OpenAccess

OLE DB Provider SDK and .NET

Technology Overview

January 10, 2003

Written by

Automation Technology, Inc.
2001 Gateway place, Suite 100.
San Jose, CA 95110

automation technology, inc.

OpenAccess OLE DB Provider SDK & .NET

Part # OA-SDK.NET-DOC

Table Of Contents

Overview

The objective of ATI's OpenAccess SDK has always been to provide a solution for enabling standardized access to any data on any platform. With the upcoming release of Microsoft's .NET platform ATI has been formulating its strategy to enable access from ADO.NET to any data source. This document describes how OpenAccess allows you to provide access to your data through ADO.NET using our current technology and our planned future products.

OpenAccess is designed to hide the developer from the complexities of SQL, ODBC, OLE DB, JDBC, or networking. The developer focuses on writing a set of functions that deal with access to his or her data. OpenAccess then handles the details of exposing this data through ODBC, OLE DB, or JDBC without any additional coding. Our solution for .NET forces you to learn nothing new about Microsoft Windows or .NET. You simply write an Interface Provider to link your data source to our SQL engine. Your code can run on the .NET platform or on any of our other supported platforms. Today you can plug into ADO.NET through Microsoft's .NET Data Provider for OLE DB. This means that today you can use our OLE DB SDK to create a fully functional .NET Data Provider for your data source.

Our future plans are to support a native OpenAccess .NET Data Provider SDK. Today we feel that combination of Microsoft's support for OLE DB and our mature OLE DB SDK offer the best solution. Within days you can have a .NET Data Provider to your data source. Please check our web site http://www.atinet.com for OpenAccess .NET Data Provider SDK product availability.

ADO.NET works with provider developed using OpenAccess OLE DB SDK. In this document we have given examples of ADO.NET tested with OpenAccess OLE DB SDK. Please contact support@atinet.com or sales@atinet.com for more information. You can read more about our products at http://www.odbcsdk.com

The Microsoft ADO.NET is supported on Microsoft® Windows® 2000, Microsoft® Windows NT® 4 with Service Pack 6a, Microsoft® Windows® Millennium Edition, Microsoft® Windows® 98, Microsoft® Windows® SE, and Microsoft® Windows® 95. Use of the OLE DB .NET Data Provider requires the installation of Microsoft Data Access Components 2.6 or later.

OpenAccess OLE DB SDK & .NET Architecture

The following block diagram shows interaction between a .NET application and the OpenAccess OLE DB Data Provider through the Microsoft OLE DB .NET Data Provider. In client/server configuration, the OpenAccess OLE DB Data Provider is shipped as a shrink-wrapped component that is installed on Windows platforms. It communicates with the OpenRDA Server to execute the queries on the server platform. In local configuration, the OpenAccess OLE DB Data Provider library is linked with the DAM and your IP to build a OLE DB compliant COM object specific for your data source.

