Tuning the BBj Performance Analyzer

By Jim Douglas

For several years, BBj®, PRO/5® and Visual PRO/5® developers have optimized their applications with the help of the Performance Analyzer. Recently, we discovered many have used it to process trace files as large as hundreds of megabytes. These file sizes are much larger than anyone anticipated, which resulted in long import times. We investigated the trace file import process and found there was room for significant improvement. This article addresses the review process and details the improvements in the Performance Analyzer for the BBj 6.0 release.

To begin testing, we inserted a SETTRACE in the Performance Analyzer import code (File→Open), then we imported a typical 10 MB trace file of about 108,700 lines of code. Next, we loaded this trace file into the Performance Analyzer itself.

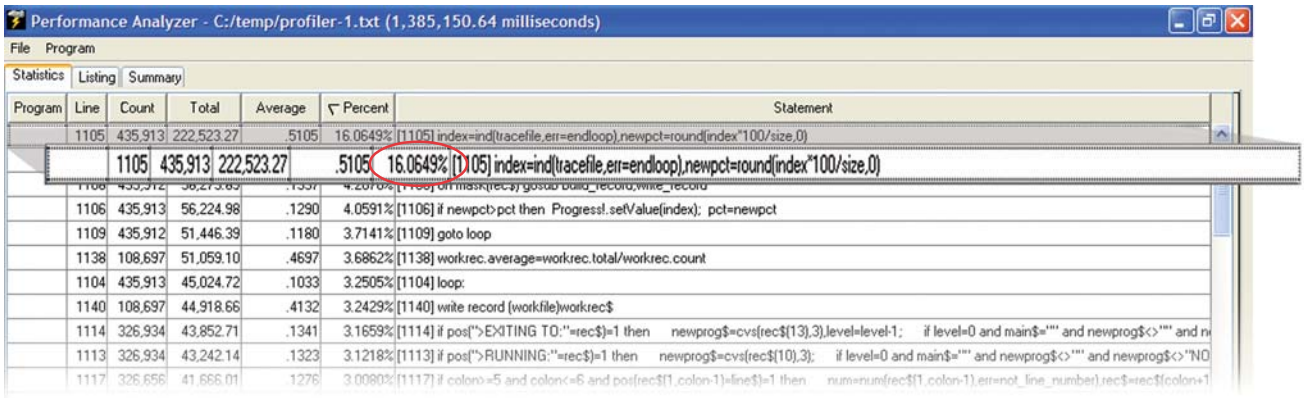


Figure 1. Results from importing a typical 10 MB trace file into the Performance Analyzer

Importing that 10 MB trace file into the Perfomance Analyzer took about 23 minutes. About 16% of that total, shown in **Figure 1** was spent tracking the import process to update the progress bar on the screen. Based on this finding, we changed the import process to only perform that calculation once for every 1,000 records read from the trace file. This change cut about a minute from the total import time and reduced the import process to more than 6.7% of the total. **Figure 2** shows the new results.

