**ADO.NET Overview**

ADO.NET provides consistent access to data sources such as SQL Server and XML, and to data sources exposed through OLE DB and ODBC. Data-sharing consumer applications can use ADO.NET to connect to these data sources and retrieve, handle, and update the data that they contain.

ADO.NET separates data access from data manipulation into discrete components that can be used separately or in tandem. ADO.NET includes .NET Framework data providers for connecting to a database, executing commands, and retrieving results. Those results are either processed directly, placed in an ADO.NET [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) object in order to be exposed to the user in an ad hoc manner, combined with data from multiple sources, or passed between tiers. The DataSet object can also be used independently of a .NET Framework data provider to manage data local to the application or sourced from XML.

The ADO.NET classes are found in System.Data.dll, and are integrated with the XML classes found in System.Xml.dll. For sample code that connects to a database, retrieves data from it, and then displays that data in a console window, see [ADO.NET Code Examples](http://msdn.microsoft.com/en-us/library/dw70f090.aspx).

ADO.NET provides functionality to developers who write managed code similar to the functionality provided to native component object model (COM) developers by ActiveX Data Objects (ADO). We recommend that you use ADO.NET, not ADO, for accessing data in your .NET applications.

**Privacy Statement**: The System.Data.dll, System.Data.Design.dll, System.Data.OracleClient.dll, System.Data.SqlXml.dll, System.Data.Linq.dll, System.Data.SqlServerCe.dll, and System.Data.DataSetExtensions.dll assemblies do not distinguish between a user's private data and non-private data. These assemblies do not collect, store, or transport any user's private data. However, third-party applications might collect, store, or transport a user's private data using these assemblies.

**ADO.NET Architecture**

Data processing has traditionally relied primarily on a connection-based, two-tier model. As data processing increasingly uses multi-tier architectures, programmers are switching to a disconnected approach to provide better scalability for their applications.

 ADO.NET Components

The two main components of ADO.NET 3.0 for accessing and manipulating data are the .NET Framework data providers and the [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx).

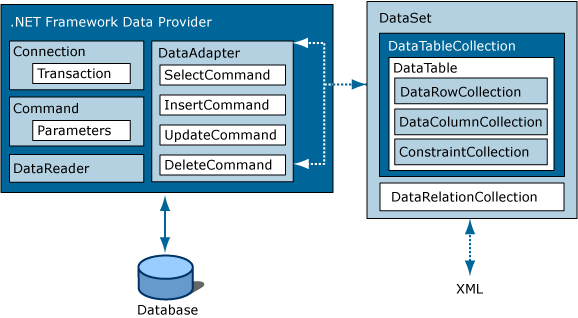
### .NET Framework Data Providers

The .NET Framework Data Providers are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data. The Connection object provides connectivity to a data source. The Command object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information. The DataReader provides a high-performance stream of data from the data source. Finally, the DataAdapter provides the bridge between the DataSet object and the data source. The DataAdapter uses Command objects to execute SQL commands at the data source to both load the DataSet with data and reconcile changes that were made to the data in the DataSet back to the data source. For more information, see [.NET Framework Data Providers (ADO.NET)](http://msdn.microsoft.com/en-us/library/a6cd7c08.aspx) and [Retrieving and Modifying Data in ADO.NET](http://msdn.microsoft.com/en-us/library/ms254937.aspx).

### The DataSet

The ADO.NET DataSet is explicitly designed for data access independent of any data source. As a result, it can be used with multiple and differing data sources, used with XML data, or used to manage data local to the application. The DataSet contains a collection of one or more [DataTable](http://msdn.microsoft.com/en-us/library/system.data.datatable.aspx) objects consisting of rows and columns of data, and also primary key, foreign key, constraint, and relation information about the data in the DataTable objects. For more information, see [DataSets, DataTables, and DataViews (ADO.NET)](http://msdn.microsoft.com/en-us/library/ss7fbaez.aspx).

The following diagram illustrates the relationship between a .NET Framework data provider and a DataSet.