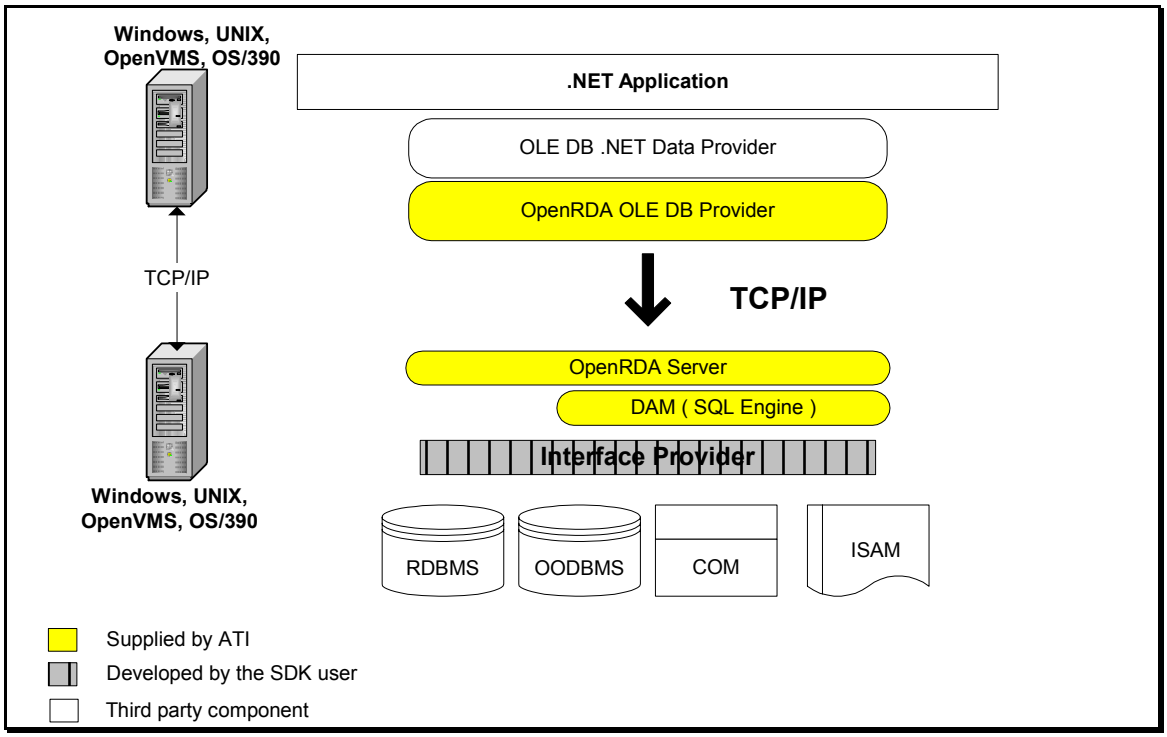

Figure 1: Client/Server Architecture

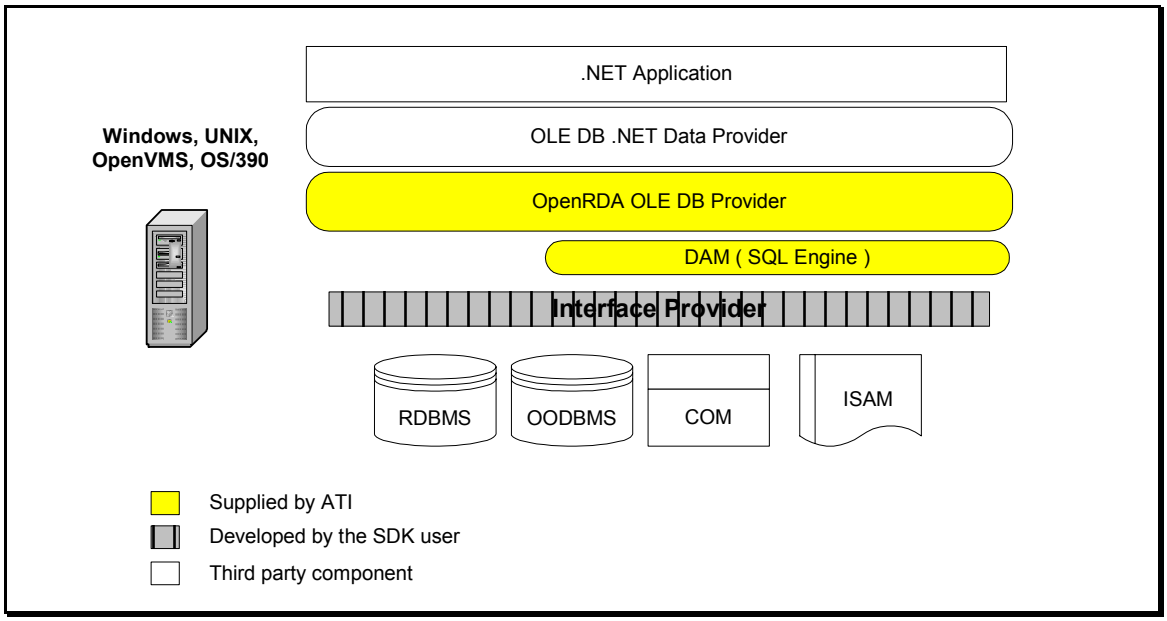

Figure 2: Local Architecture

.NET Data Providers

A .NET data provider is used for connecting to a database, executing commands, and retrieving results. Those results are either processed directly, or placed in an ADO.NET **DataSet** in order to be exposed to the user in an ad-hoc manner, combined with data from multiple sources, or remoted between tiers. The .NET data provider is designed to be lightweight, creating a minimal layer between the data source and your code, increasing performance while not sacrificing functionality. There are four core objects that make up a .NET data provider:

Object	Description
Connection	Establishes a connection to a specific data source.
Command	Executes a command at a data source. Exposes **Parameters** and can enlist a **Transaction** from a **Connection**.
DataReader	Reads a forward-only, read-only stream of data from a data source.
DataAdapter	Populates a **DataSet** and resolves updates with the data source.

The .NET Framework includes the OLE DB .NET Data Provider

The OLE DB .NET Data Provider

The OLE DB .NET Data Provider uses native OLE DB through COM interop to enable data access. The OLE DB .NET Data Provider supports both manual and automatic transactions. For automatic transactions, the OLE DB .NET Data Provider automatically enlists in a transaction and obtains transaction details from Windows 2000 Component Services.

To use the OLE DB .NET Data Provider, you must also use an OLE DB provider. The following providers are compatible with ADO.NET.

Driver	Provider
SQLOLEDB	Microsoft OLE DB Provider for SQL Server
MSDAORA	Microsoft OLE DB Provider for Oracle
Microsoft.Jet.OLEDB.4.0	OLE DB Provider for Microsoft Jet
OpenRDA/LocalOpenRDA	OpenAccess OLE DB Provider

The OLE DB .NET Data Provider does not work with the OLE DB Provider for ODBC (MSDASQL).

ADO.NET Connections

In ADO.NET you use a data **Connection** object to connect to a specific data source. To connect to an OLE DB data source, use the **OleDbConnection** object of the OLE DB .NET Data Provider.