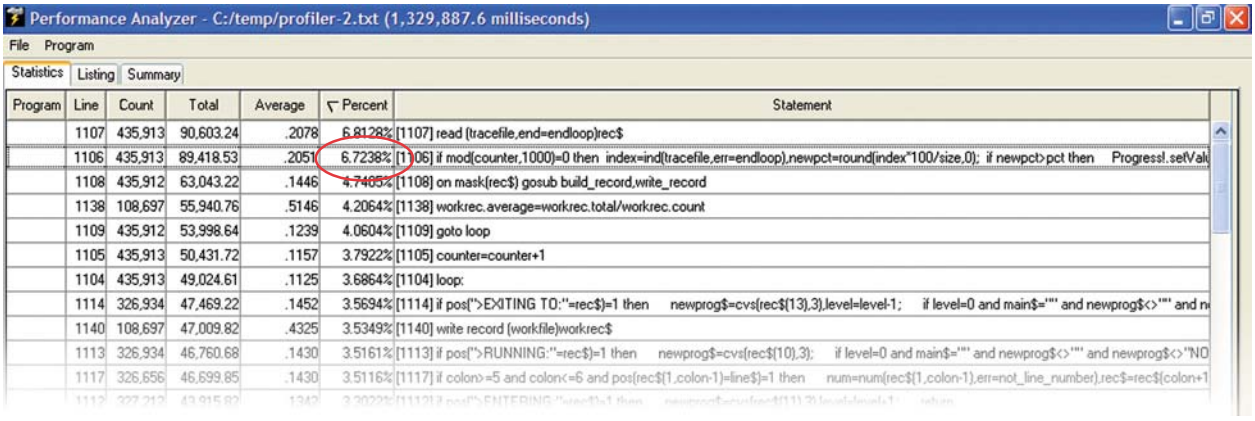


Figure 2. Results of calculation performed every 1000 records

A review of the new profile showed that several lines of code executed more than 435,000 times. The trace file contains 108,700 lines of code, plus the same number of timing lines, for a combined total of 217,400 lines. So why is the Performance Analyzer reading over 435,000 records from the file?

Since this test ran on Microsoft Windows, each line in the trace file is delimited by **`$0D0A$`**. The BBj READ verb sees this as one line terminated by **`$0D$`**, followed by another (blank) line terminated by **`$0A$`**. To eliminate excess processing, we modified the READ statement to check for

continued...

Jim Douglas
Software Engineer
Contractor

these empty lines. If a record is blank, the program skips most of the processing, and just goes back to get the next record. **Figure 3** shows the profile for this version.

