Taking Out the Garbage

By Mark Schnedar

The newest BBx generation, BBj®, is a Java application. As with all Java applications, the Java Virtual Machine (JVM) manages memory, with a major component of Java memory management being "garbage collection." When a JVM performs garbage collection, it scans memory looking for objects (like Strings, Integers, or File handles) that the application no longer uses and frees up that memory for reuse. Garbage collection technology improves with each new version of Java, but the easiest and most effective way to reduce garbage collection overhead is simply to create less garbage in the first place. With that in mind, we analyzed BBj to find areas where we could rewrite code to use fewer temporary objects. One area of significant improvement is in reading records sequentially through an MKEYED file.

Fewer collections = increased speed + performance!

To compare garbage collection overhead between BBj 4.03 and BBj 5.0, we ran BBjServices with the **-verbosegc** Java flag, which writes all garbage collection information to a log. Here is an example of what BBjServices wrote to the log with the **-verbosegc** flag:

```
[GC 27587K->7877K(33452K), 0.0030826 secs]
```

The first number in this example reports the amount of used space before the collection; the second number is the amount of used space after the collection. The number in parentheses is the maximum amount of space for the new generation. With a simple calculation (27587 KB minus 7877 KB), this example shows the JVM reclaimed 19710 KB of garbage.

The second example, available online a www.basis.com/advantage/mag-v9n1/takeout.html, quantifies the decreased garbage collection overhead while reading an MKEYED file in BBj 5.0. It creates a simple MKEYED file, writes 10000 records to it, then reads those records sequentially 1000 times. To show a comparison, we ran this program in BBj 4.03 and BBj 5.0, then extracted the garbage collection information from the logs, and inserted it into a spreadsheet. **Figure 1** summarizes the results logged in each spreadsheet. Notice the reduction in the 5.0 collection and the total size after optimization.

BBj 4.03

Collection Before Optimization	
Number of collections:	110
Total size of collections:	1.9 GB

BBj 5.0

Collection After Optimization	
Number of collections:	92
Total size of collections:	1.5 GB

Performance Improvements	
Collections decreased:	16%
Garbage reduced:	21%

Figure 1. Garbage collection comparison and summary.

In BBj 4.03, the sample program performed 110 collections for a total of 1,907,0013 KB and an average of 17,336.48 KB per collection. In BBj 5.0, the same

continued...

Mark Schnedar
Senior Software Engineer

program made only 92 collections, totaling 1,592,300 KB for an average collection size of 17,307 KB. Comparing the two examples, BBj 5.0 generated 314,713 KB less garbage than BBj 4.03. Though the amount collected during each garbage collection event for both tests was very close, BBj 5.0 made 18 fewer collections than BBj 4.03. These two tests reveal that BBj 5.0 reduced the number of collections by 16% and the amount of generated garbage by 21%.

Another area of significant improvement is in reading fields, rather than records as our example shows. BASIS rewrote the code for reading fields to create less garbage. Specifically, this improvement reduced garbage by 21%, similar to our example.

Summary

By optimizing BBj to generate fewer temporary objects, the JVM can perform fewer garbage collections. Reducing the time spent collecting garbage leaves more time for doing real work and allows BBj programs to run faster.

For additional information on tuning your JVM memory parameters in Java 5.0; see the following articles on the Java Web site:

http://java.sun.com/j2se/1.5.0/docs/guide/vm/gc-ergonomics.html

http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html