

A SQL Server Database and Object Explorer

Rick Dobson

One of the major deficiencies in Access data projects is the lack of tools for managing MSDE databases. Rick Dobson steps into the breach with an Access utility for exploring databases—and shows you how to use SQL-DMO on the way.

SINCE Office 2000, Access developers have, with Access Projects, a powerful and easy-to-use tool for creating true client/server applications. Developers have also, with one version or another of the MSDE, had a server database included with their copy of Access. However, while the MSDE comes with Access, the most popular Office editions don't ship with SQL Server Client Management tools that allow developers to manage their databases.

The SQL Server Client Management tools include the very useful Enterprise Manager—a graphical user interface for exploring databases and their objects on multiple servers. Access projects provide a Database window for showing database objects in the currently connected SQL Server database only. However, Access projects offer no graphical means of exploring the database objects within other SQL Server databases.

With ordinary Access projects, tables in server databases can only be viewed if they're linked ODBC data sources. You must explicitly link tables within one or more data sources for them to show in the Database window, preventing you from freely browsing the data sources that you're interested in. You can't, for instance, freely show all the tables from any selected database.

In this article, I'll show you how to create your own custom SQL Server database object explorer that you can run from either an Access project or an Access database. I'll start with a quick review of SQL-DMO (SQL Distributed Management Objects) programming. Then, I'll present a couple of Access form samples that use SQL-DMO programming to create custom SQL Server explorers.

Introduction to SQL-DMO

SQL-DMO is the programming environment that Microsoft used to create Enterprise Manager. It's also a COM object that you can invoke from the VBA project associated with any Access project or Access database. All you need to program SQL-DMO is a reference in your

reference list to the Microsoft SQLDMO Object Library (this is the name for SQL-DMO that appears in the References dialog box).

SQL-DMO has its own unique means of connecting to SQL Server instances. It can connect to the default instance on a computer as well as to named instances. A named SQL Server instance is another SQL Server installed on a computer after the initial installation of a default instance. SQL-DMO programming allows you to explore SQL Server instances without explicitly connecting to a database. You can use this feature to explore SQL Server instances that are visible to a computer before you connect to one or more instances.

The following code shows a simple VBA procedure for enumerating the SQL Server instances available to a computer. The ListAvailableSQLServers method of the SQLDMO object generates a list of SQL Server instances to which you can connect from your computer. The method returns a list of SQL Server instances in a NameList object. A NameList object is a container for strings. A NameList object has a Count property that indicates the number of string items in the object, with the first string at position 1 in the NameList. The sample declares the nml1 variable with a NameList type, and populates nml1 with the ListAvailableSQLServers method. The For loop in the procedure also demonstrates the syntax for extracting values from a NameList object (you can't use For...Each loops with NameList objects):

```
Sub NameAvailableSQLServers ()
Dim nml1 As SQLDMO.NameList
Dim int1 As Integer

Set nml1 = SQLDMO.ListAvailableSQLServers

'Write out server names
For int1 = 1 To nml1.Count
    Debug.Print nml1(int1)
Next int1

'Release nml1 resource
Set nml1 = Nothing

End Sub
```

This code prints all the SQL Server instances to which a computer can connect in the Immediate window. The following list of instances is from one of the computers in my office. It shows the copy of SQL Server running on my

computer as “(local)”. If you’re just starting to use SQL Server with MSDE 2000, this may be the only instance that’s visible from your computer. The default SQL Server instance on other computers will have the same name as the computer. For instance, if I ran this code on a computer called CAB2000, the copy of SQL Server running on that computer would be called “(local)”. From some other computer, the SQL Server instance running on CAB2000 appears as “CAB2000”.

I installed a standard edition of SQL Server as the default edition on CAB2000. Named instances appear with a two-part name that includes both the computer name and the instance name, separated by a backslash. The CAB2000\OFFICEDEVELOPER instance is the SQL Server that ships with the Office XP Developer edition on the CAB2000 computer:

```
(local)
209.133.228.82
CAB2000
CAB2000\OFFICEDEVELOPER
CAB2200
CAB2200\DEVEDITION
CABARMADA
CABLAT
CABLAT\CABLATDEVELOPER
CABSONY1
```

The four-part number in the list is a TCP/IP address that denotes a SQL Server instance maintained by an ISP for an association that I run. Members of the association can connect to this SQL Server instance through a proprietary application.