ADO.NET architecture  


### Choosing a DataReader or a DataSet

When you decide whether your application should use a DataReader (see [Retrieving Data Using a DataReader (ADO.NET)](http://msdn.microsoft.com/en-us/library/haa3afyz.aspx)) or a DataSet (see [DataSets, DataTables, and DataViews (ADO.NET)](http://msdn.microsoft.com/en-us/library/ss7fbaez.aspx)), consider the type of functionality that your application requires. Use a DataSet to do the following:

* Cache data locally in your application so that you can manipulate it. If you only need to read the results of a query, the DataReader is the better choice.
* Remote data between tiers or from an XML Web service.
* Interact with data dynamically such as binding to a Windows Forms control or combining and relating data from multiple sources.
* Perform extensive processing on data without requiring an open connection to the data source, which frees the connection to be used by other clients.

If you do not require the functionality provided by the DataSet, you can improve the performance of your application by using the DataReader to return your data in a forward-only, read-only manner. Although the DataAdapter uses the DataReader to fill the contents of a DataSet (see [Populating a DataSet from a DataAdapter (ADO.NET)](http://msdn.microsoft.com/en-us/library/bh8kx08z.aspx)), by using the DataReader, you can boost performance because you will save memory that would be consumed by the DataSet, and avoid the processing that is required to create and fill the contents of the DataSet.

 LINQ to DataSet

LINQ to DataSet provides query capabilities and compile-time type checking over data cached in a DataSet object. It allows you to write queries in one of the .NET Framework development language, such as C# or Visual Basic. For more information, see [LINQ to DataSet](http://msdn.microsoft.com/en-us/library/bb386977.aspx).

 LINQ to SQL

LINQ to SQL supports queries against an object model that is mapped to the data structures of a relational database without using an intermediate conceptual model. Each table is represented by a separate class, tightly coupling the object model to the relational database schema. LINQ to SQL translates language-integrated queries in the object model into Transact-SQL and sends them to the database for execution. When the database returns the results, LINQ to SQL translates the results back into objects. For more information, see [LINQ to SQL](http://msdn.microsoft.com/en-us/library/bb386976.aspx).

 ADO.NET Entity Framework

The ADO.NET Entity Framework is designed to enable developers to create data access applications by programming against a conceptual application model instead of programming directly against a relational storage schema. The goal is to decrease the amount of code and maintenance required for data-oriented applications. For more information, see [ADO.NET Entity Framework](http://msdn.microsoft.com/en-us/library/bb399572.aspx).

 ADO.NET Data Services

The ADO.NET Data Services framework is used to deploy data services on the Web or an intranet. The data is structured as entities and relationships according to the specifications of the Entity Data Model. Data deployed on this model is addressable by standard HTTP protocol. For more information, see [ADO.NET Data Services Framework](http://msdn.microsoft.com/en-us/library/cc668792.aspx).

 XML and ADO.NET

ADO.NET leverages the power of XML to provide disconnected access to data. ADO.NET was designed hand-in-hand with the XML classes in the .NET Framework; both are components of a single architecture.

ADO.NET and the XML classes in the .NET Framework converge in the DataSet object. The DataSet can be populated with data from an XML source, whether it is a file or an XML stream. The DataSet can be written as World-Wide Web Consortium (W3C) compliant XML that includes its schema as XML schema definition language (XSD) schema, regardless of the source of the data in the DataSet. Because of the native serialization format of the DataSet is XML, it is an excellent medium for moving data between tiers, making the DataSet an optimal choice for remoting data and schema context to and from an XML Web service. For more information, see [XML Documents and Data](http://msdn.microsoft.com/en-us/library/2bcctyt8.aspx).

**ADO.NET Data Platform**

The ADO.NET Data Platform is a multi-release strategy to decrease the amount of coding and maintenance required for developers by enabling them to program against conceptual entity data models. This platform includes the ADO.NET Entity Framework and related technologies.

 Entity Framework

The ADO.NET Entity Framework is designed to enable developers to create data access applications by programming against a conceptual application model instead of programming directly against a relational storage schema. The goal is to decrease the amount of code and maintenance required for data-oriented applications. For more information, see [ADO.NET Entity Framework](http://msdn.microsoft.com/en-us/library/bb399572.aspx).

### Entity Data Model (EDM)

An Entity Data Model (EDM) is a design specification that defines application data as sets of entities and relationships. Data in this model supports object-relational mapping and data programmability across application boundaries.

EDM data types and relationships are defined in an conceptual model. This is an XML schema written in conceptual schema definition language (CSDL). The conceptual model is used to build programmable classes that represent application data. Developers can extend these objects as required to support various application needs. For more information, see [Entity Data Model](http://msdn.microsoft.com/en-us/library/bb387122.aspx).