The following example shows how to do connections to OpenAccess Local OLE DB DB3 database.

```
[Visual Basic]
Imports System
Imports System.Data
Imports System.Data.OleDb
Imports Microsoft.VisualBasic

Public Class connect

  Public Shared Sub Main()
   Dim oaConn As OleDbConnection = New
OleDbConnection("Provider=LocalOpenRDA;Data
Source=test_local;UserId=pooh;Password=bear")

   oaConn.Open()
   Console.WriteLine("Connected to database" & "test_local");
   oaConn.Close()
    End Sub
End Class

[C#]
using System;
using System.Data;
using System.Data.OleDb;

class Connect
{
  public static void Main()
  {
    OleDbConnection oaConn = new
OleDbConnection("Provider=LocalOpenRDA;Data Source=test_local;User
Id=pooh;Password=bear;");

    oaConn.Open();
    Console.WriteLine("Connected to database" + "test_local");
    oaConn.Close();
  }
}
```

Connection String Format

For the OLE DB .NET Data Provider, the connection string format is identical to the connection string format used in ADO, with the following exceptions:

- The **Provider** keyword is required.
- The **URL**, **Remote Provider** and **Remote Server** keywords are not supported.

Closing the Connection

You must always close the **Connection** when you are finished using it. This can be done using either the **Close** or **Dispose** methods of the **Connection** object. Connections are not implicitly released when the **Connection** object falls out of scope or is reclaimed by garbage collection.

ADO.NET Commands

After establishing a connection to a data source, you can execute commands and return results from the data source using a **Command** object. A **Command** object can be created using the **Command** constructor, or by calling the **CreateCommand** method of the **Connection** object.
The **Command** object exposes several **Execute** methods you can use to perform the intended action. When returning results as a stream of data, use **ExecuteReader** to return a **DataReader** object. Use **ExecuteScalar** to return a singleton value. Use **ExecuteNonQuery** to execute commands that do not return rows.