Logging on

You can log into a SQL Server instance using the Connect method of the SQLServer object in the SQL-DMO object model. Both the connection and the syntax for creating the connection are unique to SQL-DMO. There are two basic ways of invoking the Connect method:

- You can use Windows integrated security by setting a SQLServer object’s LoginSecure property to True. Windows integrated security uses the Windows login credentials of the person running the application to authenticate a user to a SQL Server instance.
- The default setting for the LoginSecure property is False, which requires you to provide a SQL Server login and its password in order to log on.

The Connect method takes as many as three optional arguments. The first argument is the name of the SQL Server instance that you want to connect to. The second argument is the SQL Server login account name for the SQL Server instance. The third argument is the login’s password.

In the SQL Server world, a login account just enables you to enter a SQL Server instance. Frequently, though, the login account has a user account with the same

name and that user account grants database access to an individual database on a SQL Server instance.

You don’t need the login and its password when connecting with Windows integrated security because SQL-DMO automatically uses the Windows login of the Windows user invoking the SQL-DMO code. Furthermore, if you’re connecting to the local default SQL Server instance on a computer, you can leave out the first argument also.

The next block of code shows two procedures that illustrate the syntax for using Windows integrated security on the local default instance as well as on the default instance of the CABSONY1 computer. Both procedures list the databases on the server after connecting to the server. The enumeration of database names in the Immediate window uses a For...Each loop that iterates through the Databases collection of the SQLServer object.

This procedure shows the syntax for using Windows integrated security to connect to the database:

```
Sub ListDatabasesOnLocalSQLServer()
Dim srv1 As SQLDMO.SQLServer
Dim dbs1 As SQLDMO.Database

'Login to local SQL Server with Integrated security
Set srv1 = New SQLDMO.SQLServer
srv1.LoginSecure = True
srv1.Connect

'List database names
For Each dbs1 In srv1.Databases
    Debug.Print dbs1.Name
Next dbs1

'Release srv1 resource
srv1.Close
Set srv1 = Nothing

End Sub
```

This procedure shows the syntax for specifying the SQL Server instance name along with a login account name and its password:

```
Sub ListDatabasesOnRemoteSQLServer()
Dim srv1 As SQLDMO.SQLServer
Dim dbs1 As SQLDMO.Database

'Login to CCS1 SQL Server with SQL Server login
Set srv1 = New SQLDMO.SQLServer
srv1.Connect "CABSONY1", "SmartAccess", "password"

'List database names
For Each dbs1 In srv1.Databases
    Debug.Print dbs1.Name
Next dbs1

'Release srv1 resource
srv1.Close
Set srv1 = Nothing

End Sub
```

Exploring SQL Server databases

Now that you have a grasp of the SQL-DMO syntax for listing the databases on a server, you may be

wondering how you expose this functionality through an Access form. Figure 1 shows the frmListDatabasesOnSQLServer form in the sample Access project that's included in this month's Download file (it's available at www.vb123.com/kb/). I have the project connected to the NorthwindCS database that ships with Office, but the Access project can run disconnected. Feel free to copy the code and forms into an Access database, and run the samples from there. The code will still work because the code manages its own connections to SQL Server instances and doesn't depend on Access for its connections.

Figure 1 shows the form with a connection to the (local) SQL Server instance. Since the form connects to this instance, and my Windows login has permission to enumerate databases, there's no need to specify a login name and its password. Clicking the List Databases button populates the list box with the names of databases from the server name selected in the combo box.

The form's combo box lists the names of all SQL Server instances that your computer can connect to. If the Windows login for the user running the form is recognized by SQL Server on another computer with appropriate permission to enumerate databases, the application will connect successfully. Otherwise, an error message will appear if the SQL Server login and password text boxes aren't right. As an alternative, you can use the name of the sa login and its password—the sa login will always have permission to list database names. The text box for the password has an Input Mask property setting of Password that causes asterisks to appear instead of the actual password characters.

The following routines are the code behind the form in Figure 1. The declaration at the top of the listing specifies a module-level variable (strSQLServerName) for the name of the currently selected SQL Server instance. This variable is set in two places. The Form_Load event procedure initializes the variable to (local), the local default SQL Server instance. The Form Load event also loads the list box of available servers.