### Object Services

Object Services allow programmers to interact with the conceptual model through a set of common language runtime (CLR) classes. These classes can be automatically generated from the conceptual model or can be developed independently to reflect the structure of the conceptual model. Object Services also provides infrastructure support for the Entity Framework, including services such as state management, change tracking, identity resolution, loading and navigating relationships, propagating object changes to database modifications, and query building support for Entity SQL. For more information, see [Object Services Overview (Entity Framework)](http://msdn.microsoft.com/en-us/library/bb386871.aspx).

### LINQ to Entities

LINQ to Entities is a language-integrated query (LINQ) implementation that allows developers to create strongly-typed queries against the Entity Framework object context by using LINQ expressions and LINQ standard query operators. LINQ to Entities allows developers to work against a conceptual model with a very flexible object-relational mapping across Microsoft SQL Server and third-party databases. For more information, see [LINQ to Entities](http://msdn.microsoft.com/en-us/library/bb386964.aspx).

### Entity SQL

Entity SQL is a text-based query language designed to interact with an Entity Data Model. Entity SQL is an SQL dialect that contains constructs for querying in terms of higher-level modeling concepts, such as inheritance, complex types, and explicit relationships. Developers can also use Entity SQL directly with Object Services. For more information, see [Entity SQL Language Reference](http://msdn.microsoft.com/en-us/library/bb399560.aspx).

### EntityClient

EntityClient is a new .NET Framework data provider used for interacting with an Entity Data Model. EntityClient follows the .NET Framework data provider pattern of exposing [EntityConnection](http://msdn.microsoft.com/en-us/library/system.data.entityclient.entityconnection.aspx) and [EntityCommand](http://msdn.microsoft.com/en-us/library/system.data.entityclient.entitycommand.aspx) objects that return an [EntityDataReader](http://msdn.microsoft.com/en-us/library/system.data.entityclient.entitydatareader.aspx). EntityClient works with the Entity SQL language, providing flexible mapping to storage-specific data providers. For more information, see [EntityClient and Entity SQL](http://msdn.microsoft.com/en-us/library/bb386881.aspx).

### ADO.NET Data Services

The ADO.NET Data Services framework is used to deploy data services on the Web or on an intranet. The data is structured as entities and relationships according to the specifications of the Entity Data Model. Data deployed on this model is addressable by standard HTTP protocol. For more information, see [ADO.NET Data Services Framework](http://msdn.microsoft.com/en-us/library/cc668792.aspx).

### Entity Data Model Tools

The Entity Framework provides command-line tools, wizards, and designers to facilitate building EDM applications. The EntityDataSource control supports data binding scenarios based on the EDM. The programming surface of the EntityDataSource control is similar to other data source controls in Visual Studio. For more information, see [Entity Data Model Tools](http://msdn.microsoft.com/en-us/library/bb399249.aspx).

**LINQ and ADO.NET**

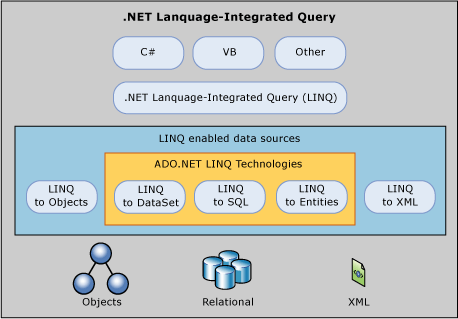
Today, many business developers must use two (or more) programming languages: a high-level language for the business logic and presentation layers (such as Visual C# or Visual Basic), and a query language to interact with the database (such as Transact-SQL). This requires the developer to be proficient in several languages to be effective, and also causes language mismatches in the development environment. For example, an application that uses a data access API to execute a query against a database specifies the query as a string literal by using quotation marks. This query string is un-readable to the compiler and is not checked for errors, such as invalid syntax or whether the columns or rows it references actually exist. There is no type checking of the query parameters and no IntelliSense support, either.