The following example demonstrates how to format a **Command** object to return a list of Employees from the OpenAccess Local OLE DB DB3 database.

```
[Visual Basic]

Imports System
Imports System.Data
Imports System.Data.OleDb
Imports Microsoft.VisualBasic

Public Class Query
    Public Shared Sub Main()
        Dim oaConn As OleDbConnection = New
OleDbConnection("Provider=LocalOpenRDA;Data
Source=test_local;UserId=pooh;Password=bear")

        Dim oaCMD As OleDbCommand = oaConn.CreateCommand()
        oaCMD.CommandText = "SELECT empno, ename FROM emp"

        oaConn.Open()
        Dim myReader As OleDbDataReader = oaCMD.ExecuteReader()
        Dim schemaTable As DataTable = myReader.GetSchemaTable()
        Dim myRow As DataRow

        For Each myRow In schemaTable.Rows
            Console.Write(myRow(schemaTable.Columns("ColumnName").ToString()) &
vbTab)
        Next
        Console.WriteLine()
        Do While myReader.Read()
            Console.WriteLine("{0}" & vbTab & "{1}", myReader.GetInt32(0),
myReader.GetString(1))
        Loop

        myReader.Close()
        oaCMD.Dispose()
    oaConn.Close()
  End Sub
End Class
```

```
[C#]

using System;
using System.Data;
using System.Data.OleDb;

class Select
{
  public static void Main()
  {
    OleDbConnection oaConn = new OleDbConnection("Provider=LocalOpenRDA;Data
Source=test_local;User Id=pooh;Password=bear;");

    OleDbCommand oaCMD = oaConn.CreateCommand();
    oaCMD.CommandText = "SELECT empno, ename FROM emp";

    oaConn.Open();

    OleDbDataReader myReader = oaCMD.ExecuteReader();

    DataTable schemaTable = myReader.GetSchemaTable();

   foreach (DataRow myRow in schemaTable.Rows)
   {
         Console.Write(myRow[schemaTable.Columns["ColumnName"]] + "\t");
   }
    Console.WriteLine();

    while (myReader.Read())
    {
      Console.WriteLine("{0}\t{1}", myReader.GetInt32(0),
myReader.GetString(1));
    }

    myReader.Close();
    oaCMD.Dispose();
    oaConn.Close();
  }
}
```

Obtaining a Single Value

You may need to return information from a database that is not in the form of a table or data stream. For example, you may want to return a single value such as the result of Count(*) or Avg(Salary). The **Command** object provides the capability to return single values using the **ExecuteScalar** method. The **ExecuteScalar** method returns the value of the first column of the first row of the result set as a scalar value.
The following example returns the number of records in a EMP table using the **Count** aggregate function.

```
[Visual Basic]

Imports System
Imports System.Data
Imports System.Data.OleDb
Imports Microsoft.VisualBasic

Public Class OneValue

  Public Shared Sub Main()
        Dim oaConn As OleDbConnection = New
OleDbConnection("Provider=LocalOpenRDA;Data
Source=test_local;UserId=pooh;Password=bear")

        Dim oaCMD As OleDbCommand = oaConn.CreateCommand()
        oaCMD.CommandText = "SELECT count(*) FROM emp"

        oaConn.Open()

        Dim count As Int32 = oaCMD.ExecuteScalar()
           Console.WriteLine(vbNewLine & "Total records={0}", count)

        oaCMD.Dispose()
        oaConn.Close()
    End Sub
End Class
```

```
[C#]

using System;
using System.Data;
using System.Data.OleDb;

class OneValue
{
  public static void Main()
  {
    OleDbConnection oaConn = new
OleDbConnection("Provider=LocalOpenRDA;Data Source=test_local;User
Id=pooh;Password=bear;");

    OleDbCommand oaCMD = oaConn.CreateCommand();
      oaCMD.CommandText = "SELECT count(*) FROM emp";
      oaConn.Open();
    Int32 count = (Int32)oaCMD.ExecuteScalar();
      Console.WriteLine("\nTotal records={0}", count);
  }
}
```

