-feel for the AIX version of your application, giving you the freedom to choose the GUI mode most appropriate for the end user and the application.***

The Microsoft Internet Explorer 4.0 team ran benchmark tests on identical UNIX systems with 64 MB RAM to compare the performance of Internet Explorer 4.0 with Netscape Communicator® on UNIX. The results showed that Internet Explorer using the MainWin technology infrastructure responds faster than Netscape Communicator using the native Motif API.

Product Stability and Quality

Because a MainWin-rehosted application is supported by millions of lines of Microsoft source code rehosted on AIX by Mainsoft, code will run on AIX as it does on Windows.

Mainsoft has the rights to use the test suites developed by Microsoft to test Windows NT and MFC. Microsoft has invested an estimated 300 staff years in developing this test suites. Mainsoft is leveraging Microsoft's intellectual investment in testing tools by using the same tools to test MainWin. MainWin is tested with the same tests: RATS (to test Win32 API), CTOLECOM (to test DCOM), MSQA (to test MFC), assuring the highest possible code quality and Windows compliance.

Microsoft chose to use MainWin to rehost Internet Explorer 4 and 5 on UNIX rather than rewriting the code. Internet Explorer is a very large, complex, and technically advanced application that demonstrates MainWin's stability, performance,

MainWin allows you to deliver applications on Windows, IBM AIX, and other UNIX platforms simultaneously.

and technologies. Internet Explorer makes extensive use of the Windows API and the latest Microsoft advanced technologies, including Internet and HTML-related technologies. It uses many more Windows APIs than typical applications.

Despite its technical complexity, Microsoft makes Internet Explorer for UNIX freely available for downloading, an indication of the reliability and robustness of MainWin.

Internet Explorer and Outlook Express demonstrate the smooth integration of MainWin-based applications with the native X environment, the Windows Manager, and with other X/Motif applications. For instance, you can cut and paste freely between Internet Explorer and Motif applications, an indication that MainWin applications are good citizens of the UNIX desktop.

Among the hundreds of Mainsoft customers who rely on MainWin for their mission-critical organization-wide applications, Computer Associates chose MainWin to rehost its next-generation enterprise and information management solutions. Many other Fortune 500 companies are MainWin customers including Cadence, Landmark Graphics, Lucent, Synopsys, Bridge, and Ford Motor Company.

Windows on AIX

MainWin allows you to deliver applications on Windows, IBM AIX, and other UNIX platforms simultaneously. Using MainWin costs a very small fraction of the time and money it would take to rewrite Windows code for AIX. And MainWin provides native AIX performance with full Windows functionality. MainWin is the answer for cross-developing on Windows and AIX simultaneously.

For more information, visit http://www.mainsoft.com.

Marius van Handel, *Mainsoft Corporation, Ofek 1 Bldg. 2, North Industrial Area, Lod 71293, Israel. e-mail: mailto://marius@mainsoft.com.*

Constructing Index Classes and Attributes within the DB2 Digital Library

By Bill Baer

The IBM DB2 Digital Library enables multimedia data to be stored in a relational database. This article describes how to programmatically create the basic structures of the Digital Library: the index class and attribute. A GUI system administration tool included with the DL can be used to manually create index classes and attributes. But if you want to replicate the structures (such as between a test system and production system), it can be very time consuming to do manually. Also, if some index classes have common attributes, it can save time to copy/paste the code programmatically to make copies.

Digital media may include text, full-motion video, audio, graphics, and images. For this kind of data, the IBM DB2® Digital Library (DL) integrates information capture, storage management, search and retrieval, and secure distribution into a single offering. It provides sufficient storage capacity, scalability, speed, multi-level security, sophisticated searching techniques, and Internet access to enable content owners to reach new markets, preserve assets from loss or deterioration, and ensure copyright protection.

Digital Library Terminology

Within DB2 DL is a data repository for video, audio, image, and text files and their related metadata. The metadata is stored within the DB2 relational database and the files are stored within a file server. DL manages links between rows of a relational table and files within a file server. The infrastructure necessary for managing these links lies within the relational tables and columns.

Within DL, a relational table is referred to as an *index class* and a column is referred to as an *attribute*. In addition to these name variations, DL index classes and attributes have some subtle behavioral differences as compared to their relational database counterparts. For example, an attribute can be reused across multiple index classes, whereas a column must be redefined for each table.

A Step-by-Step Approach

The concepts and source code presented in this article cover the process of constructing both attributes and index classes within DL. This process is broken down into seven steps, and source code for accomplishing each step

Bill Baer

is provided. Upon successful completion of all these steps, an attribute, MyVarChar, and index class, MyIndexClass, are defined. Furthermore, the index class, MyIndexClass, contains one attribute, MyVarChar. Lastly, the function, GetAttrIDFromName, shows a helpful and reusable algorithm for obtaining the attribute ID provided by the attribute name.

All of the function calls required to programmatically set up the database structures are contained within the Folder Manager (FM) application programming interface (API). The following list indicates those seven calls:

- SimLibLogon(pszDBName, pszApplicationName, pszUserID, pszPassword, pszNewPassword, pszProxyID, pszProxyScope, fSession, pAsyncCtl, pRC)
- Ip2CreateAttr(hSession, pAttrInfo, pAsyncCtl, pRC)
- Ip2CreateClass(hSession, pClassInfo, usIndexClass, usACL, pAsyncCtl, pRC)
- SimLibCreateItem(hSession, usItemType, usIndexClass, usNumOfAttrs, pAttributeList, ulAccessControl, pAsyncCtl, pRC)
- Ip2ListAttrs(hSession, pAsynCtl, pRC)
- SimLibFree(hSession, PVOID, pRC)
- SimLibLogoff(hSession, pAsyncCtl, pRC)

These functions are defined in the *DL Application Programming Reference (SC26-8652-02)*. The SimLibLogon and SimLibLogoff functions log on and log off to DL respectively. The session handle returned by the SimLibLogon call is the connection used by all successive calls to DL.

The Ip2CreateAttr and Ip2CreateClass functions construct the attribute and index class respectively. The Ip2CreateClass function requires all attributes necessary for the construction of the index class to be constructed prior to calling this function. Therefore, the Ip2CreateAttr function must be called for each attribute used by the index class prior to constructing the index class. The SimLibCreateItem and Ip2ListAttrs are utility functions used in the index class creation process. Their use is further described later.

Creating Index Classes and Attributes

This article describes a seven-step process to create index classes and attributes. To begin any process with DL, you obtain a session handle. Next, you must first create the attributes before creating the actual index classes (the class contains the attribute).

The process of creating the attribute is done in two steps. First, you define the data structures that will be used to call the function to construct the attributes. The second step is to call the function that performs the creation. You repeat these two steps for all the attributes you wish to define prior to creating an index class.

Next, you perform another two-step process to first define the data structures that will be used to construct the index class and create them. Then there are tasks that enable the DL system administration tool to see the index classes and attributes that were created programmatically.

Step 1: Log on to DL

Each DL application must have a session connection. This session is acquired through another FM function call, SimLibLogon. This function should be called at the beginning of any DL application. Upon successful completion of this call, a valid session handle will be established. In the code snippet in Figure 1, the variable hSession contains this handle, so the function knows the context of the call.