Language-Integrated Query (LINQ) enables developers to form set-based queries in their application code, without having to use a separate query language. You can write LINQ queries against various enumerable data sources (that is, a data source that implements the [IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx) interface), such as in-memory data structures, XML documents, SQL databases, and [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) objects. Although these enumerable data sources are implemented in various ways, they all expose the same syntax and language constructs. Because queries can be formed in the programming language itself, you do not have to use another query language that is embedded as string literals that cannot be understood or verified by the compiler. Integrating queries into the programming language also enables Visual Studio programmers to be more productive by providing compile-time type and syntax checking, and IntelliSense. These features reduce the need for query debugging and error fixing. For more information, see [Introduction to LINQ](http://msdn.microsoft.com/en-us/library/bb397897.aspx). Also see the [LINQ General Programming Guide](http://msdn.microsoft.com/en-us/library/bb397912.aspx), which contains detailed information about how to use LINQ technologies.

There are three separate ADO.NET Language-Integrated Query (LINQ) technologies: LINQ to DataSet ,LINQ to SQL, and LINQ to Entities. LINQ to DataSet provides richer, optimized querying over the [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) and LINQ to SQL enables you to directly query SQL Server database schemas, and LINQ to Entities allows you to query an Entity Data Model.

Transferring data from SQL tables into objects in memory is often tedious and error-prone. The LINQ provider implemented by LINQ to DataSet and LINQ to SQL converts the source data into [IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx)-based object collections. The programmer always views the data as an [IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx) collection, both when you query and when you update. Full IntelliSense support is provided for writing queries against those collections.

The following diagram provides an overview of how the ADO.NET LINQ technologies relate to high-level programming languages and LINQ-enabled data sources.



The following sections provide more information about LINQ to DataSet, LINQ to SQL, and LINQ to Entities.

 LINQ to DataSet

The [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) is a key element of the disconnected programming model that ADO.NET is built on, and is widely used. LINQ to DataSet enables developers to build richer query capabilities into [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) by using the same query formulation mechanism that is available for many other data sources. For more information, see [LINQ to DataSet](http://msdn.microsoft.com/en-us/library/bb386977.aspx).

 LINQ to SQL

LINQ to SQL is a useful tool for developers who do not require mapping to a conceptual model. By using LINQ to SQL, you can use the LINQ programming model directly over existing database schema. LINQ to SQL enables developers to generate .NET Framework classes that represent data. Rather than mapping to a conceptual data model, these generated classes map directly to database tables, views, stored procedures, and user-defined functions.

With LINQ to SQL, developers can write code directly against the storage schema using the same LINQ programming pattern as in-memory collections and the [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx), in addition to other data sources such as XML. For more information, see [LINQ to SQL](http://msdn.microsoft.com/en-us/library/bb386976.aspx).

 LINQ to Entities

Most applications are currently written on top of relational databases. At some point, these applications will need to interact with the data represented in a relational form. Database schemas are not always ideal for building applications, and the conceptual models of application are not the same as the logical models of databases. The Entity Data Model is a conceptual data model that can be used to model the data of a particular domain so that applications can interact with data as objects. See [ADO.NET Entity Framework](http://msdn.microsoft.com/en-us/library/bb399572.aspx) for more information.

Through the Entity Data Model, relational data is exposed as objects in the .NET environment. This makes the object layer an ideal target for LINQ support, allowing developers to formulate queries against the database from the language used to build the business logic. This capability is known as LINQ to Entities. See [LINQ to Entities](http://msdn.microsoft.com/en-us/library/bb386964.aspx) for more information.

**.NET Framework Data Providers (ADO.NET)**

A .NET Framework data provider is used for connecting to a database, executing commands, and retrieving results. Those results are either processed directly, placed in a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) in order to be exposed to the user as needed, combined with data from multiple sources, or remoted between tiers. .NET Framework data providers are lightweight, creating a minimal layer between the data source and code, increasing performance without sacrificing functionality.

The following table lists the data providers that are included in the .NET Framework.

|  |  |
| --- | --- |
| **.NET Framework data provider** | **Description** |
| .NET Framework Data Provider for SQL Server | Provides data access for Microsoft SQL Server version 7.0 or later. Uses the [System.Data.SqlClient](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.aspx) namespace. |
| .NET Framework Data Provider for OLE DB | For data sources exposed by using OLE DB. Uses the [System.Data.OleDb](http://msdn.microsoft.com/en-us/library/system.data.oledb.aspx) namespace. |
| .NET Framework Data Provider for ODBC | For data sources exposed by using ODBC. Uses the [System.Data.Odbc](http://msdn.microsoft.com/en-us/library/system.data.odbc.aspx) namespace. |
| .NET Framework Data Provider for Oracle | For Oracle data sources. The .NET Framework Data Provider for Oracle supports Oracle client software version 8.1.7 and later, and uses the [System.Data.OracleClient](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.aspx) namespace. |
| EntityClient Provider | Provides data access for Entity Data Model (EDM) applications. Uses the [System.Data.EntityClient](http://msdn.microsoft.com/en-us/library/system.data.entityclient.aspx) namespace. |