Modifying Data

Using a .NET data provider, you can execute data manipulation language (DML) statements (e.g. INSERT, UPDATE , DELETE). These commands do not return rows as a query would, so the **Command** object provides an **ExecuteNonQuery** method to process them.

```
[Visual Basic]

Imports System
Imports System.Data
Imports System.Data.OleDb
Imports Microsoft.VisualBasic

Public Class Insert

  Public Shared Sub Main()
        Dim oaConn As OleDbConnection = New OleDbConnection("Provider=LocalOpenRDA;Data
Source=test_local;UserId=pooh;Password=bear")
        Dim insertStr As String = "INSERT INTO EMP (EMPNO, ENAME) Values(101,'OA.NET')"

        oaConn.Open()

        Dim oaCMD As OleDbCommand = New OleDbCommand(insertStr, oaConn)
        Dim recordsAffected As Int32 = oaCMD.ExecuteNonQuery()

        Console.WriteLine("Rows affected={0}", recordsAffected)

        oaCMD.Dispose()
        oaConn.Close()

    End Sub
End Class
```

```
[C#]

using System;
using System.Data;
using System.Data.OleDb;

class Insert
{
       public static void Main()
       {
              OleDbConnection oaConn = new OleDbConnection("Provider=LocalOpenRDA;Data
Source=test_local;User Id=pooh;Password=bear;");

              oaConn.Open();

              string insertStr = "INSERT INTO EMP (EMPNO, ENAME) Values(101,'OA.NET')";
              OleDbCommand oaCMD = new OleDbCommand(insertStr,oaConn);
              Int32 recordsAffected = oaCMD.ExecuteNonQuery();

              Console.WriteLine("Rows affected={0}", recordsAffected);

              oaCMD.Dispose();
              oaConn.Close();
       }
}
```

Obtaining Schema Information

You can obtain schema information from your data source using the OLE DB .NET Data Provider. Schema information in a data source includes databases or catalogs available from the data source, tables and views in a database, constraints that exist, and so on.
The OLE DB .NET Data Provider exposes schema information using the **GetOleDbSchemaTable** method of the **OleDbConnection** object. **GetOleDbSchemaTable** takes as arguments an **OleDbSchemaGuid** that identifies which schema information to return and an array of restrictions on those returned columns. **GetOleDbSchemaTable** returns a **DataTable** populated with the schema information.
The following example returns the list of tables in the OpenAccess Local OLE DB DB3 database.

```
[Visual Basic]

Imports System
Imports System.Data
Imports System.Data.OleDb
Imports Microsoft.VisualBasic

Public Class Tables

  Public Shared Sub Main()
    Dim oaConn As OleDbConnection = New OleDbConnection("Provider=LocalOpenRDA;Data
Source=test_local;UserId=pooh;Password=bear")

       oaConn.Open()

        Dim schemaTable As DataTable =
oaConn.GetOleDbSchemaTable(OleDbSchemaGuid.Tables, New Object() {Nothing, Nothing,
Nothing, "TABLE"})

        Dim myRow As DataRow

       Console.WriteLine(schemaTable.Columns("TABLE_NAME").ColumnName & vbNewLine)

        For Each myRow In schemaTable.Rows
            Console.WriteLine(myRow(schemaTable.Columns("TABLE_NAME")) & vbTab)
        Next

        schemaTable.Dispose()
        oaConn.Close()

    End Sub
End Class
```

```
[C#]

using System;
using System.Data;
using System.Data.OleDb;

class tables
{
        public static void Main()
        {
                OleDbConnection oaConn = new
OleDbConnection("Provider=LocalOpenRDA;Data Source=test_local;User
Id=pooh;Password=bear;");

                oaConn.Open();
                DataTable schemaTable =
oaConn.GetOleDbSchemaTable(OleDbSchemaGuid.Tables,
                        new object[] {null, null, null, "TABLE"});

                Console.WriteLine(schemaTable.Columns["TABLE_NAME"].ColumnName +
"\n");

                foreach (DataRow myRow in schemaTable.Rows)
                {
                        Console.WriteLine(myRow[schemaTable.Columns["TABLE_NAME"]] +
"\t");
                }
```

