

The Oracle help in Visual Studio .NET is not bad. There is good coverage of basic Oracle idiosyncrasies, including how Oracle treats case sensitivity and naming conventions. Look in **Help** under **Server Explorer | Oracle** for more information.

Depending on which analyst report you believe, there are anywhere from 3.2 to 6.4 million Visual Studio developers at large who use an Oracle database on the backend.

This is a very interesting statistic given that Oracle really wants a larger portion of the application development market and Microsoft really wants a larger portion of the database market. It seems that there are many developers who have identified the strengths of each technology and use them accordingly.

Developing solutions that incorporate software from various vendors can be challenging—especially if the vendors in question are fierce competitors. With many Microsoft developers making the transition to .NET, and with a seemingly great number of those migrants developing against an Oracle backend, the need for technical information on Oracle data access from .NET is clear. This paper introduces the “Server Explorer” utility within the .NET development environment, used for browsing database objects. It gives some code examples, showing how to access and manipulate those objects, and comments on the functionality we wish we had from within the .NET environment.

Browsing the Oracle database with the Server Explorer

In Visual Studio .NET, Server Explorer replaces the former Data View window. It provides many of the same capabilities found in the Enterprise Manager from SQL Server. (Only we'll be using this functionality on an Oracle Database.) You have the ability to browse table structures, check constraints and even retrieve data. You also can create tables and views and design database diagrams. You can even write stored procedures in PL/SQL and compile them on the database. These things are very useful to the developer designing an application that utilizes a database. You can quickly and easily see the objects at your disposal or identify objects that should exist and create them. Regarding Oracle, it sure beats writing queries against System tables at the command line inside SQL*Plus.

Go to the **View** menu and choose **Server Explorer** or, for those of you who don't like to take your hands off the keyboard, hit Ctrl+Alt+S.

Opening the Server Explorer for the first time, you can see any local SQL Server databases. In order to connect to an Oracle database, you've got some work to do.

Step by step:

1. Go to the **Tools** menu and choose **Connect to Database ...**
2. This leaves you on the **Connection** tab. (If you want to connect to Oracle, you need to be on the **Provider** tab.)
3. Choose the **Provider** tab on the left.
4. In the **Provider** tab, you will choose the OLE DB driver you'll use to access the database. Unless you've explicitly installed your own OLE DB drivers for Oracle, you'll likely have a choice between two. One driver is provided by Microsoft and the other by Oracle. Contrary to what one might think, the OLE DB driver provided by Microsoft seems to provide a more detailed view of the database, breaking database objects down categorically. The Oracle OLE DB driver groups objects into merely Tables, Views and Stored Procedures, making no distinction between package bodies, procedures, functions and so on.
5. Choose the Microsoft OLE DB Provider for Oracle.
6. Click **Next**.
7. Enter the server name. This could be a little confusing, as you will really enter the service name specified in your tnsnames.ora file.
8. Enter your username and password (SCOTT/TIGER).
9. Browse through your database objects and take a look at the context-sensitive functionality by right-clicking on an object.

Database driver comparison

Oracle Driver	Microsoft Driver
Shows tables, views, procedures.	Shows database diagrams, tables, synonyms, views, stored procedures, functions, package specifications and package bodies.
Prompt for login: YES The Oracle driver always prompts for a password, which is good if you are concerned with security.	Prompt for login: NO.
No ancillary tables created.	Creates a table called "MICROSOFTDTPROPERTIES."
Create a table: NO	Create a table: YES, including creating primary keys, indexes and constraints. See Figure 1.
Create procedures, functions, package specifications: NO.	Create procedures, functions, package specifications: YES, including templates for each. See Figure 2.
Create view: NO.	Create view: YES.
Create database diagram: NO.	Create database diagram: YES.

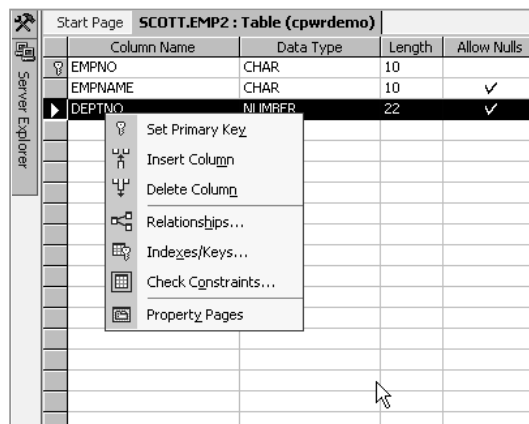


Figure 1:
Table creation interface.

