C# 8.x Next

DataReader In C

C# Corner Become a member Login

Contribute Ask Question

John Hudai Godel Feb 17 2004 281.6k 0 4

ADO.NET DataReader is used to store data returned from a database in a fast, forward-only, in-memory records. In this article, learn how to use a DataReader in a C# application.

ADO.NET DataReader

ADO.NET DataReader object is used for accessing data from the data store and is one of the two mechanisms that ADO.NET provides. As we will remember DataReader object provides a read only, forward only, high performance mechanism to retrieve data from a data store as a data stream, while staying connected with the data source. The DataReader is restricted but highly optimized. The .NET framework provides data providers for SQL Server native OLE DB providers and native ODBC drivers,

- SqlDataReader
- OleDbDataReader
- OdbcDataReader

You can use the ADO.NET DataReader to retrieve a read-only, forward-only stream of data from a database. Using the DataReader can increase application performance and reduce system overhead because only one row at a time is ever in memory. After creating an instance of the Command object, you create a DataReader by calling Command.ExecuteReader to retrieve rows from a data source, as shown in the following example.

```
SqlDataReader myReader = myCommand.ExecuteReader();
```

You use the Read method of the DataReader object to obtain a row from the results of the query. You can access each column of the returned row by passing the name or ordinal reference of the column to the DataReader. However, for best performance, the DataReader provides a series of methods that allow you to access column values in their native data types (GetDateTime, GetDouble, GetGuid, GetInt32, and so on). For a list of typed accessor methods, see the OleDbDataReader Class and the SqlDataReader Class. Using the typed accessor methods whe

The following code example iterates through a DataReader objec each row.

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

The DataReader provides an unbuffered stream of data that allows procedural logic to efficiently process results from a data source sequentially. The DataReader is a good choice when retrieving large amounts of data because the data is not cached in memory. You should always call the Close method when you have finished using the DataReader object. If your Command contains output parameters or return values, they will not be available until the DataReader is closed.

Note that while a DataReader is open, the Connection is in use exclusively by that DataReader. You will not be able to execute any commands for the Connection, including creating another DataReader, until the original DataReader is closed.

Multiple Result Sets

If multiple result sets are returned, the DataReader provides the NextResult method to iterate through the result sets in order, as shown in the following code example.

```
SqlCommand myCMD = new SqlCommand("SELECT CategoryID, CategoryName FROM Cate
nwindConn.Open();
SqlDataReader myReader = myCMD.ExecuteReader();
do {
    Console.WriteLine("\t{0}\t{1}", myReader.GetName(0), myReader.GetName(1)
    while (myReader.Read()) Console.WriteLine("\t{0}\t{1}", myReader.GetInt
}
while (myReader.NextResult());
myReader.Close();
nwindConn.Close();
```

The DataReader implementation must provide two basic capabilities: forward-only access over one or more of the resultsets obtained by executing a Command, and access to the column values within each row. Data types from your data source will be stored in your .NET-based application as .NET Framework types. Your DataReader implementation will also provide strongly typed accessor methods for your DataReader that return column values as .NET Framework types. Examples of a strongly typed accessor would be GetInt32, GetString, and so on.