The Form_Load event procedure begins by loading a NameList object (nml1) with the names of all SQL Server instances available to the computer running the Access form. The application will populate both the combo box and the list box with the resulting string values. In order to do this, the RowSourceType property settings for the controls must be set to Value List. Next, a For loop copies the values with the SQL Server instance names from nml1 to the select list for the cboSQLServers combo box.

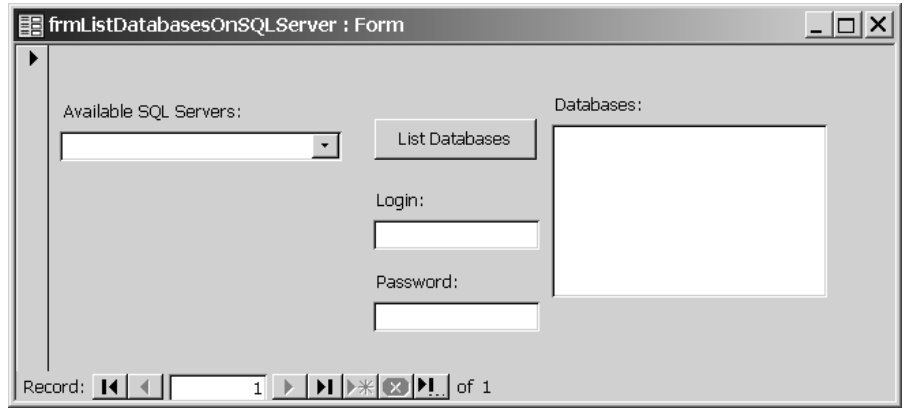


Figure 1. An Access form that lists the databases on a server.

Then, the application selects the first name in nml1 for the combo box and saves that value in strSQLServerName. The procedure closes by releasing the resource for the NameList variable and doing minor formatting for the form:

```
Dim strSQLServerName As String

Private Sub Form_Load()
    Dim nml1 As SQLDMO.NameList

    'Instantiate NameList object with available servers
    Set nml1 = SQLDMO.ListAvailableSQLServers

    'Assign Value List to RowSourceType property
    'for combo box and list box
    Me.cboSQLServers.RowSourceType = "Value List"
    Me.lstDatabases.RowSourceType = "Value List"

    'Assign server names to cboSQLServers selection list
    For int1 = 1 To nml1.Count
        Me.cboSQLServers.AddItem (nml1(int1))
    Next int1

    'Assign first SQLServer to selected combo box
    'item and save the value for use elsewhere
    Me.cboSQLServers.Value = nml1(1)
    strSQLServerName = Me.cboSQLServers.Value

    'Release nml1 resource
    Set nml1 = Nothing

    'Edit form layout
    Me.RecordSelectors = False
    Me.NavigationButtons = False
    Me.DividingLines = False

End Sub
```

The user can change the selected server at any time by selecting a new SQL Server instance from the combo box (cboSQLServers). The cboSQLServers_AfterUpdate event procedure sets the value for the strSQLServerName variable if the user selects a server from the list box procedure:

```
Private Sub cboSQLServers_AfterUpdate()

    'Save selected combo box item for use elsewhere
    strSQLServerName = Me.cboSQLServers.Text

End Sub
```

The cmdListDatabases_Click event procedure is the other major procedure in the application and builds on my two previous routines for connecting to a database. This procedure starts by logging into the SQL Server instance selected in the cboSQLServers combo box. Depending on whether the Value property is Null for the txtLogin text box, the procedure uses either Windows integrated security or a SQL Server login and its password when connecting to the SQL Server instance. After logging in, the procedure uses a Do loop to remove any previously existing items from the list box. Next, the application adds database names to the list box with a For...Each loop. By erasing prior items before adding new ones, the application ensures that just databases from the currently selected SQL Server instance appear in the list box. A very simple error trap at the end of the procedure informs the user of an unanticipated error. This allows the application to retain control so that the user can make another set of entries on the form:

```
Private Sub cmdListDatabases_Click()
On Error GoTo ListDatabasesTrap
Dim srv1 As SQLDMO.SQLServer

'Declare and instantiate SQLServer object
Set srv1 = New SQLDMO.SQLServer

'Login with Integrated security or with
'SQL Server login
If IsNull(txtLogin.Value) Then
    srv1.LoginSecure = True
    srv1.Connect strSQLServerName
Else
    srv1.Connect strSQLServerName, txtLogin.Value, _
        txtPassword.Value
End If

'Clear list box
Do Until lstDatabases.ListCount = 0
    Me.lstDatabases.RemoveItem (0)
Loop

'Add database names to list box
For Each dbs1 In srv1.Databases
    Me.lstDatabases.AddItem (dbs1.Name)
Next dbs1

ListDatabasesExit:
Exit Sub

ListDatabasesTrap:
MsgBox "Unanticipated error. " & vbCrLf & _
    "Error number: " & Err.Number & vbCrLf & _
    "Error description: " & Err.Description

End Sub
```