Step 2: Define Data Structures Used to Create Attributes

Prior to constructing an index class, all attributes referenced by the index class must be constructed. The process of constructing an attribute is broken down into Steps 2 and 3. These two steps should be

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define FRN_INCL_FI
#define FRN_INCL_FO
#include <frnp.h>
#include <frnpcapi.h>
#define FRNINDX_NUM_ATTR 21
#define FRNVIEW_NUM_ATTR 7
int main(void) {
ULONG ulRC; /*Return Code*/
HSESSION hSession; /*Session Handle*/
PUSERLOGONINFOSTRUCT pUserLogonInfo; /*User logon info struct*/
PSZ   pszDBName = (unsigned char*) "LIBSRVRX"; /*Pointer to database name*/
PSZ pszUserId = (unsigned char*) "FRNADMIN"; /*Pointer to Userid*/
PSZ pszPassword = (unsigned char*) "PASSWORD" /*Pointer to Password*/
BITS fSessionType; /*FM Session Type*/
RCSTRUCT RC; /*RC's data structure*/
int rc = 0;
fSessionType = SIM_SS_CONFIG;
/*********************************************************************/
/* Call SimLibLogon to establish a FM session. */
/* If successful, save the FM session handle for subsequent */
/* FM API calls. */
/*********************************************************************/
ulRC = SimLibLogon( (PSZ) pszDBName, /* Pointer to Database name */
       (PSZ) NULL, /* Pointer to Application Name */
       (PSZ) pszUserId, /* Pointer to User Id */
       (PSZ) pszPassword, /* Pointer to Password for User Id */
       (PSZ) NULL, /* New Password */
       (PSZ) NULL, /* Logon without proxy */
       (PSZ) NULL, /* Logon without proxy scope */
fSessionType, /* Non-configuration session logon */
(PASYNCCTLSTRUCT) NULL, /* Request synchronous processing */
(PRCSTRUCT) &RC ); /* Pointer to RC data structure */
if (ulRC == SIM_RC_OK)
{
hSession = (HSESSION) RC.ulParam1; /*Session handle and user info returned through*/
pUserLogonInfo = (PUSERLOGONINFOSTRUCT) RC.ulParam2; /*return structure*/
}
else
{
printf("SimLibLogon error ulRC = %ld \n", ulRC);
return(1);
}
//Place Step Two Code Here
//Place Step Three Code Here
//Place Step Four Code Here
//Place Step Five Code Here
//Place Step Six Code Here
//Place Step Seven Code Here
return(0); /*Return 0 from main*/
}
```

Figure 1. Sample main program for logging on to DL

repeated for each attribute to be constructed.

Before calling the Ip2CreateAttr function, the ATTRINFOSTRUCT data structure must be defined. This data structure contains the definition of the attribute to be constructed. Datatype, maximum/minimum length, and NLS properties are stored within this structure. Figure 2 contains source code for defining an alphanumeric variable character string with the name MyVarChar. The minimum length of the string is 1 and the maximum length of the string is 150.

The ATTRINFOSTRUCT, defined in Figure 2, contains five fields: lMin, lMax, fTypeFlags, usAttrType, and szAttributeName. Since attributes are simply columns within DB2, the valid datatypes and domains for attributes follow the DB2 model. The entire set of DB2 column types can be constructed by placing different combinations of these five fields together. The definitions of these fields are shown below.

- **Major type (usAttrType):** Defines which of the relational database types this attribute will take on. Valid values for this field are SIM_ATTR_DATE, SIM_ATTR_DECIMAL, SIM_ATTR_FSTRING, SIM_ATTR_LONG, SIM_ATTR_SHORT, SIM_ATTR_TIME, SIM_ATTR_TIMESTAMP, and SIM_ATTR_VSTRING.
- **Minor type (fTypeFlags):** Qualifies the datatype of the attribute. Valid values for this field are SIM_ATTR_ALPHA, SIM_ATTR_ALPHANUM, SIM_ATTR_ALPHANUM_EXT, SIM_ATTR_NONE, SIM_ATTR_NUMERIC, and SIM_ATTR_OTHER.
- **lMin and lMax:** The use of these fields depends on the values specified in the usAttrType and fTypeFlags. In some cases, lMin and lMax specify the minimum/maximum length of the field (such as variable characters). In other situations, these properties specify the minimum/maximum values the field can contain (such as integers, long, and so on). Consult the *DL Application Programming Reference* for the lMin/lMax rules of use.

Step 3: Create the Attribute

Once the PATTRINFOSTRUCT, defined in Figure 2, has been filled out, the process of calling the Ip2CreateAttr is fairly straightforward. In the code snippet in Figure 3, the return code from Ip2CreateAttr is checked. If it succeeds, a call to SimLibFree is made. The SimLibFree function is used to free up memory acquired by FM. In this case, the return code structure contains the memory that needs to be freed.

```
ATTRINFOSTRUCT pAttrInfo;
memset(&pAttrInfo, 0, sizeof(ATTRINFOSTRUCT));
(pAttrInfo).ulStruct = sizeof(ATTRINFOSTRUCT);
(pAttrInfo).fUseBidirectional = (BOOL) FALSE;
(pAttrInfo).fSymmetricSwapping = (BOOL) FALSE;
(pAttrInfo).fShaping = (BOOL) FALSE;
(pAttrInfo).lMin = 1; /*Minimum length of VarChar*/
(pAttrInfo).lMax = 150; /*Maximum length of VarChar*/
(pAttrInfo).fTypeFlags = SIM_ATTR_ALPHANUM; /*The VarChar is alphanumeric*/
(pAttrInfo).usAttrType = SIM_ATTR_VSTRING; /*This is a variable length character string*/
(pAttrInfo).usHorizontalOrientation = SIM_ATTR_LEFT_TO_RIGHT;
(pAttrInfo).usVerticalOrientation = NULL;
(pAttrInfo).usMode = SIM_ATTR_IMPLICIT;
(pAttrInfo).usNumericSelectionDefault = SIM_ATTR_ARABIC;
strcpy((pAttrInfo).  szAttributeName, "MyVarChar"); /*Name of the attribute*/
strcpy((pAttrInfo).achLanguageCode, "ENU"); /*Language is specified to be English*/
```

Figure 2. Source for defining a VarChar attribute

```
RCSTRUCT RC2;
ulRC = Ip2CreateAttr(hSession, /*Pointer to session handle*/
(PATTRINFOSTRUCT) &pAttrInfo, /*Pointer to attribute data structure*/
(PASYNCCTLSTRUCT) NULL, /* Request synchronous processing */
(PRCSTRUCT) &RC2); /*Pointer to return code data structure*/
if (ulRC == SIM_RC_OK)
{
//Call Ip2CreateAttr Succeeded
SimLibFree(hSession, /*Pointer to session handle*/
(PVOID) RC2.ulParam1, /*Pointer to return code structure to be freed*/
&RC2);
}
else
{
  printf("Ip2CreateAttr error ulRC = %ld \n", ulRC);
return(1);
}
```

Figure 3. Source for constructing an attribute

It is a general rule of thumb to call SimLibFree after each successful FM call. Consult the *DL Application Programming Reference* for more information regarding when to use this call.

Step 4: Define Data Structures Used to Create the Index Classes

The data structures necessary for constructing an index class are defined in Step 4. In Step 5, the index class is constructed, and in Step 6 the system tables are updated with the new index class information. These three steps—4, 5, and 6—are performed in tandem for each index class constructed.

Once all of the attributes have been constructed, the arguments used by the Ip2CreateClass function can be defined. The two data structures needed by the Ip2CreateClass function are PCLASSATTRSTRUCT and CLASSINFOSTRUCT. The PCLASSATTRSTRUCT defines a contiguous list of attributes defined by CLASSATTRSTRUCT, which contains three fields:

- fAttrRequiredField
- fAttrAccess
- usAttrId

The fAttrRequiredField and fAttrAccess fields define whether an attribute is required and its read/write status respectively. When an attribute is constructed, it is identified by a number which serves as a unique identifier. The usAttrId field contains this unique identifier. Previously in Steps 2 and 3, the attribute was identified by name, MyVarChar. This presents a dilemma of how to map the name, MyVarChar, to DL's unique identifier needed by the CLASSATTRSTRUCT.

In Figure 4, this unique identifier is assumed to be stored in usAttrID. A function, GetAttrIDFromName (defined in the Helper Functions section) can be used to find the attribute ID from a name. In Figure 4, the GetAttrIDFromName function is used to retrieve attribute, MyVarChar's, unique identifier. Once the unique identifier is retrieved via GetAttrIDFromName, it is stored within the CLASSATTRSTRUCT's usAttrID field. The remaining code within Figure 4 defines properties (such as name and description) of the index class required by Ip2CreateClass within Step 5.

Step 5: Create the Index Class

In this step, the Ip2CreateClass function call is made to create an index class called MyIndexClass. The properties of this index class were defined in Step 4 within the PCLASSINFOSTRUCT and PCLASSATTRSTRUCT structures. In Figure 5, the result of Step 4 is passed to Ip2CreateClass via the pClassInfo argument.