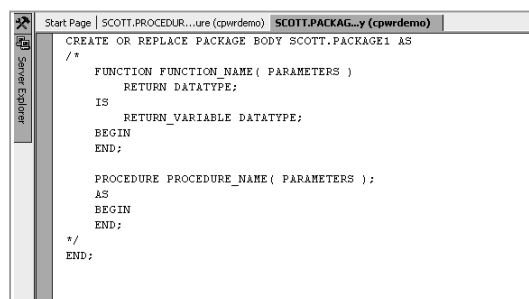


Figure 2:
PL/SQL templates supported.

Tip: If you don't feel like typing the connection string by hand, go to the Server Explorer and right-click on an established connection, selecting Properties. This will populate the Properties window with the attributes for the connection. One of these attributes is the connection string, which you can cut and paste into your code. (See Figure 3)

Accessing data from your .NET code

Accessing Oracle data from your .NET code will involve using ActiveX Data Objects for the .NET Framework or ADO.NET. In essence, ADO.NET is simply a set of classes that you can use to access data. These classes can be accessed by including the System.Data namespace in your .NET code. Doing this exposes the OLE DB namespace, which will allow you to access Oracle.

This paper covers three objects exposed in these classes that are used to manipulate data:

```
OleDbConnection  
OleDbCommand  
OleDbDataReader
```

First, let's examine the Connection Object:

The Connection Object allows you to establish a connection to the data. In our case, since we are accessing Oracle, we will be using the connection object called OleDbConnection. The syntax looks like this:

```
` Create the variable  
Dim conToOracle as OleDb.OleDbConnection  
` Create the actual Object  
conToOracle = new OleDb.OleDbConnection()  
` Set the ConnectionString conToOracle.Open()  
conToOracle.ConnectionString =  
"Provider=MSDAORA.1;Password=tiger;User ID=scott;Data  
Source=cpwrdemo"
```

Here we are creating the Connection Object and setting its connection string so that when the .Open() method is invoked, a connection can be established. The connection string consists of name/value pairs needed to establish a connection. Note that we are using the same OLE DB provider that we used to connect to Oracle in the Server Explorer. In the string, we are simply specifying the information that the Server Explorer prompted us for.