Using Stored Procedures

Stored procedures offer many advantages in data-driven applications. Database operations can be encapsulated in a single command, optimized for best performance, and enhanced with additional security. While a stored procedure can be called by simply passing the stored procedure name followed by parameter arguments as an SQL statement, using the **Parameters** collection of the ADO.NET **Command** object enables you to more explicitly define stored procedure parameters.
To call a stored procedure, set the **CommandType** of the **Command** object to **StoredProcedure**. Once the **CommandType** is set to **StoredProcedure**, you can use the **Parameters** collection to define parameters.

```
[Visual Basic]

Imports System
Imports System.Data
Imports System.Data.OleDb
Imports Microsoft.VisualBasic

Public Class StoreProc

  Public Shared Sub Main()
        Dim oaConn As OleDbConnection = New
OleDbConnection("Provider=LocalOpenRDA;Data
Source=test_local;UserId=pooh;Password=bear")
        Dim procStr As String = "refresh_db"

        oaConn.Open()

        Dim oaCMD As OleDbCommand = New OleDbCommand(procStr, oaConn)
        oaCMD.CommandType = CommandType.StoredProcedure
        Dim recordsAffected As Int32 = oaCMD.ExecuteNonQuery()

        Console.WriteLine("Rows affected={0}", recordsAffected)

        oaCMD.Dispose()
        oaConn.Close()

    End Sub
End Class
```

```
[C#]

using System;
using System.Data;
using System.Data.OleDb;

class StoreProc
{
      public static void Main()
      {
            OleDbConnection oaConn = new
OleDbConnection("Provider=LocalOpenRDA;Data Source=test_local;User
Id=pooh;Password=bear;");

            oaConn.Open();

            string procStr = "refresh_db";
            OleDbCommand oaCMD = new OleDbCommand(procStr,oaConn);
            oaCMD.CommandType=CommandType.StoredProcedure;
            Int32 recordsAffected = oaCMD.ExecuteNonQuery();

            Console.WriteLine("Rows affected={0}", recordsAffected);

            oaCMD.Dispose();
            oaConn.Close();
      }
}
```

Using Parameters with a Command

The OLE DB .NET Data Provider does not support named parameters for passing parameters to an SQL statement or a stored procedure called by a **Command** of **CommandType.Text**. In this case, you must use the question mark (?) placeholder. As a result, the order in which **Parameter** objects are added to the **Parameters** collection must directly correspond to the position of the question mark placeholder for the parameter.

```
[Visual Basic]

Imports System
Imports System.Data
Imports System.Data.OleDb
Imports Microsoft.VisualBasic

Public Class Param
  Public Shared Sub Main()
    Dim oaConn As OleDbConnection = New OleDbConnection("Provider=LocalOpenRDA;Data
Source=test_local;UserId=pooh;Password=bear")

   Dim oaCMD As OleDbCommand = oaConn.CreateCommand()
   oaCMD.CommandText = "SELECT empno, ename FROM emp where ename = ?"
   Dim  myParm As OleDbParameter = oaCMD.Parameters.Add("1", OleDbType.VarChar, 15)
   myParm.Value = "Joe"
    oaConn.Open()

    Dim myReader As OleDbDataReader = oaCMD.ExecuteReader()
    Dim schemaTable As DataTable = myReader.GetSchemaTable()
    Dim myRow As DataRow

    For Each myRow In schemaTable.Rows
          Console.Write(myRow(schemaTable.Columns("ColumnName").ToString()) & vbTab)
    Next
    Console.WriteLine()

    Do While myReader.Read()
          Console.WriteLine("{0}" & vbTab & "{1}", myReader.GetInt32(0),
myReader.GetString(1))
    Loop

    myReader.Close()
    oaCMD.Dispose()
    oaConn.Close()

    End Sub
End Class
```

```
[C#]

using System;
using System.Data;
using System.Data.OleDb;

class Param
{
  public static void Main()
  {
    OleDbConnection oaConn = new OleDbConnection("Provider=LocalOpenRDA;Data
Source=test_local;User Id=pooh;Password=bear;");

    OleDbCommand oaCMD = oaConn.CreateCommand();
    oaCMD.CommandText = "SELECT empno, ename FROM emp where ename = ?";

    OleDbParameter myParm = oaCMD.Parameters.Add("1", OleDbType.VarChar, 15);
    myParm.Value = "Joe";

    oaConn.Open();

    OleDbDataReader myReader = oaCMD.ExecuteReader();

    DataTable schemaTable = myReader.GetSchemaTable();

   foreach (DataRow myRow in schemaTable.Rows)
   {
         Console.Write(myRow[schemaTable.Columns["ColumnName"]] + "\t");
   }
    Console.WriteLine();

    while (myReader.Read())
    {
      Console.WriteLine("{0}\t{1}", myReader.GetInt32(0), myReader.GetString(1));
    }

    myReader.Close();
    oaCMD.Dispose();
    oaConn.Close();
 }
```

