Debugging SPROCs and Triggers

Stored procedures (SPROCs) and triggers have proven themselves over the last few years as they provide developers with critical new capabilities and expand data access and connectivity options. After the developer has written, debugged, and deployed them, they run like a well-oiled machine and rarely require any maintenance or updates. Running silently in the background, stored procedures and triggers do their job behind the scenes, unbeknown to most users. Getting them to that point, however, may take a little more time and expertise compared to a typical 'foreground' application. The reason for this extra effort is because these types of programs must be able to run in the background without any user interaction and be prepared to handle any error condition or abnormality that occurs during its lifetime.

Therefore, the writing and debugging phase of trigger and stored procedure development is typically more challenging than writing a foreground program, however a new feature in BBj® 9.0 provides welcome relief!

By Nick Decker
Engineering Supervisor

Debugging Triggers and Stored Procedures – an Exercise in Frustration

If you have ever tried to debug a SPROC or a trigger, you are familiar with that initial feeling of helplessness that often overcomes new developers. After all, you only have control of the client application that causes the trigger or SPROC program to execute; you do not have direct control over that program's execution. Furthermore, attempting to execute the trigger or SPROC directly always results in failure. This is because they rely on constructs such as the BBjTriggerData or BBjStoredProcedureData objects in order to execute successfully – and these will not be available. So you are in a difficult position as you cannot run the trigger program directly. Instead, you must let the BBj file system execute it in response to designated file access. However, when BBj launches the program in the background, you have no way of interacting with it in order to debug problems.

A Clumsy Workaround

Before BBj 9.0, a programmer's only recourse was to log everything out to a file. Because it was not possible to dot-step through the trigger program interactively, developers would sprinkle the code with dozens of lines of debug code to print out program and variable state to a log, almost like debugging in a non-interpretive language. This workaround succeeded, but was clumsy and cluttered the program with large quantities of debug code. To add insult to injury, the developer would then remove the debug code, only to realize that it was necessary once again during a future debug session and required reinsertion.

The Elegant Solution

BASIS realized that this process was not optimal and sought a solution. The most straightforward solution – letting the developers interact with a background process – seemed unlikely at first because that has never been possible in the past. The benefits were so tempting, though, that the clever coders behind BBj figured out a way to do just that!

Taking the Plunge – Debugging a Trigger or SPROC

Before jumping into a trigger or SPROC debug session, there are a couple of prerequisites. First, you must turn on the ability to debug triggers and SPROCs in Enterprise Manager (EM). Interactive SPROC and trigger debugging is aimed primarily at developers, and because the option is a global setting for BBjServices, it is 'off' by default. **Figure 1** shows EM's Server Environment tab with the 'Allow Trigger Debugging' checkbox selected.

In order to debug a SPROC, set the similar option for the desired database. **Figure 2** shows the database's miscellaneous tab with the 'SPROC Debugging' checkbox selected. >>

After enabling interactive debugging, ensure that BBj Services is running in an environment that supports a graphical SysConsole. Finally, if BBj is running on MS Windows, BBj Services must not run as a service as it will not have sufficient access to the desktop to display a SysConsole.

Once you have met all of the prerequisites, initiating a debug session is as simple as modifying the program code for the trigger or SPROC, usually by adding an ESCAPE. Then, force the program execution by calling the SPROC or accessing the file with the trigger.

Our example uses a trigger for the ChileCompany Customer data file. Begin by mounting the ChileCompany data directory, then adding the Customer file as shown in **Figure 3**. >>