```
unsigned short usAttrID;
rc = GetAttrIDFromName(hSession, “MyVarChar”, usAttrID);
if (rc == -1)
{
printf(“GetAttrIDFromName error rc = %d \n”, rc);
return(1);
}
PCLASSATTRSTRUCT pClassAttrStruct;
//Create and initialize a piece of memory to hold the new attribute def.
//We are only adding one attribute to the index class.
pClassAttrStruct = (PCLASSATTRSTRUCT) malloc (1 * sizeof(CLASSATTRSTRUCT));
memset(pClassAttrStruct, 0, (1 * sizeof(CLASSATTRSTRUCT)));
pClassAttrStruct->ulStruct = sizeof(CLASSATTRSTRUCT);
pClassAttrStruct->fAttrRequiredField = FALSE; /*Field is not required. Aka not a key*/
pClassAttrStruct->fAttrAccess = SIM_ATTR_READWRITE; /*Make this attribute read writeable*/
pClassAttrStruct->usAttrId = usAttrID; /*usAttrID is assumed to have attribute id*/
CLASSINFOSTRUCT pClassInfo;
//Create the class info structure for the Ip2CreateClass call.
memset(&pClassInfo, 0, sizeof(CLASSINFOSTRUCT));
(pClassInfo).ulStruct = sizeof(CLASSINFOSTRUCT);
(pClassInfo).pClassAttrStruct = (PCLASSATTRSTRUCT) pClassAttrStruct; /*Assign attribute struct*/
(pClassInfo).usNbrAttrIds = 1; /*One attribute in pClassAttrStruct*/
(pClassInfo).usMaxVersions = 0;
(pClassInfo).usIndexClass = 0; /*Create a new class id*/
(pClassInfo).usViewID = 0; /*Create a new view id*/
strcpy((pClassInfo).  szACLName, “All Privileges”); /*Name of Access Control List*/
strcpy((pClassInfo).szClassName, “MyIndexClass”); /*Index Class name*/
strcpy((pClassInfo).szDescription, “A Description”); /*Index Class Description*/
strcpy((pClassInfo).szCollectionName, “CBR.CLLCTO01”); /*Collection class where parts are stored*/
strcpy((pClassInfo).szStoreSite, “OBJSRVRX”); /*Object server where parts are stored*/
strcpy((pClassInfo).achLanguageCode, “ENU”); /*English language*/
```

Figure 4. Source for defining an index class with name MyIndexClass

```
RCSTRUCT RC3;
unsigned short usRecentClassID;
ulRC = Ip2CreateClass(hSession,
(PCLASSINFOSTRUCT) &pClassInfo, /*Structure defined in step four*/
0,
(USHORT) OIM_USER, /*Use ACL provided in CLASSINFOSTRUCT*/
(PASYNCCTLSTRUCT) NULL,
(PRCSTRUCT) &RC3);
usRecentClassID = (unsigned short) RC3.ulParam1; /*Save the class unique ID for step 6*/
if (ulRC == SIM_RC_OK)
{
SimLibFree(hSession, //Free up the return structure.
(PVOID)RC3.ulParam1,
&RC3);
}
else
{
  printf(“Ip2CreateClass return code %ld\n”, ulRC);
return(1);
}
```

Figure 5. Source for constructing an index class

Successfully completing the `Ip2CreateClass` does not mean that the index class is available for storing data. When constructing an index class, DL constructs a link library on the server which is used for accessing the relational table. The generation/compilation of this link library occurs in an asynchronous fashion. In other words, the `Ip2CreateClass` may return prior to the completion of the link library. Thus, the index class may not yet be ready for access even though the `Ip2CreateClass` function call completed successfully.

If in the event an error is found during data load within a newly constructed index class, it may be due to this problem. The availability of an index class can be checked using the DL Windows NT system administration tool. If the problem persists, check your DL configuration and log files.

Step 6: Cleanup

In Step 5, the index class, `MyIndexClass`, was constructed. However, the DL system administration tool requires two additional function calls to be made to `SimLibCreateItem` in order for it to be able to see the structures created programmatically. `SimLibCreateItem` constructs an item (a row) within an index class (table). The variable `usRecentClassID`, defined in Step 5, is used in this step for populating the data structure used by the `SimLibCreateItem`. The source code provided in Figure 6 is usable for all types of index class creation. Therefore, only high-level concepts of the source will be described.

In Figure 6, a row is added to both the `FRN$INDX` (AVT00004) and `FRN$VIEW` (AVT00005) index classes via the `SimLibCreateItem` function call. These index classes (`FRN$INDX` and `FRN$VIEW`) contain system data for the index class and corresponding view used by the DL system administration tool. Upon successful completion of this step, the index class will be viewable from the system administration tool and accessible from DL-enabled applications.

```
/*Add an item to the FRN$INDX table a.k.a. AVT00004. This item has to have
two columns within that table populated. Column 12 and column 19. These
two columns apply to the index class ID just created and the work priority.
Below, I set the priority to the default of 1. */

RCSTRUCT RC4;
ATTRLISTSTRUCT frnIndexAttrList[FRNINDX_NUM_ATTR];
int j;
//Initialize the data structures needed to create a class index.
for ( j = 0; j < FRNINDX_NUM_ATTR; j++)
{
frnIndexAttrList[j].ulStruct = sizeof(ATTRLISTSTRUCT );
frnIndexAttrList[j].fAttrFlags = 0;
frnIndexAttrList[j].usAttrType = 0;
frnIndexAttrList[j].usAttrId = j + 12;
frnIndexAttrList[j].pszAttributeValue = NULL ;
}
//Initialize the default workflow priority because it is one of two
//datatypes that are prerequisites for index class creation.
// Attr 19 = default WorkFlow Priority
frnIndexAttrList[7].usAttrId = 19 ;
frnIndexAttrList[7].pszAttributeValue = (unsigned char *) malloc(2);
strcpy((char*) frnIndexAttrList[7].pszAttributeValue,"1") ;
char* tempID;
tempID = (char*) malloc(6 * sizeof(char));
sprintf(  tempID, "%u", usRecentClassID);
frnIndexAttrList[0].usAttrId = 12; /* 12 = FRN$INDXCLASSID datatype */
frnIndexAttrList[0].pszAttributeValue = (unsigned char*) tempID;
```

Continued on following page

Figure 6. Source for updating DL's system tables

Continued from previous page

```
ulRC = SimLibCreateItem(hSession,
       SIM_DOCUMENT,
       4, /*id of FRN$INDX*/
       FRNINDX_NUM_ATTR,
       frnIndexAttrList,
       SIM_ACC_CLASS,
       (PASYNCCTLSTRUCT) NULL,
       (PRCSTRUCT) &RC4);
free(tempID);
if (ulRC == SIM_RC_OK)
{
SimLibFree(hSession, //Free up the return structure.
(PVOID)RC4.ulParam1,
&RC4);
}
else
{
printf("1    st    SimLibCreateItem return code %ld\n", ulRC);
return(1);
}
/*Add an item to the FRN$VIEW table aka. AVT00005. This item has to have
one column within the table populated. Column 33. This column applies to
the view ID of the index class just created. Below, I set the view ID
for the frnViewAttrList. */
RCSTRUCT RC5;
ATTRLISTSTRUCT frnViewAttrList[FRNVIEW_NUM_ATTR];
for ( j = 0; j < FRNVIEW_NUM_ATTR; j++) {
frnViewAttrList[j].ulStruct = sizeof(ATTRLISTSTRUCT );
frnViewAttrList[j].fAttrFlags = 0;
frnViewAttrList[j].usAttrType = 0;
frnViewAttrList[j].usAttrId = j + 33;
frnViewAttrList[j].pszAttributeValue = NULL ;
}
tempID = (char*) malloc(6 * sizeof(char));
sprintf(  tempID, "%u", usRecentClassID);
frnViewAttrList[0].usAttrId = 33; /* 33 = FRN$VIEWID datatype*/
frnViewAttrList[0].pszAttributeValue = (unsigned char*) tempID;
ulRC = SimLibCreateItem(hSession,
       SIM_DOCUMENT,
       5, /*id of FRN$VIEW*/
       FRNVIEW_NUM_ATTR,
       frnViewAttrList,
       SIM_ACC_CLASS,
       (PASYNCCTLSTRUCT) NULL,
       (PRCSTRUCT) &RC5);
if (ulRC == SIM_RC_OK)
{
SimLibFree(hSession, //Free up the return structure.
(PVOID)RC5.ulParam1,
&RC5);
}
else
{
printf("2nd SimLibCreateItem return code %ld\n", ulRC);
return(1);
}
free(tempID);
```

Figure 6. Source for updating DL's system tables

Step 7: Log off DL

The final step is to close the connection previously constructed in Step 1. Known as logging off of DL, this is accomplished by calling the `SimLibLogoff` function. Figure 7 shows an example.