Exploring a database

The second SQL Server explorer uses the form

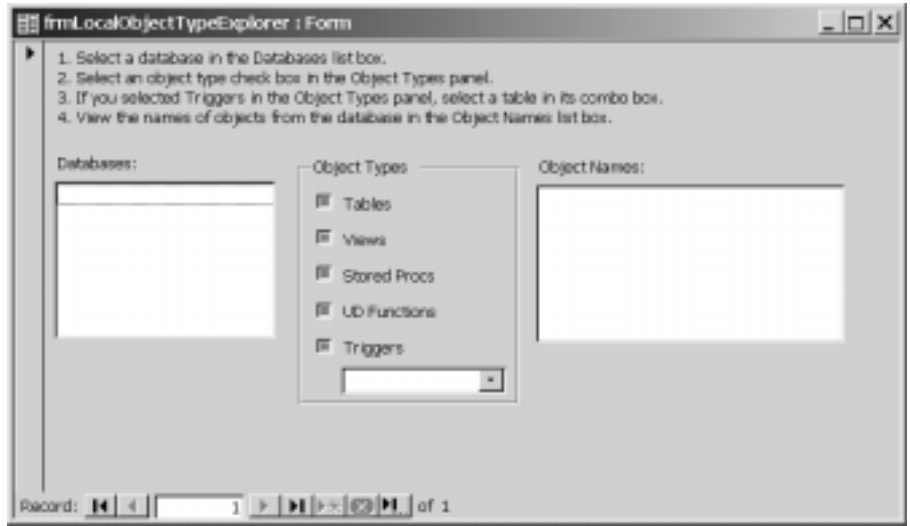


Figure 2. An Access form that lists the database objects from any database in the local default SQL Server instance on a computer.

frmLocalObjectTypeExplorer (see Figure 2) in the SA0403 Access project. This explorer focuses exclusively on the local default SQL Server instance (as if you installed just MSDE 2000 on a computer). You can use the explorer to select any database on the local default SQL Server instance. Once you've selected the database, you can select a type of object. After that, the form automatically returns the names of all of the objects of that type in the database. You can cause the list box on the right of the form to show the tables, views, stored procedures, user-defined functions, or triggers for a designated table.

The first time that you open the NorthwindCS Access project, it loads the NorthwindCS database in the local copy of SQL Server. Figure 3 (on page 12) shows the form displaying the NorthwindCS database (you can install the NorthwindCS Access project and open it after installing MSDE 2000 on a computer). As you can see from Figure 3, the NorthwindCS database includes tables named similarly to the familiar Northwind Jet database. SQL Server views and stored procedures are roughly equivalent to queries in the Access user interface.

User-defined functions and triggers aren't likely to be familiar objects to those who are used to working with Jet databases. User-defined functions operate somewhat similarly to functions in VBA, except you must write them in Transact-SQL. Triggers are like event procedures for tables or views that can fire whenever a user updates a column value or inserts or deletes a row.

The explorer implemented by the form frmLocalObjectTypeExplorer also lets you select a table to reveal any triggers associated with it. There are no user-defined functions or triggers defined by default for the NorthwindCS database, so you won't be able to use that

feature in the NorthwindCS database. However, the employee table in the SQL Server sample pubs database has a trigger named employee_insupd. You can get the pubs database with the SQL Server edition that ships with Office XP Developer edition or any commercial version of SQL Server but not, unfortunately, with MSDE 2000. Chapter 14 of my book *Programming Microsoft Access Version 2002* includes a thorough review of the SQL Server object types available from this explorer and how to create those objects from an Access project.