 Core Objects of .NET Framework Data Providers

The following table outlines the four core objects that make up a .NET Framework data provider.

|  |  |
| --- | --- |
| **Object** | **Description** |
| Connection | Establishes a connection to a specific data source. The base class for all Connection objects is the [DbConnection](http://msdn.microsoft.com/en-us/library/system.data.common.dbconnection.aspx) class. |
| Command | Executes a command against a data source. Exposes Parameters and can execute in the scope of a Transaction from a Connection. The base class for all Command objects is the [DbCommand](http://msdn.microsoft.com/en-us/library/system.data.common.dbcommand.aspx) class. |
| DataReader | Reads a forward-only, read-only stream of data from a data source. The base class for all DataReader objects is the [DbDataReader](http://msdn.microsoft.com/en-us/library/system.data.common.dbdatareader.aspx) class. |
| DataAdapter | Populates a DataSet and resolves updates with the data source. The base class for all DataAdapter objects is the [DbDataAdapter](http://msdn.microsoft.com/en-us/library/system.data.common.dbdataadapter.aspx) class. |

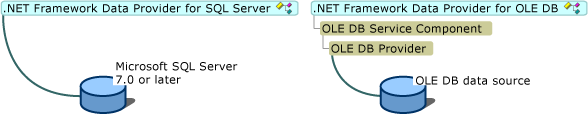
In addition to the core classes listed in the table earlier in this document, a .NET Framework data provider also contains the classes listed in the following table.

|  |  |
| --- | --- |
| **Object** | **Description** |
| Transaction | Enlists commands in transactions at the data source. The base class for all Transaction objects is the [DbTransaction](http://msdn.microsoft.com/en-us/library/system.data.common.dbtransaction.aspx) class. ADO.NET also provides support for transactions using classes in the [System.Transactions](http://msdn.microsoft.com/en-us/library/system.transactions.aspx) namespace. |
| CommandBuilder | A helper object that automatically generates command properties of a DataAdapter or derives parameter information from a stored procedure and populates the Parameters collection of a Command object. The base class for all CommandBuilder objects is the [DbCommandBuilder](http://msdn.microsoft.com/en-us/library/system.data.common.dbcommandbuilder.aspx) class. |
| ConnectionStringBuilder | A helper object that provides a simple way to create and manage the contents of connection strings used by the Connection objects. The base class for all ConnectionStringBuilder objects is the [DbConnectionStringBuilder](http://msdn.microsoft.com/en-us/library/system.data.common.dbconnectionstringbuilder.aspx) class. |
| Parameter | Defines input, output, and return value parameters for commands and stored procedures. The base class for all Parameter objects is the [DbParameter](http://msdn.microsoft.com/en-us/library/system.data.common.dbparameter.aspx) class. |
| Exception | Returned when an error is encountered at the data source. For an error encountered at the client, .NET Framework data providers throw a .NET Framework exception. The base class for all Exception objects is the [DbException](http://msdn.microsoft.com/en-us/library/system.data.common.dbexception.aspx) class. |
| Error | Exposes the information from a warning or error returned by a data source. |
| ClientPermission | Provided for .NET Framework data provider code access security attributes. The base class for all ClientPermission objects is the [DBDataPermission](http://msdn.microsoft.com/en-us/library/system.data.common.dbdatapermission.aspx) class. |

 .NET Framework Data Provider for SQL Server (SqlClient)

The .NET Framework Data Provider for SQL Server (SqlClient) uses its own protocol to communicate with SQL Server. It is lightweight and performs well because it is optimized to access a SQL Server directly without adding an OLE DB or Open Database Connectivity (ODBC) layer. The following illustration contrasts the .NET Framework Data Provider for SQL Server with the .NET Framework Data Provider for OLE DB. The .NET Framework Data Provider for OLE DB communicates to an OLE DB data source through both the OLE DB Service component, which provides connection pooling and transaction services, and the OLE DB provider for the data source.

|  |
| --- |
| **Note:** |
| The .NET Framework Data Provider for ODBC has a similar architecture to the .NET Framework Data Provider for OLE DB; for example, it calls into an ODBC Service Component. |

Comparison of the dnprdnshort Data Provider for ssNoVersionr and the dnprdnshort Data Provider for OLE DB  


To use the .NET Framework Data Provider for SQL Server, you must have access to SQL Server 7.0 or later versions. The .NET Framework Data Provider for SQL Server classes are located in the [System.Data.SqlClient](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.aspx) namespace. For earlier versions of SQL Server, use the .NET Framework Data Provider for OLE DB with the SQL Server OLE DB provider [System.Data.OleDb](http://msdn.microsoft.com/en-us/library/system.data.oledb.aspx).