```
RCSTRUCT RC6;
SimLibLogoff( hSession,
(PASYNCCTLSTRUCT) NULL,
(PRCSTRUCT) &RC6);
```

Figure 7. Source for logging off DL

Helper Functions Used By Step 4

In Step 4, the function `GetAttrIDFromName` was utilized for obtaining an attribute identifier from the attribute name. Figure 8 shows the source code for this function. The `GetAttrIDFromName` function uses the FM function `Ip2ListAttrs` to retrieve a list of all the attributes defined on a system. Once the list of attributes are returned, a loop is established and the results are iterated. Each iteration checks for the attribute name specified in the argument list. If it is located, the attribute ID is set, which is returned by reference within the third argument, `attrID`, to the caller. *Note:* Not all erroneous situations are handled in this code.

***Bill Baer**, IBM Corporation, Santa Teresa Laboratory, 555 Bailey Ave, San Jose, California 95141. Mr. Baer is a member of the Content Management Solutions Team. He has worked in numerous engagements, architecting and developing content management solutions for IBM's customers. In the past, he has worked in the application development tools, automated test tools, and knowledge-based systems groups within IBM. Mr. Baer has a BS in Computer Science from the University of Missouri at Rolla.*

```
int GetAttrIDFromName(HSESSION hSession, char* szAttrName, unsigned short &attrID)
{
ULONG ulRC; /*Return Code*/
RCSTRUCT RCStructAttrNameSet;
PNAMESTRUCT pAttrInfo;
NAMESTRUCT attrInfo;
attrID = -1;
ulRC = Ip2ListAttrs(hSession,
       (PASYNCCTLSTRUCT) NULL,
       (PRCSTRUCT) &RCStructAttrNameSet);
if (ulRC == SIM_RC_OK)
{
for (int i = 0; i < (int) RCStructAttrNameSet.ulParam2; i++) /*loop through all attributes */
{
       pAttrInfo = (PNAMESTRUCT) RCStructAttrNameSet.ulParam1;
       attrInfo = (NAMESTRUCT) pAttrInfo[ i ];
       if ((strcmp(attrInfo.szName, szAttrName)) == 0 )
       {
              //Attribute name has been found.
              attrID = attrInfo.usID;
}
} //for
SimLibFree(hSession,
       (PVOID)RCStructAttrNameSet.ulParam1,
       &RCStructAttrNameSet);
}
else
{
  printf("Ip2ListAttrs return code %ld\n", ulRC);
}
if (attrID == -1)
       return -1; //Attribute, szAttrName, not found.
return ulRC;
} //function
```

Figure 8. Example of code to log off DL

AIX Questions Part 1

Compiled by Sandra Herzik

The Solution Developer Marketing Technical Support Group in Austin, Texas, supports software vendors who are developing or porting applications to AIX. This article is a compilation of questions that are frequently asked by vendors. The name of the responding Technical Support Group staff member appears after each response.

AIX Architecture

What RS/6000 models are included in each of the RS/6000 processor groups?

The two categories of RS/6000® servers are entry and advanced servers. Entry servers include the following processor groups: D5—2xx, 4xx, 8xx, Cxx, Exx, and F30. Advanced servers include all of the remaining supported systems, 5xx, and 9xx.

AIX® Version 4 for Servers and Advanced Servers can be installed on all machines that run standard AIX 3.2. This includes not only the processor group D5 machines, but the remaining processor groups, E5, F5, and G5.

- Advanced desktop systems (processor group E5) includes the following machines: 7012 (3xx-RS/6000)
- Deskside systems (processor group F5) includes the following machines: 7013 (5xx-RS/6000)
- Rack systems (processor group G5) includes the following machines: 7015 (9xx-RS/6000, Rxx-RS/6000)
- SP™ rack system (processor group P5) includes the following machines: 9076 (all)

—Eric Meachem

How does AIX handle memory load control?

When a process references a virtual memory page that is on disk—because it has been paged out or it has never been read—the referenced page must be paged in and, on average, one or more pages must be paged out. This creates I/O traffic and delays the progress of the process.

AIX attempts to steal real memory, via the page-replacement algorithm, from pages that are unlikely to be referenced in the near future. A successful page-replacement algorithm allows the operating system to keep enough processes active in memory to keep the CPU busy. At some level of competition for memory—depending on the total amount of memory in the system, the number of processes, the time-varying memory requirements of each process, and the page-replacement algorithm—no pages are good candidates for paging out to disk because each of them will be reused in the near future by the active set of processes.

Sandra Herzik

Eric Meachem

This causes continuous paging in and paging out to occur, which is a condition called *thrashing*. Thrashing results in incessant I/O to the paging disk, and causes each process to encounter a page fault almost as soon as it is dispatched. The result is that none of the processes make significant progress. Although thrashing may have been triggered by a brief, random peak in workload (such as all of the users of a system happening to hit the Enter key in the same second), the system may continue thrashing for an indefinitely long time.

AIX has a memory load control algorithm that detects when the system is starting to thrash. It then suspends active processes and delays the initiation of new processes for a time. Five parameters set rates and bounds for the algorithm. The default values of these parameters have been chosen to "fail safe" across a wide range of workloads.

—Anant Badrayani

Anant Badrayani

How are paging-space slots allocated in AIX?

AIX supports two schemes for allocation of paging-space slots. Under the normal, late allocation algorithm, a paging slot is allocated to a page of virtual memory only when that page is first read from or written into. That is the first time that the executing program is interested in the content of the page.

Many programs exploit late allocation by allocating virtual-memory address ranges for maximum-sized structures, and then only using as much of the structure as the situation requires. Those pages of the virtual-memory address range that are never accessed do not require real-memory frames or paging-space slots.

This technique involves some risk. If all of the programs running in a machine encountered maximum-size situations simultaneously, paging space might be exhausted. Some programs might not be able to continue to completion.

The second AIX paging-spaces slot-allocation scheme is intended for use in installations in which this situation is likely to occur, or where the cost of failure to complete is intolerably high. Aptly called early allocation, this algorithm causes the appropriate number of paging-space slots to be allocated at the time the virtual-memory address range is allocated, for example, with `malloc`. If not enough paging-space slots are available to support the `malloc`, an error code is set. The early-allocation algorithm is invoked with the following:

```
export PSALLOC=early
```

This causes all future programs executed in the environment to use early allocation. It does not affect the currently executing shell.

The performance analyst is interested in early allocation primarily because of its paging-space size implications. Many existing programs use the "`malloc` a lot, use what you need" technique. If early allocation is turned on for those programs, paging-space requirements can increase many fold. The normal recommendation for paging-space size is at least twice the size of the system's real memory, but the recommendation for systems that use `PSALLOC=early` is at least four times real memory size.

This is just a starting point. It is best to analyze the virtual storage requirements of your workload and allocate paging spaces to accommodate these requirements. For example, at one time the AIXwindows® server required 250 MB of paging space when run with early allocation.

You should remember that paging-space slots are only released by process (not thread) termination or by the disclaim system call; they are not released by free.

—Anant Badrayani

What are memory-load-control parameters?

Memory-load-control parameters specify the following:

- System memory-overcommitment threshold
- Number of seconds required to make a safe interval
- Individual process's memory-overcommitment threshold by which an individual process is qualified as a suspension candidate
- Minimum number of active processes when processes are being suspended
- Minimum number of elapsed seconds of activity for a process after reactivation

Figure 1 shows these parameters and their default values (shown in parentheses).

Parameter	Default Values
h	High memory-overcommitment threshold (6)
w	Wait to reactivate suspended processes (1 second)
p	Process memory-overcommitment threshold (4)
m	Minimum degree of multiprogramming (2)
e	Elapsed time exempt from suspension (2 seconds)

Figure 1. Parameters and default values

All parameters are positive integer values. These parameters of the memory-load-control facility can be set by the system administrator via the `schedtune` command.

—Anant Badrayani

What is file space allocation?

File space allocation is the method by which data is apportioned physical storage space in the operating system. The kernel allocates disk space to a file or directory in the form of logical blocks. A logical block is the division of a file or directory contents into 4096-byte units.

Logical blocks are not tangible entities; however, the data in a logical block consumes physical storage space on the disk. Each file or directory consists of zero or more logical blocks. Fragments, instead of logical blocks, are the basic units for allocated disk space in the journaled file system (JFS).

—Anant Badrayani

AIX Programming

How can I identify what characters the TCP client is sending to the TCP server?

The sample program in Figure 2 runs on a TCP server and listens for the TCP clients. It receives the TCP client message. The server displays the message on the console and sends it back to the client.

—Anant Badrayani

What is the 64-bit version of the flock() function?

The `flock()` function is from `libbsd`, which is not ported to the 64-bit platform. Its equivalent function in 64-bit is `lockf64()`.

—Anant Badrayani

How can I build an application from the command line in VisualAge C++?

Use the `vacbld` command to build from the command line. Entering this command is equivalent to building from the Integrated Development Environment (IDE) with two exceptions:

- The IDE itself is not available during the build.
- Error messages appear on the console rather than in the Messages window of the IDE.

—Anant Badrayani