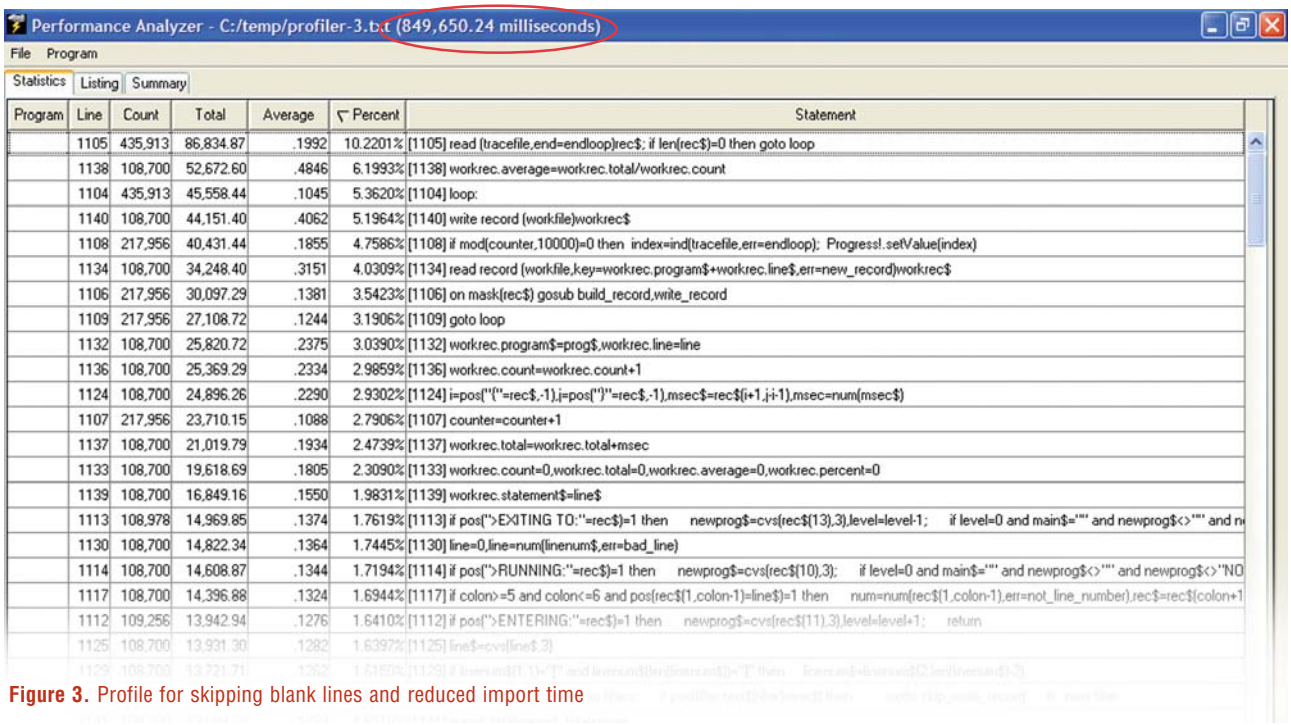


Figure 3. Profile for skipping blank lines and reduced import time

This last change trimmed the import process by another 8 minutes. In total, the import time was cut to 850 seconds (14:10). This is almost 9 minutes (535 seconds) faster than our original test – a 39% reduction. That's a significant improvement for a few hours work, but it was still taking 14 minutes to import a moderately sized trace file. Since developers work with trace files as large as 500 megabytes, these incremental improvements were not enough.

Examine Your Assumptions

At this point, we thought about the overall structure of BBj trace files and the Performance Analyzer. As originally designed, the Performance Analyzer reads through a trace file and stores summary information in an MKEYED file. When it finishes reading the trace file, it attaches the MKEYED file to a data-aware grid. This means that it has to read and update an MKEYED record for each line processed from the trace file. The trace file used in our testing contains 108,700 total lines of code, but only 2,027 unique lines of code. This is fairly typical; as most programs execute the same few lines of code many times. Though it is not practical to keep information about hundreds of thousands of lines of code in memory, we can easily calculate the summary information for a few thousand lines of code. By performing all of the processing in memory, we can completely eliminate the MKEYED file. Instead of attaching an MKEYED file to a data-aware grid, we can update the summary information directly from memory to a standard grid.

Ultimately, we decided to implement this approach using a Java utility program (referenced at the end of this article). However, the most dramatic improvements did not come from rewriting in Java; they came from examining our basic assumptions and taking a new approach.

Figure 4 shows the final test using the Java implementation. Total import time is less than 3½ seconds – nearly 250 times faster than our best time with the previous approach identified in **Figure 3**, and almost 400 times faster than the original verison shown in **Figure 1**.

BBj 6.0 Performance Analyzer Enhancements

Improved performance is the most obvious change in the BBj 6.0 Performance Analyzer – it loads trace files hundreds of times faster than the original Performance Analyzer. We also added several usability enhancements, many of them requested by BBj developers.

continued...