If your .NET data provider has proprietary types that cannot adequately be exposed as .NET Framework types, you may extend the interfaces to support proprietary types, then add typed accessors for your DataReader that return proprietary types as well. For example, you can add GetMyStructure, GetMyTimeStamp, and so on. An example of this is the SQL Server .NET Data Provider, which exposes proprietary types using the System.Data.SqlTypes Namespace. The SqlDataReader then exposes those types as SqlTypes using strongly typed accessor methods. For example: GetSqlBinary, GetSqlDateTime, GetSqlDecimal, and so on.

```
using System.Globalization;
namespace DotNetDataProviderTemplate {
    public class TemplateDataReader: IDataReader {
        // The DataReader must always be open when returned to the user.
        private bool dReaderOpen = true;
        // Keep track of the results and position
        // within the resultset (starts prior to first record).
        private TestDataBase.TestDataBaseResultSet testResultset;
        private static int testSTARTPOS = -1;
        private int testNPos = testSTARTPOS;
        private TemplateConnection testconnection = null;
        internal TemplateDataReader(TestDataBase.TestDataBaseResultSet resul
            testResultset = resultset;
        }
        internal TemplateDataReader(TestDataBase.TestDataBaseResultSet resul
            testResultset = resultset;
            testconnection = connection;
        }
        public int Depth {
            get {
                return 0;
```

```
        public bool IsClosed {
            get {
                return !dReaderOpen;
            }
        }
        public int RecordsAffected {
            get {
                return -1;
            }
        }
        public void Close() {
            dReaderOpen = false;
        }
        publicbool NextResult() {
            returnfalse;
        }
        public bool Read() {
            if (++testNPos >= testResultset.data.Length / testResultset.meta
            else returntrue;
        }
        public DataTable GetSchemaTable() {
            thrownew NotSupportedException();
        }
        public int FieldCount {
            get {
                return testResultset.metaData.Length;
            }
        }
        public String GetName(int i) {
            return testResultset.metaData[i].name;
        }
        public String GetDataTypeName(int i) {
            return testResultset.metaData[i].type.Name;
        }
        public Type GetFieldType(int i) {
            return testResultset.metaData[i].type;
            return testResultset.data[testNPos, i
        }
        public int GetValues(object[] values) {
            for (int i = 0; i < values.Length && i < testResultset.metaData.
                values[i] = testResultset.data[testNPos, i];
            }
            return i;
        }
        public int GetOrdinal(string name) {
            for (int i = 0; i < testResultset.metaData.Length; i++) {
                if (0 == _cultureAwareCompare(name, testResultset.metaData[i
                    return i;
                }
            }
            thrownew IndexOutOfRangeException("Could not find specified colu
        }
        public object this[int i] {
            get {
                return testResultset.data[testNPos, i];
            }
        }
```

```
                returnthis[GetOrdinal(name)];
            }
        }
        public bool GetBoolean(int i) {
            return (bool) testResultset.data[testNPos, i];
        }
        public byte GetByte(int i) {
            return (byte) testResultset.data[testNPos, i];
        }
        public long GetBytes(int i, long fieldOffset, byte[] buffer, int bu
            thrownew NotSupportedException("GetBytes not supported.");
        }
        public char GetChar(int i) {
            return (char) testResultset.data[testNPos, i];
        }
        public long GetChars(int i, long fieldoffset, char[] buffer, int bu
            thrownew NotSupportedException("GetChars not supported.");
        }
        public Guid GetGuid(int i) {
            return (Guid) testResultset.data[testNPos, i];
        }
        public Int16 GetInt16(int i) {
            return (Int16) testResultset.data[testNPos, i];
        }
        public Int32 GetInt32(int i) {
            return (Int32) testResultset.data[testNPos, i];
        }
        public Int64 GetInt64(int i) {
            return (Int64) testResultset.data[testNPos, i];
        }
        public float GetFloat(int i) {
            return (float) testResultset.data[testNPos, i];
        }
        public double GetDouble(int i) {
            return (double) testResultset.data[testNPos, i];
        }
        }
        public Decimal GetDecimal(int i) {
            return (Decimal) testResultset.data[·
        }
        public DateTime GetDateTime(int i) {
            return (DateTime) testResultset.data[testNPos, i];
        }
        public IDataReader GetData(int i) {
            thrownew NotSupportedException("GetData not supported.");
        }
        public bool IsDBNull(int i) {
            return testResultset.data[testNPos, i] == DBNull.Value;
        }
        private int _cultureAwareCompare(string strA, string strB) {
            return
            CultureInfo.CurrentCulture.CompareInfo.Compare(strA, strB, ompar
        }
    }
}
```

refers to Japanese hiragana and katakana characters, which represent phonetic sounds in the Japanese language.

Summary

In this article we had a discussion about the DataSet and its role in data-oriented applications. The DataSet is main one of the main components and it is important to understand to DataAdapter, DataTable, DataView, DataGrid and other objects in ADO.NET. Finally we create an example, which it has several functionality about DataSet and its relations with other ADO.NET classes. Next article we will discuss about multiple data tables and it will give us more idea on complex, advanced DataSets.

Next Recommended Article

The "ins" and "outs" of Using Stored Procedures in C#

This article reviews the creation of stored procedures with parameters using the Visual Studio IDE.

ADO.NET DataReader   Data Reader   C# DataReader

John Hudai Godel TOP 500

http://www.alpinegate.com

341   2.3m   1

4   0

FEATURED ARTICLES

Azure Security Foundation: Safety First In An Uncertain World

Create A Blazor Server SPA With Dapper

Create a Single Page App with Blazor Server and Entity Framework Core 3.0

Setup Azure CI/CD Pipelines Using Visual Studio

Create A Simple Blazor Server Application With .NET Core 3.0

View All

C# 8.x Next

TRENDING UP

01 Learn Angular 8 Step By Step In 10 Days - Angular Service (Day 8)

02 Overview Of Polymorphism In C#

03 Detailed Insight On The SharePoint 5000 List View Threshold Limit Issue

04 Linear Regression

05 C# 8.x Next

06 Sending an Email to a Distribution List in Microsoft Flow

07 Isolated Web Parts In SPFx

08 Implementing .NET Core Health Checks

09 SharePoint Empowers MS Teams

10 Top 10 Cloud Service Providers In 2020

View All