```
# include <stdio.h>
# include <sys/types.h>
# include <sys/socket.h>
# include <netinet/in.h>
# include <arpa/inet.h>
# define MAXLINE                512  /* Buffer Length */
# define SERV_TCP_PORT    9876 /* User Defined IP Port number */

main(int ac, char * av[] )
{
        int SockId = 0, NewSockId = 0, CliLen = 0 , ChildPid = 0 ;
        struct sockaddr_in  CliAddr, ServAddr ;
        char SERV_HOST_ADDR [16]; /* server's IP Address */
        if ( ac != 2 )
        {
        {
                printf ("\nSyntax : servtcp IP Address\n");
                printf ( "Please specify IP address on command line\n" );
                return 0 ;
        }
        strcpy ( SERV_HOST_ADDR, av[1] ) ;
        /* open a TCP Socket */
        if ( ( SockId = socket ( AF_INET, SOCK_STREAM, 0 ) ) < 0 )
        {
                printf ( "\nServer : Can't open stream socket \n" );
                return 0;
        }

  /* Bind our local address so that the client can send to us */
        bzero ( ( char *) &ServAddr, sizeof ( ServAddr ) );
  ServAddr.sin_family = AF_INET ;
        ServAddr.sin_addr.s_addr = htonl ( INADDR_ANY );
        ServAddr.sin_port = htons ( SERV_TCP_PORT );
        /* bind the socket to local address */
        if ( bind ( SockId, ( struct sockaddr *)&ServAddr,
                                              sizeof ( ServAddr ) ) < 0 )
        {
                printf ( "\nCan't bind local address \n" );
                return 0;
        }

        /* listen to socket for a possible input  */
        if ( listen ( SockId, 5 ) < 0 )
                printf ("\nListen Error\n" );
        else
                printf ("\nListening\n" );

        /* Continue forever */
        for ( ; ; )
        {
                printf ( "Waiting\n" );
                /* wait for connection from client process */
                CliLen = sizeof ( CliAddr );
                do
                {
                     NewSockId = accept ( SockId, ( struct sockaddr *)&CliAddr,
                        (unsigned long * ) &CliLen );
```

(continued on following page)

Figure 2. TCP client

(continued from previous page)

```
                } while ( NewSockId < 0 );
                ChildPid = fork ();
                if ( ChildPid == 0 )
                {
                        close ( SockId ); /* close original socket */
                        EchoClient ( NewSockId );   /* process the request */
                }
        }
        close ( NewSockId ) ;
        return 0 ;

} /* end of main function */

/* Send the received string back to Client */
int EchoClient ( int SockId )
{
        int NumOfChars  = 0;
        char line [ MAXLINE ];
        FILE * fp;
        fp = fopen ( "message.log", "w" );
        if ( fp == NULL )
        {
                printf ("\nstr_echo : Unable to open file for writing\n" );
                return 0;
        }

        for ( ; ; )
        {
                /* read a complete line from the socket */
                NumOfChars = ReadLine ( SockId, line, MAXLINE );

                /* if any string received from Client, print it */
                if ( NumOfChars > 0 )

                {
                        printf ( "Client's String = %s",line );
                        fprintf ( fp, "%s", line );
                }
                if ( NumOfChars == 0 )
                {
                        fprintf ( fp, "Client Connection Terminated\n" );
                        break ;   /* connection terminated. */
                }
                /* send the same string back to client */
                if ( WriteAll ( SockId, line, NumOfChars ) != NumOfChars )
                {
                        printf ( "\nstr_echo : writen error\n" );
                        break;
                }
        }
        fclose ( fp );
        return 0;
} /* end of function EchoClient */

/* Write all Characters from line buffer on the socket */
int WriteAll ( int fd, char * ptr, int nbytes )
{
```

(continued on following page)

Figure 2. TCP client

(continued from previous page)

```
        int nleft = 0, nwritten = 0;
        nleft = nbytes ;
        while ( nleft > 0 ) /* while chars, write to socket */
        {
                nwritten = write ( fd, ptr, nleft );
                if ( nwritten <=0 )
                        return ( nwritten ); /* error */
                nleft -= nwritten ;
                ptr += nwritten ;
        }

        return ( nbytes - nleft );

} /* end of WriteAll function */

/* read the complete line from the socket */
int ReadLine ( int fd, char * ptr, int maxlen )
{
        int n = 0, rc = 0;
        char c ;
        for ( n = 1; n < maxlen ; n++ )
        {
                rc = read ( fd, &c, 1 ) ;
                if ( rc == 1 )
                {
                        *ptr++ = c ;
                        /* check for CR or LF character sequence */
                        if ( ( c == 0x0D ) || ( c == 0x0A ) )
                                break ;
                        if ( c == 0x03 )
                              return 0 ; /* EOF Detected */
                }
                else
                if ( rc == 0 )
                {
                        if ( n == 1 )
                                return ( 0 ) ;    /* No data read */
                        else
                                break ;          /* EOF */
                }
                else
                        return ( -1 ) ;  /* Error */
        }
        *ptr = NULL ;
        return ( n ) ;

} /* end of ReadLine Function */
```

Figure 2. TCP client

How can I specify the bigtoc option to VisualAge C++?

In the configuration file, specify `link(bigTOC, yes)`. To generate a big table of contents, specify `link(bigTOC, no)` to turn it off.

—Anant Badrayani

During the debugging of optimized code in VisualAge C++, the information in the debugger pane shows the logical error in the code, but in reality the logic is correct. Why?

When you debug optimized code, information in the debugger panes may lead you to suspect logic problems that do not actually exist. Consider the following points:

1. Do not rely on panes, such as stack frame panes, to show the current values of variables. Numeric and char values may be kept in processor registers. This is similar to pointers to other types of variables, such as strings and class objects. In the optimized program, these values and pointers are not always written to memory; in some cases, they may be discarded because they are not needed.

2. Static or external variables can be monitored at function entry and exit points, but their value as displayed by the debugger during debugging of a function may not be valid. The debugger displays the storage allocated to the static or external variable. However, in optimized code, changes to a variable may not be written immediately to storage.

3. The register and storage panes are correct. Unlike panes that show actual variables, such as the stack frame, the registers and storage panes are always up-to-date at the last time execution stopped.

4. Use the Disassembly view of the program to determine whether source statements, whose results you were relying on, have been eliminated during optimization (that is, via dead code elimination, in which code that performs no useful work is removed). You may find, for example, that an assignment to a variable in the source code does not result in any disassembly code. This may indicate that the variable's value is never used after the assignment.

—Anant Badrayani

Jeff Simon

Inocencio Richiez

How can I call C native methods from a Java program?

The example in Figure 3 explains how to call native methods written in C from a Java program using Java Native Interface (JNI) on AIX.

—Jeff Simon and Inocencio Richiez

(continued on following page)

Figure 3. How to call native methods written in C from a Java program using JNI

(continued from previous page)

The sample program is provided to you on an "AS IS" basis, without warranty of any kind. IBM HEREBY EXPRESSLY DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow for the exclusion or limitation of implied warranties, so the above: limitations or exclusions may not apply to you. IBM shall not be liable for any damages you suffer as a result of using, modifying or distributing the sample program or its derivatives.

Each copy of any portion of this sample program or any derivative work, must include the above copyright notice and disclaimer of warranty.

Step 1: Write the Java program that declares the native method.