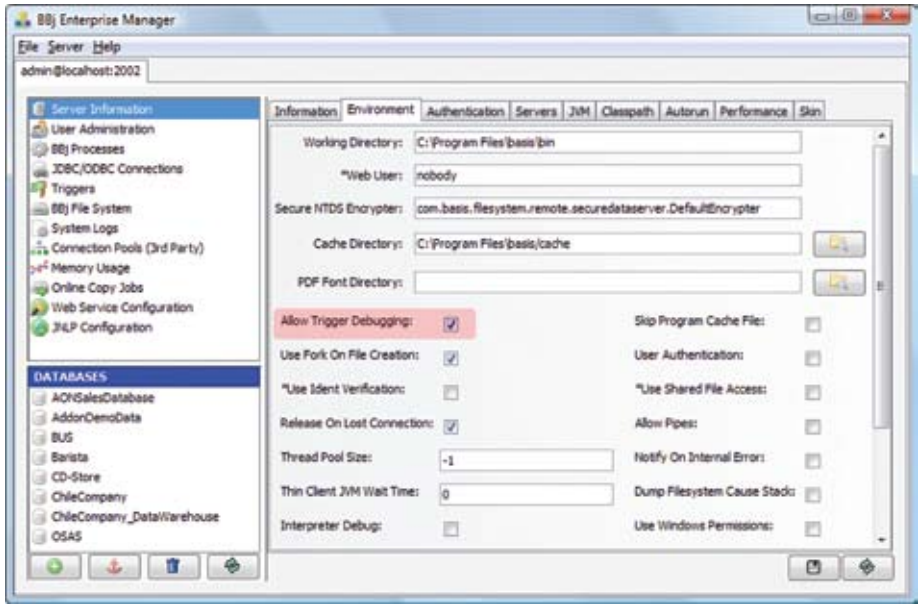

Figure 1. EM setting that allows trigger debugging

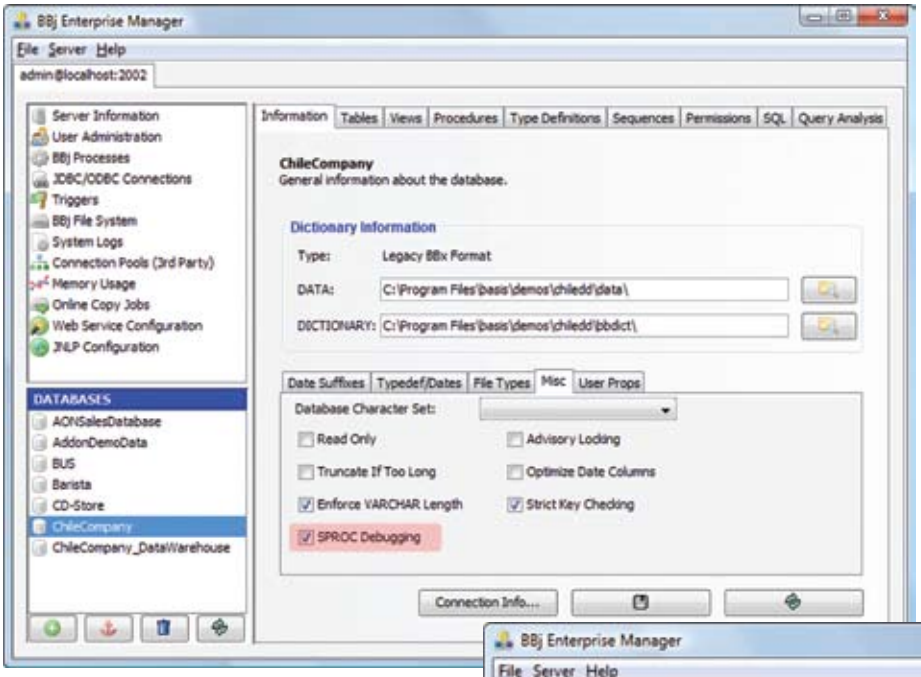

Figure 2. EM setting that allows SPROC debugging

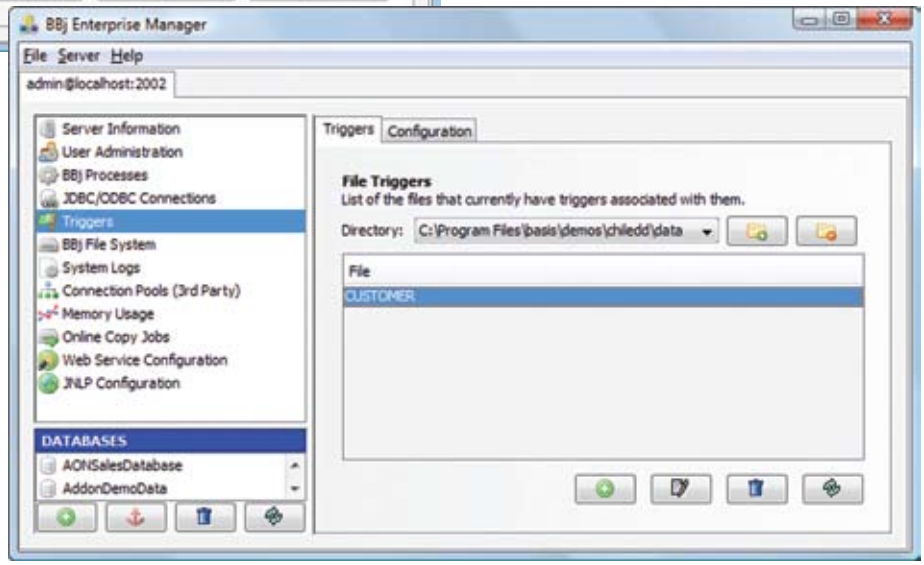

Figure 3. Mounting the data directory in Enterprise Manager

Next, select and enable the desired trigger. **Figure 4** shows that we have defined an 'After Read' trigger, meaning that the BBj trigger program executes after BBj reads the requested data from the disk.

The Trigger Editor in **Figure 4** also displays a read-only copy of the trigger program, allowing for easy identification and preview capabilities. Note the ESCAPE at the end of the code causes BBj to display an interactive SysConsole as soon as it executes that line. To test the trigger, simply perform an ordinary read on the customer data file. BBj will read the specified data as usual and then fire the After Read trigger. The trigger code executes until it hits the ESCAPE, as shown in **Figure 5**.