The following code is from the form in Figure 3. The listing includes two module-level declarations and three event procedures. The module-level declarations are for SQLServer (srv1) and Database (dbs1) SQL-DMO object types that are used by several procedures:

```
Dim srv1 As SQLDMO.SQLServer
Dim dbs1 As SQLDMO.Database
```

The main objective of the Form_Load event procedure is to populate the list box (lstDatabases) showing the database names on the left side of the form. The procedure does this in three steps. First, the code connects to the local default SQL Server instance. Second, it assigns Value List as the RowSourceType property setting for the lstDatabases list box. While the procedure sets this property for the lstDatabases control, it also sets the property to the same value for two other controls referenced in other procedures. Third, the procedure iterates through the Databases collection of the local SQL Server instance and adds the name of each database to the select list for the lstDatabases control:

```
Private Sub Form_Load()

'Login to local SQL Server with Integrated security
Set srv1 = New SQLDMO.SQLServer
srv1.LoginSecure = True
srv1.Connect

'Assign Value List to the RowSourceType
'property for the lstDatabases, cboTables, and
'lstObjNames controls
Me.lstDatabases.RowSourceType = "Value List"
Me.cboTables.RowSourceType = "Value List"
Me.lstObjNames.RowSourceType = "Value List"

'List database names in lstDatabases control
For Each dbs1 In srv1.Databases
    Me.lstDatabases.AddItem (dbs1.Name)
Next dbs1

'Edit form layout
Me.RecordSelectors = False
Me.NavigationButtons = False
Me.DividingLines = False

End Sub
```

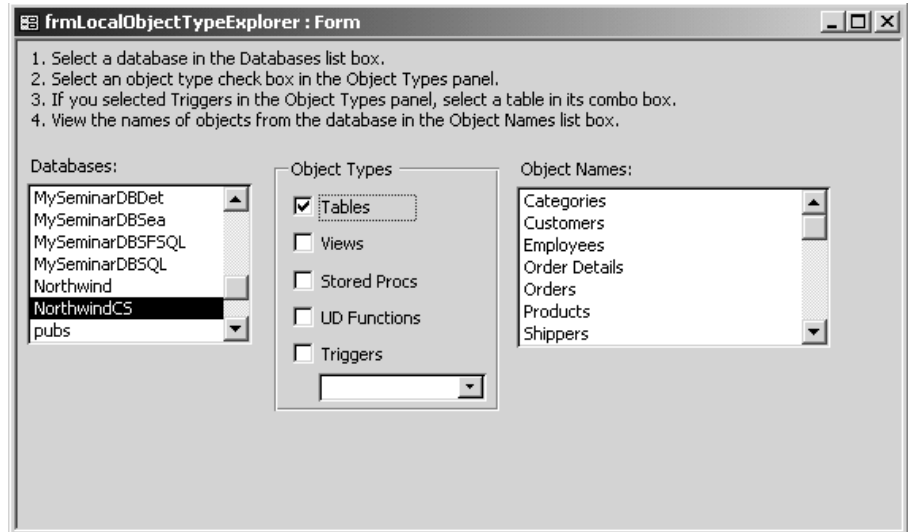


Figure 3. The same form displaying the NorthwindCS database.

The optType_AfterUpdate procedure fires after the user makes a selection from the option group control (optType) in the center of the form. If a user attempts to make a selection from the optType control without first selecting a database from lstDatabases, the application displays a reminder to do so in a message box. Before processing the value of the optType control, the procedure clears the selection lists for the list box on the right of the form (lstObjNames) and the combo box (cboTables) below the bottom check box in the option group:

```
Private Sub optType_AfterUpdate()
Dim tbl1 As SQLDMO.Table

'Remind about selecting a database,
'if necessary
If IsNull(lstDatabases.Value) Then
    MsgBox "Select a Database."
    Exit Sub
End If

'Clear Object Name list box
Do Until lstObjNames.ListCount = 0
    Me.lstObjNames.RemoveItem (0)
Loop

'Clear cbo_Tables combo box and display value
Do Until cboTables.ListCount = 0
    Me.cboTables.RemoveItem (0)
Loop
Me.cboTables.Value = ""
```