```
public class ShowCmsg {

            // declare your native method
            public native void printCoutput();
                    // static initializer code: load library at class load time
                    static {
                            System.loadLibrary("ShowCmsgImp");
                    }

             public static void main(String[] args) {

                    // instantiate the super class and invoke it's printCou
                    put method
                    new ShowCmsg().printCoutput();
            }
    }
```

Step 2: Compile the Java program.

From the command line, compile the `ShowCmsg.java` code: `Javac ShowCmsg.java`

Step 3: Create the header file.

From the command line, run the `javah` utility program to create the header file (`ShowCmsg.h`) using the `ShowCmsg.class`. This header file is created with the C function signature that will need to be implemented by native method `printCoutput` of the native code.

From the command line, type: `javah -jni ShowCmsg`

Note that the file `ShowCmsg` does not require an extension. It uses the `.class` file instead of the `.java` file. The code (`ShowCmsg.h`) generated by this utility is shown below.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
     #include <jni.h>
     /* Header for class ShowCmsg */

     #ifndef _Included_ShowCmsg
     #define _Included_ShowCmsg
     #ifdef __cplusplus
     extern "C" {
     #endif
```

(continued on following page)

Figure 3. How to call native methods written in C from a Java program using JNI

(continued from previous page)

```
/*
 * Class:     ShowCmsg
 * Method:    printCoutput
 * Signature: ()V
 */

JNIEXPORT void JNICALL Java_ShowCmsg_printCoutput(JNIEnv *, jobject);
#ifdef __cplusplus
}
#endif
#endif
```

Step 4: Write the native method implementation.

Implement the native method as shown below.

```
// native code: ShowCmsgImp.c
#include <jni.h>
include <jni.h>
#include “ShowCmsg.h”
#include <stdio.h>

// write the native method implementation

JNIEXPORT void JNICALL Java_ShowCmsg_printCoutput(JNIEnv *env, jobject obj) {
           printf(“This message comes from a C program using JNI\n”);
           // other “printf” statements
           return;
}
```

Step 5: Create the makefile.

```
VA_HOME=/usr/bin
JAVA_LIB=/jdk118/J1.1.8/lib/aix/native_threads

ShowCmsgImp.so: ShowCmsg.o
        rm -f ShowCmsgImp.so
        ld -o libShowCmsgImp.so ShowCmsg.o -bnoentry -bM:SRE \
            -G -bexpall -L{JAVA_LIB} -ljava -lc_r
#  Note:   Optionally, the ‘-bE:ShowCmsgImp.exp’ flag can be used to
#         export shared symbol(s) using an export list. The’-bexpall’
#         can be used to export all shared symbols.


ShowCmsgImp.o: ShowCmsgImp.c
         ${JAVA_HOME}/javac ShowCmsg.java
         ${JAVA_HOME}/javah -jni ShowCmsg
         xlc_r -c -I. -I/usr/lpp/jdk118/J1.1.8/include -o
                  ShowCmsgImp.o ShowCmsg.c -L/usr/lib/libc.a

clean:
         rm -f *.class *.o ShowCmsg.h ShowCmsgImp.so
```

(continued on following page)

Figure 3. How to call native methods written in C from a Java program using JNI

(continued from previous page)

Step 6: Run the Java program.

```
note: Set 'LIBPATH' to point to any libraries you are using.
         i.e. (ksh) export LIBPATH=/usr/lib:/lib:/usr/lpp/jdk118/j1.1.8/lib/
aix/native_threads


         At the command line type: java ShowCmsg


output:

         This message comes from a C program using JNI.


</HTML>
```

Figure 3. How to call native methods written in C from a Java program using JNI

How can I embed assembler code directly into C source code? I do not want to simply call an assembler routine from a C routine; I want to include assembler instructions directly within the C code.

Figure 4 shows how to include assembler instructions directly in the C code.

The sample code suggests the use of the keyword asm, which is not supported in the AIX 3.6 or VisualAge 4.0 compilers. The easiest method is to call an assembler routine from a C routine. See *AIX 4.3 Assembler Language Reference* for detailed information.

You can embed assembler instructions directly into C code if you do not need more than one or two assembler instructions. This requires access to the actual machine code for the instructions, which are usually found in the microprocessor programming book. You can use the `pragma mc_func` and `reg_killed_by` to inline the machine code. Figure 5 shows sample code from the IBM documentation of OpenGL.

—Sandra Herzik

```
#  Call assembler from C
#  testfile.c
#  main()
#       {
#  int  xyz;
#
#    _asmCode {
       st      0, 8(1)
       st      12, 4(1)
       stu     1, -szdsa(1)
          }; /* End of Assembler code */
/*  Continue with C code here */
```

Figure 4. Assembler instructions in C code

Where is my command-line compiler located when I use VisualAge 4.0 in AIX 4.3?

The command-line compiler is located in `usr/vacpp/bin`. For a graphical user interface (GUI), use the `vacide` command; otherwise, use the `vaclbd` command.

—Eric Meachem

Where can I get the 3.6.6 compiler upgrade files?

The C/C++ Compiler 3.6.6.0 and the VisualAge C++ Compiler 4.0.2.0 are product refreshes and available with a try-and-buy

```
extern float oneoverx(float x);

#pragma mc_func oneoverx {                      \
        "ec200830"  /* fres fp1,fp1 */          \
}
#pragma reg_killed_by oneoverx fp1

extern void __glTouchNextCacheBlock(volatile char *);

#pragma mc_func __glTouchNextCacheBlock { \
        "7c001a2c"              /* dcbt r0,r3 */ \
}
#pragma reg_killed_by __glTouchNextCacheBlock

extern float fselgreq(float a, float b, float c);

#pragma mc_func fselgreq {                              \
        "fc2110ee"  /* fsel fp1,fp1,fp2,fp3 */          \
                    /* if(fp1 >= 0.0)       */          \
                    /*     fp1 = fp3;       */          \
                    /* else                 */          \
                    /*     fp1 = fp2;       */          \
}
#pragma reg_killed_by fselgreq fp1

extern float fselgr(float a, float b, float c);

#pragma mc_func fselgr {                                \
        "fc200850"  /* fneg fp1,fp1         */          \
        "fc2110ee"  /* fsel fp1,fp1,fp2,fp3 */          \

                    /* if(fp1 > 0.0)        */          \
                    /*     fp1 = fp2;       */          \
                    /* else                 */          \
                    /*     fp1 = fp3;       */          \
}
#pragma reg_killed_by fselgr fp1
```

Figure 5. Sample code from OpenGL

license. Use your favorite network browser to download these from the *ftp://ftp.software.ibm.com/aix/products/ccpp* ftp site.

Be sure to read the readme.txt file for the latest instructions and notices.

—Brenda Hagler

Brenda Hagler

***Compiled by Sandra Herzik**. IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Ms. Herzik is an advisory software engineer. She has a BA in International Relations and an MBA in Finance from American University in Washington, D.C.*

AIX Questions Part 2

Compiled by Sandra Herzik

The AIX Solution Provider Technical Support Group in Austin, Texas, supports software vendors who are developing or porting applications to AIX. This article is a compilation of questions that are frequently asked by vendors. The name of the responding Technical Support Group staff member appears after each response.

AIX System Administration

Does IBM have a Web site that lists the products available on AIX and their availability dates?

The *AIX® Application Availability Guide* can be viewed on the Web at http://www.ibm.com/servers/aix/products/ibmsw/list/. The guide lists products that run on AIX and their availability dates.

—Brenda Hagler

Why does the uptime command complain about the date the system booted?

In most cases where the `uptime` command is complaining about the system boot date, the users have been testing time or date for year 2000. Generally when this occurs, users have moved the date on the system forward and during that test period, the system was rebooted.

Most users move the date backward when they complete their testing. Until the system is rebooted, the system finds the boot time out of sync with the current date. Rebooting the system should solve the problem.

Note that AIX and other applications were designed for time to always go forward; time is not supposed to go backward. For database and other time-sensitive applications, it is always best to reinstall the system rather than reset the date backward. Reinstalling the system to a valid state is the best way to ensure that data corruption will not occur.

—Brenda Hagler

Why do I get the error GRL-2050 when I try to set up LUM?

License User Management (LUM) is the license use management system. If licenses are not set up correctly, users cannot access and use the licensed products on AIX. Users get a GRL-2050 failure when the license server is not set up correctly. Several conditions can cause this problem, so some obvious ones are listed here.

You should ensure that the ports used by LUM are not in use. The command `netstat -a` can determine which ports on AIX are already in use. As you go through the `i4cfg -script` to set up the configuration, the port numbers to be used are listed


Sandra Herzik


Brenda Hagler

and you are given a chance to override the default port number choices. You can run the command `grep bindport /var/ifor/* | pg` to see that the ports normally used are 1515, 10999, and 12999.

Another problem could be another `i4llmd` or `netlsd` process running on the same machine. Kill these processes if they are running. Of course, ensure that others are not using them currently; otherwise, those users will not be able to use the LUM managed products.

To determine if the processes are running, use the command `i4cfg -stop`, then use the commands `ps -ef | grep i4llmd` and `ps -ef | grep netlsd`.

