PROGRESS® DataDirect®

WHITE PAPER

www.progress.com

OPENACCESS™ .NET SOFTWARE DEVELOPMENT KIT

PROGRESS

TABLE OF CONTENTS

OVERVIEW

The objective of the Progress® DataDirect® 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 Progress Software 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 technology.

OpenAccess is designed to hide the developer from the complexities of SQL, ODBC, OLE DB, JDBC, .NET, and 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, JDBC, or .NET 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. We plug into ADO.NET through native .NET Provider. This means that you can use our .NET SDK to create a fully functional .NET Provider for your data source. Within days you can have a .NET Provider to your data source.

ADO.NET works with provider developed using OpenAccess .NET provider SDK. In this document we have given examples of ADO.NET tested with OpenAccess .NET provider 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 native .NET Data Provider requires the installation of Runtime .NET Framework.

OPENACCESS .NET SDK ARCHITECTURE

Figure 1 shows interaction between a .NET application and the Progress® DataDirect® OpenAccess™ .NET Provider. In client/server configuration, the OpenAccess .NET 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 .NET Provider library is linked with the DAM and your IP to build a ADO.NET compliant .NET provider specific for your data source.

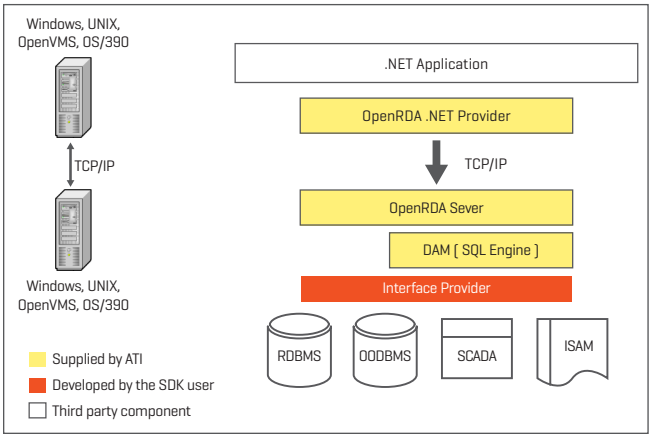

Figure 1: *Client/Server Architecture*

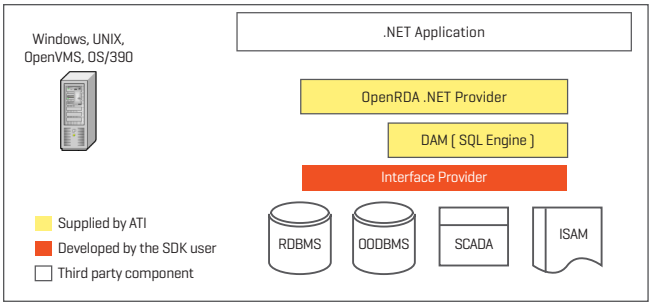

Figure 2: *Local Architecture*

.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.

ADO.NET CONNECTIONS

In ADO.NET you use a data Connection object to connect to a specific data source. To connect to a native data source, use the OaConnection object of the OpenAccess .NET Data Provider.

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

```
[Visual Basic]
Imports System
Imports System.Data
Imports OpenRDA.Data.OaClient
Imports Microsoft.VisualBasic

Public Class connect

  Public Shared Sub Main()
    Dim oaConn As OaConnection = New OaConnection("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 OpenRDA.Data.OaClient;

class Connect
{
  public static void Main()
  {
    OaConnection oaConn = new OaConnection("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 OpenAccess native .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 not 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 .NET DB3 database.

```
[Visual Basic]

Imports System
Imports System.Data
Imports OpenRDA.Data.OaClient
Imports Microsoft.VisualBasic

Public Class Query
    Public Shared Sub Main()
      Dim oaConn As OaConnection = New OaConnection("Data
Source=test _ local;UserId=pooh;Password=bear")

      Dim oaCMD As OaCommand = oaConn.CreateCommand()
      oaCMD.CommandText = "SELECT empno, ename FROM emp"
      oaConn.Open()
      Dim myReader As OaDataReader = 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 OpenRDA.Data.OaClient;

class Select
{
  public static void Main()
  {
    OaConnection oaConn = new OaConnection("Data Source=test _ local;User
Id=pooh;Password=bear;");

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

    oaConn.Open();

    OaDataReader 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 OpenRDA.Data.OaClient
Imports Microsoft.VisualBasic

Public Class OneValue

  Public Shared Sub Main()
    Dim oaConn As OaConnection = New OaConnection("Data
Source=test _ local;UserId=pooh;Password=bear")

    Dim oaCMD As OaCommand = 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 OpenRDA.Data.OaClient;

class OneValue
{
  public static void Main()
  {
    OaConnection oaConn = new OaConnection("Data
Source=test _ local;User Id=pooh;Password=bear;");

    OaCommand 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 OpenRDA.Data.OaClient
Imports Microsoft.VisualBasic

Public Class Insert

  Public Shared Sub Main()
     Dim oaConn As OaConnection = New OaConnection("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 OaCommand = New OaCommand(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 OpenRDA.Data.OaClient;

class Insert
{
  public static void Main()
  {
    OaConnection oaConn = new OaConnection("Data Source=test _ local;User
Id=pooh;Password=bear;");

    oaConn.Open();

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

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

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

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 OpenRDA.Data.OaClient
Imports Microsoft.VisualBasic

Public Class StoreProc

  Public Shared Sub Main()
    Dim oaConn As OaConnection = New OaConnection("Data
Source=test _ local;UserId=pooh;Password=bear")
    Dim procStr As String = "refresh _ db"

    oaConn.Open()

    Dim oaCMD As OaCommand = New OaCommand(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 OpenRDA.Data.OaClient;

class StoreProc
{
  public static void Main()
  {
    OaConnection oaConn = new OaConnection("Data Source=test _ local;User
Id=pooh;Password=bear;");

    oaConn.Open();

    string procStr = "refresh _ db";
    OaCommand oaCMD = new OaCommand(procStr,oaConn);
    oaCMD.CommandType=CommandType.StoredProcedure;
    Int32 recordsAffected = oaCMD.ExecuteNonQuery();

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

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