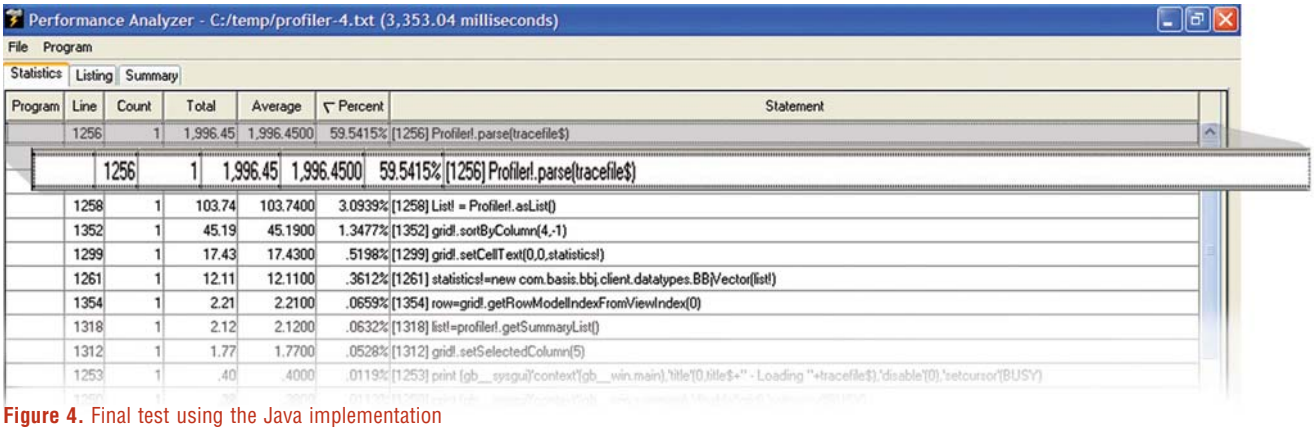


Figure 4. Final test using the Java implementation

Client-Side Grid Sorting

The original Performance Analyzer stored statistical information in an MKEYED file and presented it using a data-aware grid. Clicking in a column header caused the program to generate a sorted SELECT command and attached that new SELECT command to the data-aware grid. Though this process was quick, there was a noticeable pause of a second or two and a visible repaint of the screen. The BBj 6.0 Performance Analyzer handles all sorting directly on the client. This is faster and visually smoother than the original approach. It turns out that this new approach is also easier to implement; it uses less code – and simpler code – than the original file-based approach. For more information on sorting, refer to the article, Grid to Order on page 16.

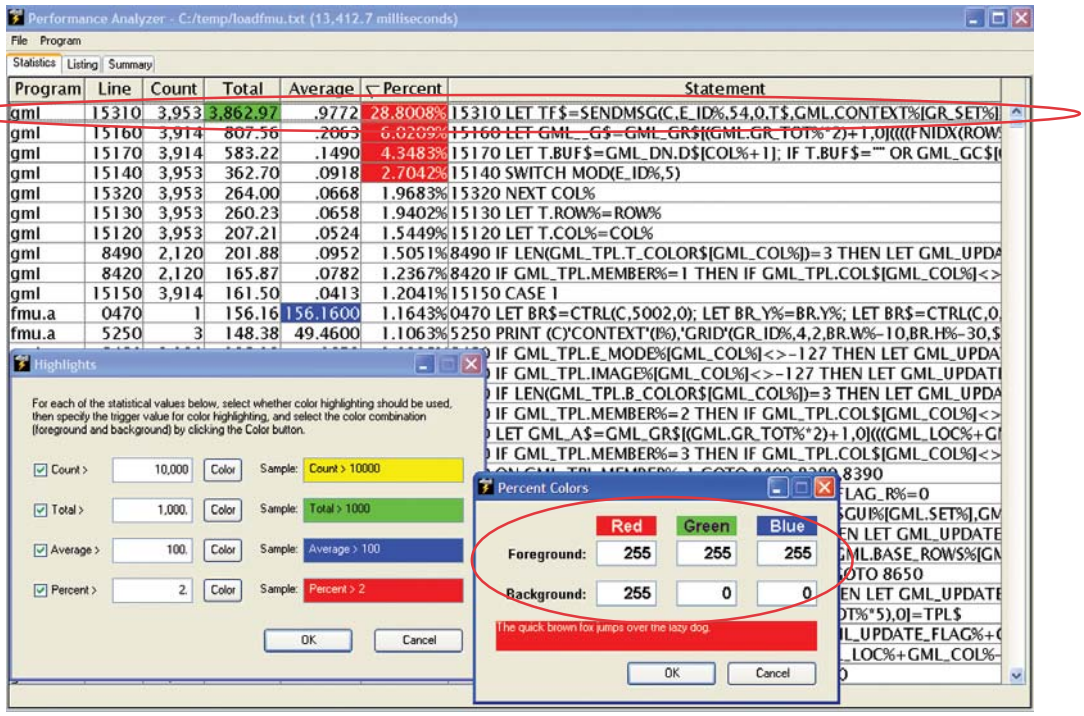


Figure 5. Color highlighting

Highlights

The Performance Analyzer presents a great amount of information, making it hard to decide on which lines of code to focus. The new color-highlighting feature, available from the Program→Highlights menu, enables developers to choose which criteria are important and highlight those specific pieces of information. The example shown in **Figure 5** highlights the following values:

- all lines executed more than 10,000 times appear highlighted in yellow (none in this sample)
- all lines responsible for more than 1,000 milliseconds (1 second) of total execution time appear highlighted in green
- all lines that took more than 100 milliseconds to execute (on average) appear highlighted in blue
- all lines responsible for more than 2% of total execution time appear highlighted in red

Figure 5 shows how the lines that meet the highlighting criteria jump off the screen, directing the viewer's attention to the areas of the program worth considering for optimization. The color highlighting also appears in the listing view when loading a program listing (by double-clicking on any line of the Statistics or Summary grids).

continued...

Also new in BBj 6.0 are Filters, accessible from the Program→Options dialog. Filters, shown in **Figure 6**, omit lines containing specified strings from the statistical calculations.

To illustrate how this works, **Figure 7** shows a complete profile of the Performance Analyzer from the beginning of the program through to the parsing of a trace file. It is hard to focus on specific execution bottlenecks because about 59% of the measured time in this trace is spent waiting in the 'fileopen' dialog or otherwise waiting for user input (reading events from the event queue).

Figure 8 shows the trace without the program lines that only measure user typing time ('fileopen' and reading from the SYSGUI event queue). This gives us a better understanding of the real performance bottlenecks.

continued...

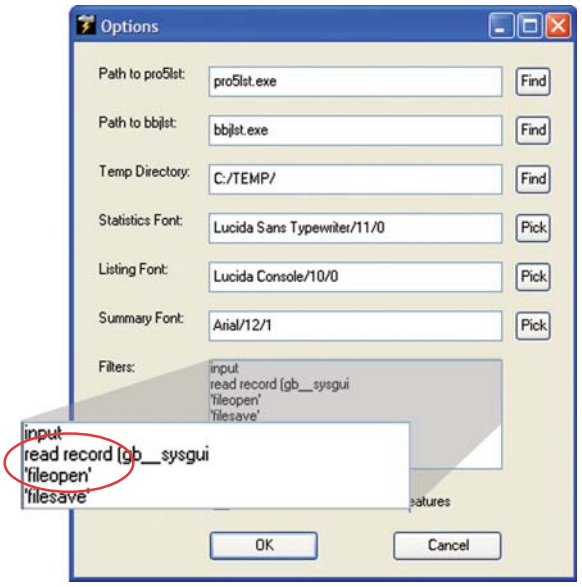
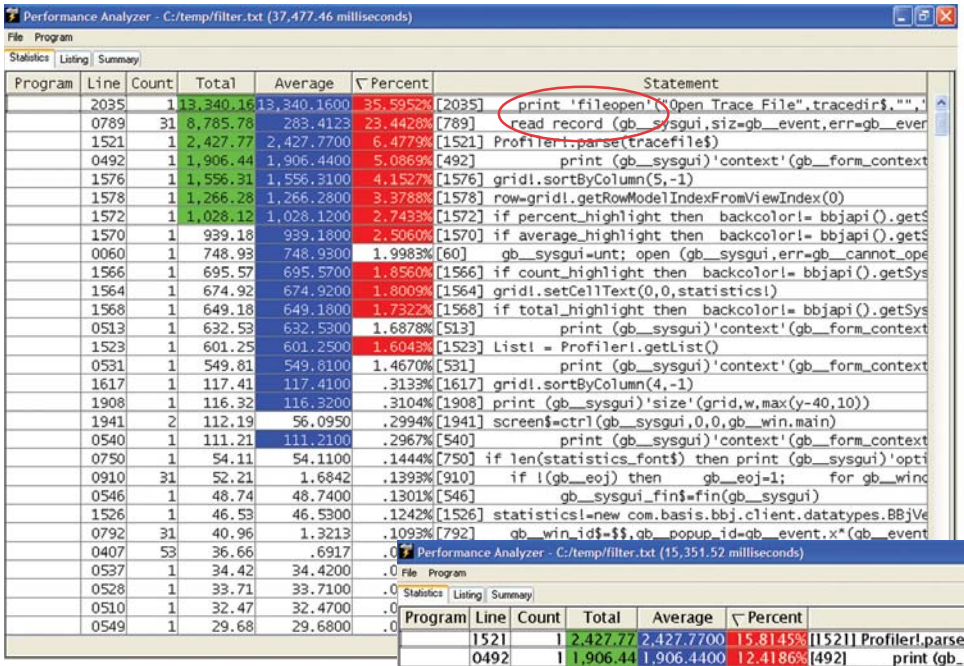