If neither of the above situations is the problem, then remove the files `*.dat` and `*.idx` in the directory `/var/ifor`, start the services with `i4cfg -start`, and see if the license server daemons are up by using the `i4cfg -list` command. Three `ifor` daemons could be running:

- License Server Subsystem: `i4llmd`
- Central Registry Subsystem: `i4gdb`
- Global Location Broker Data Cleaner Subsystem: `i4glbcd`

If the three daemons are running, or if you are only using the one `i4llmd` daemon and it is running, then it is not necessary to re-run the `i4cfg -script` since the appropriate `ifor` daemons are running at this point.

Double check that the `/usr/ibmcxx/cxx36_cn.lic` file contains the correct information. Then, run the license script to enroll the license:

```
# cd /usr/ibmcxx
# ./cxx36_cn.lic
```

Lastly, use this command to confirm the license is in place:

```
# /var/ifor/i4blt -s -l cn
```

The compiler should now work.

For more detailed information on LUM, visit the following URL on IBM License User Management: http://www.software.ibm.com/is/lum/. Also ensure that the URL files are available on your system.

View *file:///usr/opt/ifor/ls/os/aix/doc/en_US/lumusg.htm* as a URL to access the LUM User Guide if you have `ifor_ls.msg.en_US.base.cli` installed.

—Brenda Hagler

When would I increase the tcp_keepidle value for my network?

`tcp_keepidle` is the total length of time to keep an idle TCP connection alive. The default is 14400 half seconds (two hours). After the keep-idle time has passed without any communications, a probe is sent to the client/remote site to see if the client/remote site is alive. Nine probes are sent to determine if the client/remote site is alive. If the client/remote site does not respond, the connection is terminated.

If your network is very busy, the client system may not always respond in the keep-alive time period and the connection could be dropped. If this is the problem with your connections, you could increase the time limit of the `tcp_keepidle` variable to ensure that you do not lose your connection. You can increase the `tcp_keepidle` to a higher value—say 28800 to make it four hours—before the timeout happens.

Use the command `/usr/sbin/no -o tcp_keepidle=28800` to change the `tcp_keepidle` value to four hours. To make this change permanent on your system, add this command to the `/etc/rc.net` script.

Note: You can use the commands `netstat -sp tcp | grep keepalive` and `netstat -sf inet | grep keepalive` to see the connections dropped by `keepalive`.

—Brenda Hagler

How can I easily configure and debug a 7133 Disk Subsystem in AIX 4.3?

As you begin the configuration, keep the following in mind:

1. Serial Storage Architecture (SSA) disk drives are represented in AIX as SSA logical disks (hdisk0, hdiskN) and SSA physical disks (pdisk0,... pdiskN).

2. SSA logical disks represent the logical properties of the disk drive and have volume groups and file systems mounted on them. SSA physical disks represent the physical properties of the disk drive.

3. By default, one pdisk is always configured for each physical disk drive.

Cabling a SSA 4-Port Disk Adapter

1. Each adapter (A and B) has two pairs of SSA connectors.

2. Each pair of ports (A or B) must be connected to the same loop, and the loops must not mix A and B ports from the same adapter in the same loop. That is, SSA Loop B Port 1 -> SSA Loop B Port 2 and SSA Loop A Port 1 -> SSA loop B Port 2.

3. Each loop supports up to 48 devices, but for maximum performance the total number may be considerably less.

Preconfiguration

The following commands ensure that the microcode and the SSA device drivers are at the minimum required levels.

- For microcode level: `lscfg -v`
- For specific pdisk microcode level:
  `lscfg -vl pdisk17`
- For SSA device driver:
  `lslpp - Lc | grep SSA`

The required SSA device drivers are available on the AIX install media and are available to download from http://www.hursley.ibm.com/~ssa/

Configuring SSA Drives

1. The disk drives connected to the system are configured automatically by the system boot process; therefore, it is generally not necessary to take any action to configure the drives.

2. SSA devices can be connected without taking the system off-line. In this case, it could be necessary to configure the SSA disks after the boot process has completed. In this case, configure the devices by running the configuration manager and type the following command at the command line: `cfgmgr`.

3. After the drives are configured, issue the following commands to check on the status of your SSA configuration or use SMIT\Devices\SSADisks\Logical Disks\ Show Logical to Physical Relationship.

4. When you look at the output in Figure 1, keep in mind the following:

   - SSAR (SSA router) will always be in the defined state.
   - Hdisks: There will be one pdisk per hdisk for non-RAID and multiple pdisks per hdisk for RAID.
   - Pdisks and the corresponding hdisks are initially numbered according to a numeric sort involving each disk's hexadecimal serial number (shown on front of disk), not according to the physical location.

5. Check the running processes on the system to make certain the SSA daemon is running by entering the following command: `ps -ef | grep ssa`

If the SSA drives are configured correctly, you should see one process per adapter similar to this output:
`/usr/lib/methods/ssa_daemon -l ssa0`

Debugging Your SSA Configuration

1. Use the SSA Services Aids menus to do any work that involves understanding physical locations of SSA drives. The SSA Services Aids menus are found by using SMITTY\Problem Determination\Hardware Diagnostics\Current Shell\Task Selection\SSA Service Aids or, from the command line, enter:

```
Diag
 -Task Selection
```

2. Use the option "Configuration Verification" to match logical disks (hdisks) to physical disks.

```
lsdev -C | grep SSA

ssa0     Available 20-60        IBM SSA RAID Adapter (14104500)
ssar       Defined                SSA Adapter Router
pdisk0     Available 20-60-P      9.1GB SSA C Physical Disk Drive
pdisk1     Available 20-60-P      9.1GB SSA C Physical Disk Drive
pdisk2     Available 20-60-P      9.1GB SSA C Physical Disk Drive
pdisk3     Available 20-60-P      9.1GB SSA C Physical Disk Drive
pdisk4     Available 20-60-P      9.1GB SSA C Physical Disk Drive
pdisk5     Available 20-60-P      9.1GB SSA C Physical Disk Drive
pdisk6     Available 20-60-P      9.1GB SSA C Physical Disk Drive
pdisk7     Available 20-60-P      9.1GB SSA C Physical Disk Drive
pdisk8     Available 20-60-P      9.1GB SSA C Physical Disk Drive
pdisk9     Available 20-60-P      9.1GB SSA C Physical Disk Drive
pdisk10    Available 20-60-P      9.1GB SSA C Physical Disk Drive
pdisk11    Available 20-60-P      4GB SSA F Physical Disk Drive
pdisk12    Available 20-60-P      4GB SSA F Physical Disk Drive
pdisk13    Available 20-60-P      9.1GB SSA C Physical Disk Drive
pdisk14    Available 20-60-P      9.1GB SSA C Physical Disk Drive
pdisk15    Available 20-60-P      9.1GB SSA C Physical Disk Drive
pdisk16    Available 20-60-P      9.1GB SSA C Physical Disk Drive
pdisk17    Available 20-60-P      9.1GB SSA C Physical Disk Drive
hdisk1     Available 20-60-L      SSA Logical Disk Drive
hdisk2     Available 20-60-L      SSA Logical Disk Drive
hdisk3     Available 20-60-L      SSA Logical Disk Drive
hdisk4     Available 20-60-L      SSA Logical Disk Drive
```

Figure 1. SSA devices

3. Use the Set Service Mode option to replace a disk. This mode will turn on a check light for disk replacement.

4. The best tool for viewing and understanding the loop configuration and status of each disk is the SSA Link Verification Option. When using this menu selection, it is helpful to have a picture of the physical layout of the disks and cables.

5. To list the attributes of your SSA installation, use the following commands:

```
lsattr -El ssar
   lsattr -El pdisk0
   lsattr -El hdisk1
```

6. Several SSA diagnostic tools described in the `/usr/ssa/disk/ README.ssatools` are available to be included in scripts. *Note:* These tools are also available in SMIT/SMITTY.

7. The SSA Spare Tool manages SSA storage networks. This tool functions with the family of SSA adapters, SSA disk units, and SSA enclosures developed by the IBM Storage System Division. You can download the SSA Spare Tool by accessing the following URL:
http://www.hursley.ibm.com/~ssa/rs6k/

   Once you have selected your adapter and level of AIX, you can download SSA Spare Tool.

—Sandra Herzik

AIX System Administration

My PC emulator does not display the correct keys when they are pressed. How can I determine what keys the emulator is mapping on my RS/6000?