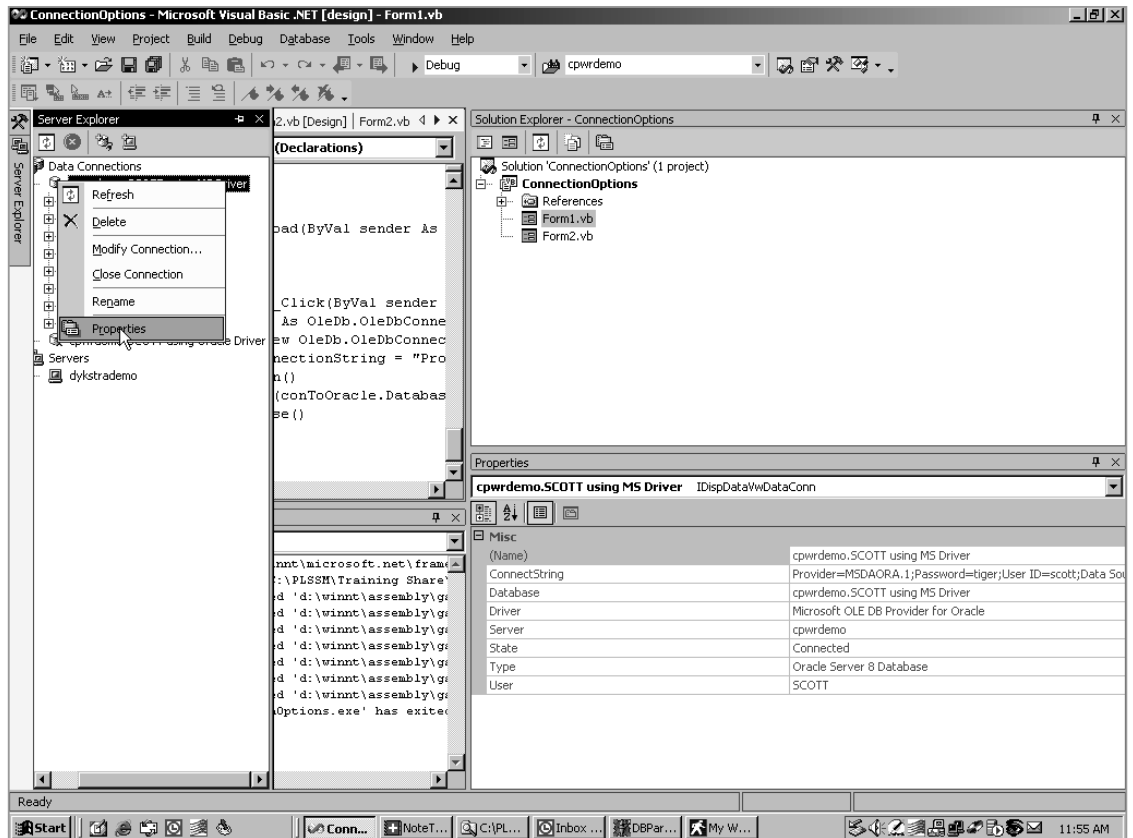


Figure 3: Connection Object properties window.

Now let's look at the Command Object:

The Command Object is used to issue a command to the database and optionally return data from it. It executes any DML, DDL or DCL statement held in the CommandText property of the command object you create. There are two ways to create a Command Object:

1. Use the Command Constructor passing in the Connection Name as an argument. Syntax:

```
Dim ACommandObject As System.Data.OleDb.OleDbCommand
ACommandObject = New
System.Data.OleDb.OleDbCommand("Select ENAME from
EMP; ")
```

2. Use the CreateCommand method of your Connection Object. Syntax:

```
Dim ACommandObject As System.Data.OleDb.OleDbCommand
ACommandObject = conToOracle.CreateCommand()
ACommandObject.CommandText = "Select ENAME
from EMP;"
```

Once the Command Object is created, there are several useful methods to invoke.

.ExecuteScalar is used when a single record is expected as a result of the query. This is useful when there is an aggregate function in the statement that returns a single value in the first column. Note: Any data not in the first row and first column will be discarded.

.ExecuteNonQuery is used for statements that don't return a value, such as DCL or INSERT statements.

.ExecuteReader is used to execute a query that returns multiple records. The records are returned in the form of an OleDbDataReader object. (See OleDbDataReader section.)

The Command Object can also be used to execute a stored procedure. This is done by creating a Command Object and setting its CommandType property to "Stored Procedure." Also, set the name of the procedure in the CommandText property of the Command Object. If the stored procedure takes parameters, then parameter objects can be created and added to the Command Object. Finally, if the stored procedure returns a result set, a DataReader can be used to iterate through the results returned by the stored procedure. Here is an example:

```
Dim conToOracle As OleDb.OleDbConnection
conToOracle = New OleDb.OleDbConnection()
conToOracle.ConnectionString =
"Provider=MSDAORA.1;Password=tiger;User ID=scott;Data
Source=cpwrdemo"
conToOracle.Open()
Dim OracleComm As OleDbCommand = New
OleDbCommand()
OracleComm.Connection = conToOracle
` Set the CommandType property to StoredProcedure
OracleComm.CommandType =
CommandType.StoredProcedure
OracleComm.CommandText = ("PROCEDURE1")

` Create the parameters for the Stored Procedure
Dim param1 As New OleDb.OleDbParameter()
param1.Direction = ParameterDirection.Input
`input parameter param1.Value = "40"
OracleComm.Parameters.Add(param1)

` Create a DataReader object to handle the
returned data

Dim OrcDataReader As OleDb.OleDbDataReader
OrcDataReader = OracleComm.ExecuteReader()

` From here you can iterate through the records
in the OrcDataReader object (See OleDbDataReader
section)
```

Tip: If your stored procedure is only returning one record, you will improve performance by returning the value in the form of an out parameter. For example:
`param2.Direction = ParameterDirection.Output`

Now let's look at the `OleDbDataReader` object:

The `OleDbDataReader` object is used to hold the results of a query or stored procedure. It is read-only, forward-only, and thus is useful for a set of records that will be iterated through once. It does this very efficiently as only one record is held in memory at any given time. You can instantiate an `OleDbDataReader` object by using the `ExecuteReader` method of the `OleDbCommand` object. After you have a `OleDbReader` object, you can iterate through the records using the `.Read` method. This method returns a `False` when there are no more records in the set. Here is an example:

```
Dim OrcDataReader As OleDbDataReader
    Try
        Dim OracleComm As OleDbCommand = New
OleDbCommand("Select ENAME as ENAME from
EMP;")OrcDataReader = OracleComm.ExecuteReader()
        lstEmployees.Items.Clear()
        While (OrcDataReader.Read() = True)
            lstEmployees.Items.Add
                (OrcDataReader("ENAME"))
        End While

    Catch excp As Exception
        MessageBox.Show(excp.ToString())

    End Try
OrcDataReader = Nothing
```

Conclusion and comments

Microsoft does a satisfactory job of allowing developers to utilize an Oracle database for their applications. By supporting the OLE DB interface, developers can use both GUI features within the development environment to see objects in the Oracle database, and classes included in the .NET Framework to access those objects programmatically. Visual Studio .NET exposes many cool development aids such as context-sensitive help, auto-fill and mouseover variable evaluation during debugging. Microsoft even provides PL/SQL templates for creating stored procedures on the Oracle database.

The one noticeable shortcoming is that one cannot *debug* those stored procedures. Stored procedures are still widely used to centralize business logic, ensure transactional integrity and—most importantly, in the fast paced world of e-business—improve performance. In fact, this benefit may be why you chose to go with Oracle in the first place. But with all the benefits possible using stored procedures, none are worthwhile if the code is buggy. Unfortunately, when a stored procedure is called, none of the Visual Studio

Compuware DevPartnerDB Debugger for Oracle *Visual Studio Edition* provides developers with the unique ability to perform proactive analysis and debugging of PL/SQL during application development and completely within the Visual Studio .NET environment. Without the Debugger for Oracle, any PL/SQL called by the application code you are debugging is executed in the background, without being accessible through the Visual Studio .NET environment.

When integrated into Visual Studio .NET, DevPartnerDB Debugger for Oracle provides a subset of the functionality available in the DBPartner product suite. The DBPartner Debugger also includes a lock monitor, database browser and search window, text editor with syntax templating, table editor and hookup mechanism for debugging stored procedures when they are executed in an external application.

You can find more information on the DevPartnerDB Debugger for Oracle *Visual Studio Edition* at <http://www.compuware.com/products/numega/dbpartner/index.htm>.

debug features are at your disposal. The procedure merely executes with the parameters you specified and it either works or doesn't work. This is true even when using SQL Server. Fortunately, Microsoft was wise enough to make the .NET Framework and Visual Studio itself flexible and extensible. This leaves the door open for Microsoft to add this key functionality themselves or collaborate with a partner to add features that help with stored procedure debugging.

System as tested

Oracle Database 8.1.7

OLE DB Providers:

- Microsoft OLE DB Provider for Oracle
- Oracle Provider for OLE DB 8.1.7.0.0

Release Candidate 7 build 9466 - Microsoft Development Environment 7.0.9466
.NET Framework 1.0.3705

Compuware products and professional services—
delivering quality applications

Compuware is a leading global provider of software products and professional services which IT organizations use to develop, integrate, test and manage the performance of the applications that drive their businesses. Our software products help optimize every step in the application life cycle—from defining requirements to supporting production service levels—for web, distributed and mainframe platforms. Our services professionals work at customer sites around the world, sharing their real-world perspective and experience to deliver an integrated, reliable solution.

Please contact us to learn more about how our comprehensive products and services can help your organization improve productivity, create higher quality applications and ensure performance in production.

All Compuware products and services listed within are trademarks or registered trademarks of Compuware Corporation. All other company or product names are trademarks of their respective owners.
© 2002 Compuware Corporation



www.compuware.com

References

Oracle 9i PL/SQL Programming by Scott Urman, McGraw Hill/Osborne 2002.

Programming with Microsoft Visual Basic .NET, Microsoft Official Curriculum Courses 2415ACP & 2373ACP.

IDC and Giga Analyst Conference Calls.

SQL Server Magazine at www.sqlmag.com, "Set Sail with Visual Studio .NET," Article ID 23457, Michael Otey.