The optType control assumes a unique value for each of the database objects represented by the check box controls. The value 1 represents the selection of the check box with a caption of Tables, and the value 5 denotes the selection of the check box for Triggers. A Select Case statement processes the optType values. The code for the Tables, Views, and Stored Procedures collections merely iterates through the collections for the selected database in the lstDatabases control. The code enumerating the user-defined functions instantiates a new SQLServer object (srv2). This object is based on a different class type

than that used for enumerating the other database objects because user-defined function collections are only available in SQLServer2 classes, as opposed to SQLServer classes. SQLServer class instances are generally preferable to SQLServer2 class instances. This is because SQLServer class instances work with version 7 and version 2000 of SQL Server, but SQLServer2 class instances work only with SQL Server 2000:

```

Set dbs1 = srv1.Databases (Me.lstDatabases.Value)

'Process selection from optType control
Select Case Me.optType
  Case 1
    For Each obj1 In dbs1.Tables
      If obj1.SystemObject = False Then _
        lstObjNames.AddItem (obj1.Name) Next obj1
  Case 2
    For Each obj1 In dbs1.Views
      If obj1.SystemObject = False Then _
        lstObjNames.AddItem (obj1.Name) Next obj1
  Case 3
    For Each obj1 In dbs1.StoredProcedures
      If obj1.SystemObject = False Then _
        lstObjNames.AddItem (obj1.Name) Next obj1
  Case 4
    'Must use database with user-defined functions
    Set srv2 = New SQLDMO.SQLServer2 srv2.Connect "(local)", "sa", "password"
    Set dbs2 = srv2.Databases (Me.lstDatabases.Value) Dim udf2 As SQLDMO.UserDefinedFunction
    For Each udf2 In dbs2.UserDefinedFunctions lstObjNames.AddItem (udf2.Name)
    Next udf2
  Case 5
    'Re-fill cboTables
    For Each obj1 In dbs1.Tables
      If obj1.SystemObject = False Then _ Me.cboTables.AddItem (obj1.Name)
    Next obj1
End Select

End Sub

```

The Case clause for the Trigger value 5 of the optType control merely populates the cboTables combo box with table names. The application doesn't populate the lstObjNames list box until the user selects a table from the cboTables control. Each table in a SQL Server database can have its own Triggers collection. The cboTables_AfterUpdate procedure populates the lstObjNames control with the names of any triggers for the currently selected table in the cboTables control. Users can successively designate tables to view the triggers for each table:

```

Private Sub cboTables_AfterUpdate() Dim trgl As SQLDMO.Trigger
Dim int1 As Integer

'Clear Object Name list box
Do Until lstObjNames.ListCount = 0
  Me.lstObjNames.RemoveItem (0) Loop

'Point dbs1 at currently selected database
Set dbs1 = srv1.Databases (Me.lstDatabases.Value)

```

```
'Add trigger names for the selected table in the
'cboTables control to the lstObjNames list box
For int1 = 1 To _
    srv1.Databases(Me.lstDatabases.Value) . _ Tables(Me.cboTables.Value) .Triggers.Count
    lstObjNames.AddItem _ (srv1.Databases(Me.lstDatabases.Value) . _
        Tables(Me.cboTables.Value) .Triggers(int1) .Name)
Next int1

End Sub
```

Conclusion

In this article I've shown you the tools for creating custom explorers for SQL Server database objects. This kind of tool is particularly convenient for any user of MSDE 2000, since this edition of SQL Server 2000 doesn't ship with any SQL Server Client Management Tools (such as Enterprise Manager). In addition, you can use custom explorers in your MSDE 2000 applications to show users the resources available to them. ▲



[SQLEXP1.ZIP at www.vb123.com/kb/](http://www.vb123.com/kb/)

Rick Dobson is an author/trainer/Webmaster. His practice, CAB, Inc., sponsors national and Web-based seminar tours. His most recent two books are *Programming Microsoft Visual Basic .NET for Microsoft Access Database* and *Programming Microsoft SQL Server 2000 with Microsoft Visual Basic .NET*. Rick is an MCP for Visual Basic .NET.

You can learn more about his practice, books, and seminars at www.programmingmsaccess.com. rickd@cabinc.net.