To identify the mapping, enter the following at the AIX command line after the login:

```
cat | od -x (press return)
<press the "backspace" then <return>>
<control-d>
```

The result on AIX should be as follows:

```
0000000  0a00
0000001
```

If this does not occur when you enter this on the PC emulator, then you must remap the backspace key on the PC emulator.

—Sandra Herzik

Can I create a file system using removable media on AIX?

Journaled File Systems (JFSs) cannot be created on a tape device for the following reasons:

- AIX does not support this process.
- No JFS logging mechanism exists for these devices.
- A tape device is not randomly accessible.
- IBM tape device drivers do not provide a block interface; therefore, we mount `/dev/hd4` (block interface) rather than `/dev/rhd4` (character interface).

Even if creating a file system on tape were possible, the performance would be very slow. Also, AIX does not support JFS file systems on floppy diskettes.

—Wade Carlin

How can I display characteristics of the file system on AIX 4.3.x?

Enter the following command as root:

```
lsfs -a -q
```

—Wade Carlin

How can I determine the current block size of the tape device?

Enter the following to determine the block size:

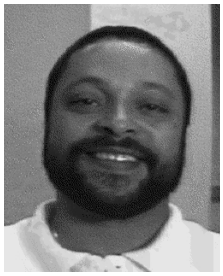
```
tctl -f /dev/xxx status  (xxx = rmt0
or the name of the tape device)
```

—Wade Carlin

What filesets are required to make a backup copy of a non-rootvg file system on AIX 4.3.0?

The bos.sysmgt.sysbr.4.3.0.0 fileset must be installed to make a backup copy.

—Wade Carlin



Wade Carlin

AIX Gotchas

Why do undefined symbols messages appear when I perform compiles across NFS-mounted file systems?

During the link phase of the build process, the following message may appear: `ld: 0711-317 ERROR: Undefined symbol: <symbol_name>`. If your file is to be written onto the NFS-mounted file system on the server, the file could end up with 4 KB sections of all zero bytes. If you move the files onto your system and the build is successful, then you need to upgrade the bos.net.nfs.client fileset on your system.

The upgrade fix for this on AIX 4.3 is IX75226; for AIX 4.2 the fix is IX75138. These fixes will upgrade the bos.net.nfs.client fileset to AIX 4.3.1.0 and AIX 4.2.1.11, respectively.

—Brenda Hagler

When I try to install ipx.base.rte, why do I receive an error message requesting that ipx.rte be uninstalled first?

The previous name for one of the filesets is ipx.rte; however, ipx.base.rte is the new fileset name beginning in AIX 4.3. You must remove the previous ipx.rte fileset before the new ipx.base.rte will install.

To remove the fileset, use the `smitty` command, then follow these steps:

1. Select “Software Maintenance and Utilities.”
2. Choose “Remove Installed Software.”
3. Press F4 to list the software.

4. Find the ipx.rte fileset in the list and press F7 to select it.

5. Press the Enter key to return to the menu.

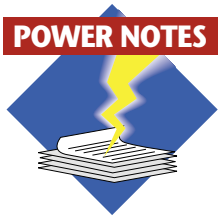
6. Change the "PREVIEW only? (remove operation will NOT occur)" to a "no" and press Enter to actually remove the fileset.

7. Next, install the ipx.base.rte fileset.

—Brenda Hagler

Compiled by Sandra Herzik. *IBM Corporation, 11400 Burnet Road, Austin, TX 78758. Ms. Herzik is an advisory software engineer. She has a BA in International Relations and an MBA in Finance from American University in Washington, D.C.*

Backing Up a Non-Root Volume Group in AIX 4.3



By Wade M. Carlin

The savevg command finds and backs up all files belonging to a specified volume group. A volume group must be varied-on, and the file systems must be mounted. The savevg command uses the data file created by the mkvgdata command. This hint should be used only as a supplement. See the AIX System Management Guide *for more details.*

The volume group has many files that must be backed up. The `mksysb` command only backs up the root volume group, so another approach is required to back up other volume groups on the system: the `savevg` command. This command creates a backup of all files that belong to a specific volume group. The `restvg` command restores a `savevg` image, including re-creating the volume group and its logical volumes and file systems.

Although all logical volume information is archived and will be re-created, only Journaled File System- (JFS-) mounted file system data is archived. The `savevg` command does not archive raw logical volume data. For more information on raw logical volumes, contact your AIX support center.

Creating a savevg

Two methods can be used to create the `savevg` command:

- **Using System Management Interface Tool (SMIT).** Fill in the correct device or file name to be used. Press Enter to start the backup. If the user is writing the `savevg` to tape and more than one tape is required, SMIT will prompt the tape change. Figure 1 shows the steps.

- **Using the command line.** From the command line, enter the following:

```
savevg -if /dev/rmt# vgname
2>/tmp/savevg.err
```

Note: A `savevg` that is written to tape cannot be booted. When creating a `savevg` to tape, do not stack the image onto an existing tape. A `savevg` image must be the first image on the tape in order to restore the tape.

```
smit savevg
Backup DEVICE or FILE                              [ ]
Create MAP files?                                  no
EXCLUDE files?                                     no
Number of BLOCKS to write in a single output       [ ]
   (Leave blank to use a system default)
```

Figure 1. Creating savevg using SMIT

Wade M. Carlin

Verifying a savevg

The only method to verify that a volume group backup will correctly restore with no problems is to actually restore the `savevg` onto another machine. This should be done by following the disaster recovery plan for your company.

The test described below can help minimize problems caused by tape media damage. This test verifies that the tape media can be read on the machine, but it does not guarantee that the `savevg` will be restored successfully. Figure 2 shows the use of SMIT.

1. Type or select values, and press Enter only after selecting all values.

2. Enter the following: `restore -Tvqf /dev/rmt#.1 > /tmp/savevg.log`

Restoring a savevg

The `resvg` command restores the user volume group with all of its containers and files, as specified in the `tmp/vgdata/vgname/vgname.data` file (where `vgname` is the name of the volume group) found within the backup image created by the `savevg` command.

Determine which drive to use for the restored volume group. A volume group cannot already exist on the physical drive that you choose. Be sure to select all target physical drives.

Using SMIT, enter the code in Figure 3. Be sure to select all physical volumes that will make up the new volume group.

To restore from the command line, enter the following:

```
restvg -f /dev/rmt# hdisk# {hdisk# ...}
```

To restore individual files or directories from a `savevg` image, treat the `savevg` like any other relative backup and restore image. For example:

```
restore -xvqf /dev/rmt# ./path/file_to_restore
```

The `savevg` command is always created with relative path names.

Tips and Hints for savevg

The `restvg` command expects the `savevg` image to be the first image on the tape. If the `savevg` image is not the first image on a tape, the `restvg` command is not supported as a means to restore that image. Individual files can still be restored from that image, but the tape must be forwarded to the correct filemark. The AIX support center can provide further information about the `restore` command.

For `savevg` and `restvg` to work properly with an image that was created with

```
smit lsbackvg
DEVICE or FILE                                    [/dev/rmt#]
Number of BLOCKS to read in a single input         [ ]
    (Leave blank to use a system default)
```

Figure 2. Using SMIT to verify tape media

```
smit restvg
Restore DEVICE or FILE                            [/dev/rmt0]
SHRINK the filesystems?                            no
PHYSICAL VOLUME names                             [ ]
   (Leave blank to use the PHYSICAL VOLUMES listed
    in the vgname.data file in the backup image)
Number of BLOCKS to read in a single input         [ ]
   (Leave blank to use a system default)
```

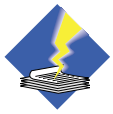
Figure 3. Choose the drive for the restored volume group

both maps and mirroring, AIX must be running at the proper maintenance level of `bos.sysmgt.sysbr`. To verify the level, enter:

```
lslpp -h bos.sysmgt.sysbr
```

Then use the following table to verify that AIX is at the correct fileset level:

```
4.3.0        bos.sysmgt.sysbr 4.3.0.0
```



Wade M. Carlin, *IBM Corporation, 11400 Burnet Road, Austin, TX 78758. E-mail: wadec@us.ibm.com. Mr. Carlin is the AIX lead subject matter expert in AIX Solution Developer Technical Support. Mr. Carlin has over 15 years experience with AIX protocol. He is currently the project leader of the AIX Partners In Development team within IBM's Solution Developer Marketing Technical Support Center. He has a BS in Computer Science from Southern University in Baton Rouge, Louisiana.*