

# Putting Stored Procedures - and DBAs - to the Test: Part 2

William Vaughn

In Part 1 I pointed out that it's not just tables you have to grant privileges on, but also some stored procedures and even extended stored procedures. For example, I said that "If your developers want to step through or debug T-SQL stored procedures, you (or the DBA) will need to grant permission on the Extended Stored Procedure, *sp\_sdiidebug*" Now that the groundwork has been laid, let's write some code to execute a typical stored procedure. I've written several articles on executing stored procedures so I won't go into much detail here—and there's lots more detail in my book if you need more depth.

The following code creates a new SqlConnection object, a new SqlCommand object and populates the properties to permit us to execute a sample stored procedure. Notice that the name of the stored procedure is prefixed with the "owner" In this case, the account "developer" was used to create the stored procedure, so we use this name to address the stored procedure.

```
Dim cn As SqlConnection
Dim cmd As SqlCommand
Private Sub Form1_Load(ByVal sender _
As System.Object, ByVal e As System.EventArgs) _
Handles MyBase.Load
    Try
        cn = New SqlConnection("server=.;database=biblio;_
uid=developer;pwd=pw")
        cmd = New SqlCommand("Developer.StoredProcedure1",_
cn)
        With cmd
            .CommandType = CommandType.StoredProcedure
            .Parameters.Add("@StateWanted", _
txtStateWanted.Text)
            Parameters.Add("@AuthorCount", _
SqlDbType.Int).Direction = _
ParameterDirection.Output
        End With
    Catch ex As Exception
        MsgBox(ex.ToString)
    End Try
End Sub
```

The following code executes the stored procedure after having set the input parameter and captures the OUTPUT parameter it returns.

```
Private Sub btnQuery_Click(ByVal sender _
As System.Object, ByVal e As System.EventArgs) _
Handles btnQuery.Click
    Try
        cmd.Parameters("@StateWanted").Value = _
txtStateWanted.Text
        cn.Open()
        cmd.ExecuteNonQuery()
        txtAuthorsInState.Text = cmd.Parameters_
("@AuthorCount").Value.ToString
    Catch ex As Exception
        MsgBox(ex.ToString)
    Finally
        cn.Close()
    End Try
End Sub
End Class
```

Tip: For more detail on handling stored procedure parameters, see my website ([www.betav.com/articles.htm](http://www.betav.com/articles.htm)).

## Stepping through the procedure

I (generally) like to test my stored procedures before shipping applications that call them. Fortunately, Visual Studio .NET gives you two distinct ways to test stored procedures. First, if you've set up the permissions correctly, you can use the Visual Studio .NET IDE to step through a selected procedure. (This requires your developer account to have rights to *sp\_sdiidebug*.) To use this feature, right click on a selected stored procedure in the Server Explorer or in the Stored Procedure Edit window and choose "Step Into Stored Procedure". Yes, you can also simply run the stored procedure from this dialog by choosing "Run Stored Procedure". If the stored procedure has either input *or* OUTPUT parameters (or INPUT/OUTPUT parameters), the T-SQL debugger exposes a dialog to capture them. And no, Microsoft still hasn't fixed the problem with the data access

interfaces—Visual Studio .NET still can't tell the difference between OUTPUT and INPUT/OUTPUT parameters. If your stored procedure returns an OUTPUT parameter, you'll still have to provide a dummy value for the parameter or SQL Server will complain about missing parameter values. Once the Visual Studio NET IDE starts the stored procedure, you'll see a debugging window that exposes the stored procedure source code as well as a mechanism to fetch and alter local or @@global variables.

*Figure 1. Stepping through a T-SQL stored procedure.*

When you execute a stored procedure using the Visual Studio .NET IDE “step-into” technique, the IDE exposes a “Database Output” window to display any “print” statement output values or other messages sent from SQL Server.

### **Using the T-SQL debugger from code**

Another technique you can use with Visual Studio .NET to debug stored procedures is the T-SQL debugger. This approach is similar to the “step-into” approach just described, but in this case your code triggers execution of the T-SQL debugger. You'll need to enable another switch in your application for this to work—the T-SQL application property setting “T-SQL Debugging”. You'll find this switch by clicking on your Project Properties and choosing the Configuration Properties, SQL Server Debugging flag as shown in Figure 2.

*Figure 2. Setting the Application Debug Enable properties.*

Next, you'll need to open up the Server Explorer and the stored procedure(s) you want to debug. Drill down into the stored procedure logic and find the location in the code where you want the T-SQL debugger to break—returning control to your application. Without this configuration setting and the breakpoint, the Visual Studio .NET IDE won't open the T-SQL debugger window.

### **Examining the variables during the break**

Once the T-SQL debugger “breaks” (stops executing the T-SQL code at a selected breakpoint), you can examine the T-SQL parameters simply by hovering over them with the mouse cursor. You can examine local variables (such as @RowCount) or global variables (such as @@Connections) using the Command window and yes, you can also use the Command window to change the value of these variables. While there's no way to move “backward” in the code, you can slide the yellow arrow pointer forward to skip over portions of the logic. (The same function keys you use in Visual Studio .NET to step through code work here in the T-SQL debugger window as well.) For VB developers, this means F8 steps through the code line-by-line and F5 runs until the next breakpoint.

## **Summary**

In this pair of short articles, I've described how DBAs can expose all, some, little, or none of a DBMS to application developers - both to protect the database integrity and schema and to support better team development. I also showed you how to create and test stored procedures using both stored procedure “step-into” as well as code-launched T-SQL debugger sessions. I hope this helps both you and your DBA get along better.

## **Selected Resources**

Debugging SQL [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsdebug/html/\\_core\\_Debugging\\_SQL.asp](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsdebug/html/_core_Debugging_SQL.asp)

318632 FIX: SQL Debugging May Fail Because of Strict Local Password Complexity Policy

817253 PRB: Cannot Debug Stored Procedure in Visual Studio .NET After You Rename SQL Server

317241 PRB: Renaming a Host Computer for SQL Server Causes Stored Procedure Debug to Fail

URL [HTTP://MSDN.MICROSOFT.COM/LIBRARY/DEFAULT.ASP?URL=/LIBRARY/EN-US/VSDEBUG/HTML/\\_CORE\\_DEBUGGING\\_SQL.ASP](http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsdebug/html/_core_Debugging_SQL.asp)

bio on file