Figure 6. Program options showing filter's applied



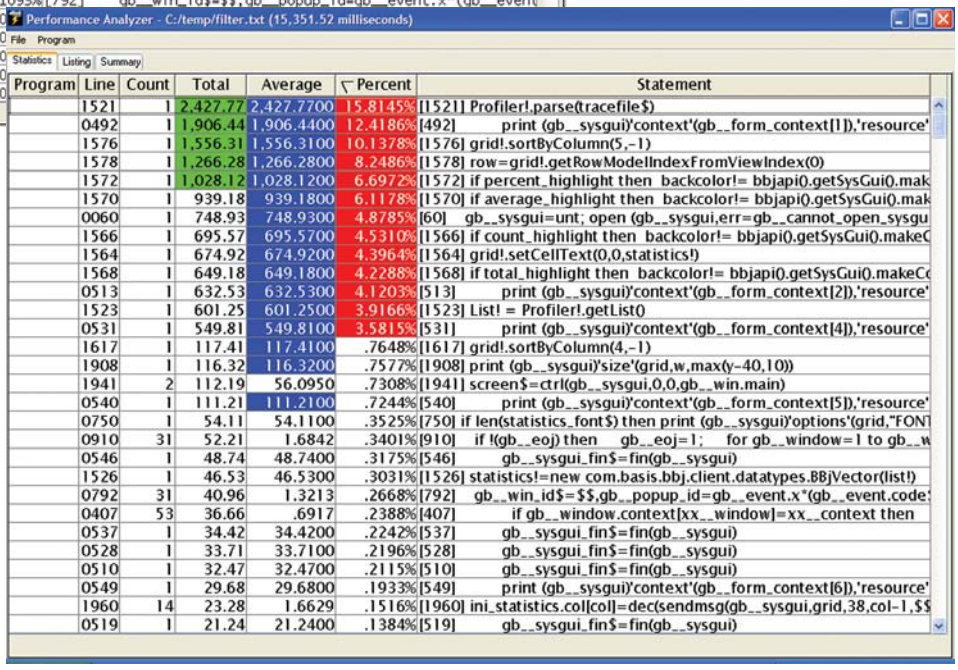


Figure 7. A sample profile without filters

Figure 8. Profile showing the effect of filtering out the 'fileopen' and user input lines

Other Enhancements

- The Performance Analyzer remembers the most recent directories in which the developer opened trace and program files, setting those directories as the defaults until otherwise changed.
- The program listing grid is sortable, and it shows the same color highlighting as the statistics grid.
- The summary grid is now sortable. This is useful for picking which program of many is responsible for the biggest portion of total execution time.
- The initial presentation of the statistics and summary grids now appear in descending order by the Percent column. This is the most commonly used view for focusing on program hot spots.
- The Performance Analyzer now shows complete program statements. The original Performance Analyzer wrote statistical data to fixed-length MKEYED records, limiting the display to only the first 156 characters of each program line.

Summary

The new Performance Analyzer will be available in BBj 6.0. A preview version is available in the BBj nightly build at www.basis.com/devtools/bbj/download_nightly.html.

Download, compile, and run the sample Java utility program referenced in this article at www.basis.com/advantage/mag-v9n2/PA.zip.

BASIS first released the Performance Analyzer in BBj version 2.01. Read *Tuning Your Code With the Performance Analyzer* at www.basis.com/advantage/mag-v6n3/tuning.html.