This also demonstrates that we are able to interact with the trigger program. After the trigger has paused execution following the ESCAPE, we are able to print out the Trigger! object and the contents of rec$, which is the trigger's read buffer (the data that was read from the disk and is destined for the client application). Because the session is completely interactive, it is possible for us to modify the contents of rec$ to return different information back to the original program or query that caused the trigger to fire.

After completing the debug session, type in **BYE** or **RELEASE** in the SysConsole to exit the trigger code and return the results to the client. Once the program runs to satisfaction, simply remove the ESCAPE and disable interactive debugging in Enterprise Manager to return everything to normal production mode.

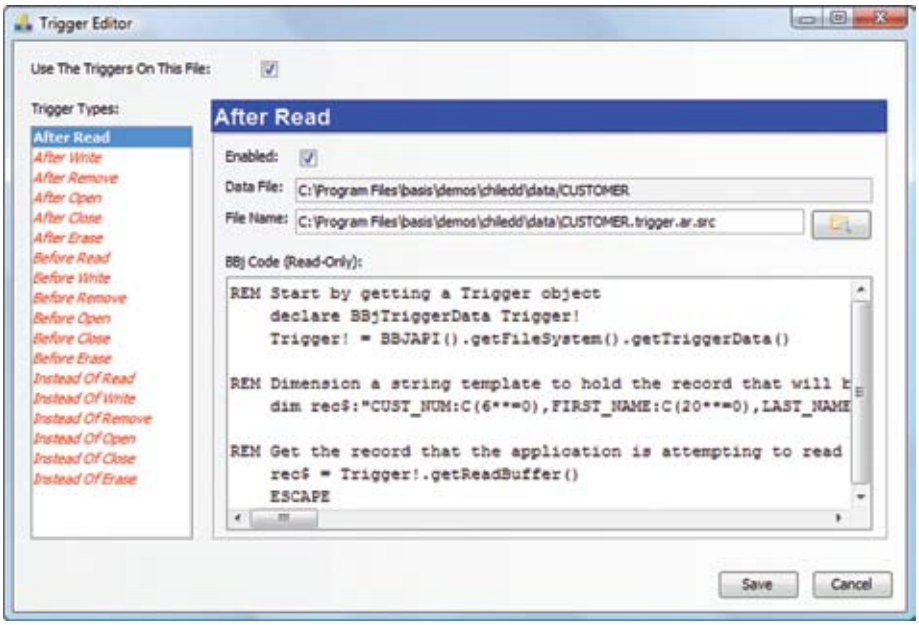

Figure 4. The After Read trigger definition

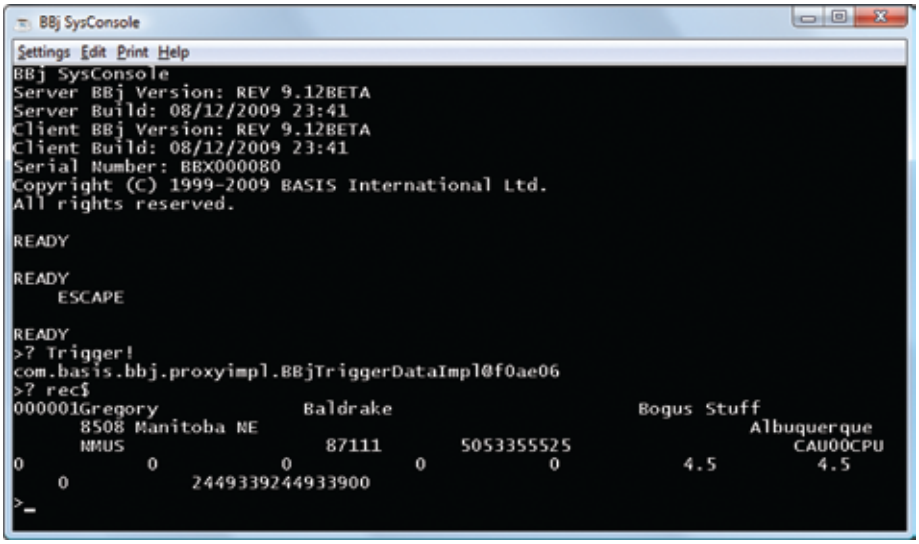

Figure 5. Interactively debugging the trigger code

Summary

In the past, determining what occurred in the inner workings of a SPROC or trigger program required a great deal of debug code and even more patience. BBj 9.0's new interactive debugging capability makes that a thing of the past as developers are free to debug their SPROCs and triggers in the same manner as all of their other application code.

The newfound ability to view and modify variable data, change program flow, and dot-step through the code ought to bring a sigh of relief to developers. We sure think so. ■