Using a DataSet With DataAdapter

Following are the general steps to use a **DataSet** with a **DataAdapter**.

- Build and fill each table in a **DataSet** with data from a database, using the OLE DB .NET data provider.
- Change the data in individual **DataTable** objects by adding, updating, or deleting **DataRow** objects.
- Call the **Update** method of the **DataAdapter**, passing the updated **DataSet** as an argument.

```
[Visual Basic]

Imports System
Imports System.Data
Imports System.Data.OleDb
Imports Microsoft.VisualBasic

Public Class DataAdapterQuery
    Public Shared Sub Main()
        Dim oaConn As OleDbConnection = New
OleDbConnection("Provider=LocalOpenRDA;Data
Source=test_local;UserId=pooh;Password=bear")
        oaConn.Open()
        Dim empDS As DataSet = New DataSet()
        Dim empDA As OleDbDataAdapter = New OleDbDataAdapter("SELECT empno,ename
FROM emp", oaConn)
        empDA.Fill(empDS, "EmpTable")
        Dim empTable As DataTable = empDS.Tables("EmpTable")
        Dim myCol As DataColumn

        For Each myCol In empTable.Columns
            Console.Write("{0}" & vbTab, myCol.ColumnName)
        Next
        Console.WriteLine()
        Dim myRow As DataRow
        For Each myRow In empTable.Rows
            For Each myCol In empTable.Columns
                Console.Write("{0}" & vbTab, myRow(myCol))
            Next
            Console.WriteLine()
        Next
        empTable.Dispose()
        empDS.Dispose()
        empDA.Dispose()
        oaConn.Close()
    End Sub
End Class
```

```
[C#]

using System;
using System.Data;
using System.Data.OleDb;

class DataAdapterSelect
{
  public static void Main()
  {
    OleDbConnection oaConn = new OleDbConnection("Provider=LocalOpenRDA;Data
Source=test_local;User Id=pooh;Password=bear;");

    oaConn.Open();
    DataSet empDS = new DataSet();
    OleDbDataAdapter empDA = new OleDbDataAdapter("SELECT empno,ename FROM
emp", oaConn);
    empDA.Fill(empDS, "EmpTable");

    DataTable empTable = empDS.Tables["EmpTable"];
   foreach (DataColumn myCol in empTable.Columns)
     Console.Write("{0}\t", myCol.ColumnName);
  Console.WriteLine();

  foreach (DataRow myRow in empTable.Rows)
  {
    foreach (DataColumn myCol in empTable.Columns)
      Console.Write("{0}\t", myRow[myCol]);
    Console.WriteLine();
  }
   empTable.Dispose();
   empDS.Dispose();
   empDA.Dispose();
oaConn.Close();
  }
}
```