The .NET Framework Data Provider for SQL Server supports both local and distributed transactions. For distributed transactions, the .NET Framework Data Provider for SQL Server, by default, automatically enlists in a transaction and obtains transaction details from Windows Component Services or [System.Transactions](http://msdn.microsoft.com/en-us/library/system.transactions.aspx). For more information, see [Transactions and Concurrency (ADO.NET)](http://msdn.microsoft.com/en-us/library/777e5ebh.aspx).

The following code example shows how to include the System.Data.SqlClient namespace in your applications.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl47VisualBasic');)

Imports System.Data.SqlClient

C#

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl48CSharp');)

using System.Data.SqlClient;

 .NET Framework Data Provider for OLE DB

The .NET Framework Data Provider for OLE DB (OleDb) uses native OLE DB through COM interop to enable data access. The .NET Framework Data Provider for OLE DB supports both local and distributed transactions. For distributed transactions, the .NET Framework Data Provider for OLE DB, by default, automatically enlists in a transaction and obtains transaction details from Windows 2000 Component Services. For more information, see [Transactions and Concurrency (ADO.NET)](http://msdn.microsoft.com/en-us/library/777e5ebh.aspx).

The following table shows the providers that have been tested 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 |
| **Note:** | |
| Using an Access (Jet) database as a data source for multithreaded applications, such as ASP.NET applications, is not recommended. If you must use Jet as a data source for an ASP.NET application, realize that ASP.NET applications connecting to an Access database can encounter connection problems. For help troubleshooting connection problems using ASP.NET and an Access (Jet) database, see article Q316675, "PRB: Cannot Connect to Access Database from ASP.NET" in the [Microsoft Knowledge Base](http://support.microsoft.com/). | |

The .NET Framework Data Provider for OLE DB does not support OLE DB version 2.5 interfaces. OLE DB Providers that require support for OLE DB 2.5 interfaces will not function correctly with the .NET Framework Data Provider for OLE DB. This includes the Microsoft OLE DB provider for Exchange and the Microsoft OLE DB provider for Internet Publishing.

The .NET Framework Data Provider for OLE DB does not work with the OLE DB provider for ODBC (MSDASQL). To access an ODBC data source using ADO.NET, use the .NET Framework Data Provider for ODBC.

.NET Framework Data Provider for OLE DB classes are located in the [System.Data.OleDb](http://msdn.microsoft.com/en-us/library/system.data.oledb.aspx) namespace. The following code example shows how to include the System.Data.OleDb namespace in your applications.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl63VisualBasic');)

Imports System.Data.OleDb

C#

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl64CSharp');)

using System.Data.OleDb;

|  |
| --- |
| **Note:** |
| The .NET Framework Data Provider for OLE DB requires MDAC 2.6 or a later version, and MDAC 2.8 Service Pack 1 (SP1) is recommended. You can download MDAC 2.8 SP1 from the [Data Access and Storage Developer Center](http://go.microsoft.com/fwlink/?linkid=4173). |

 .NET Framework Data Provider for ODBC

The .NET Framework Data Provider for ODBC (Odbc) uses the native ODBC Driver Manager (DM) to enable data access. The ODBC data provider supports both local and distributed transactions. For distributed transactions, the ODBC data provider, by default, automatically enlists in a transaction and obtains transaction details from Windows 2000 Component Services. For more information, see [Transactions and Concurrency (ADO.NET)](http://msdn.microsoft.com/en-us/library/777e5ebh.aspx).

The following table shows the ODBC drivers tested with ADO.NET.

|  |
| --- |
| **Driver** |
| SQL Server |
| Microsoft ODBC for Oracle |
| Microsoft Access Driver (\*.mdb) |

.NET Framework Data Provider for ODBC classes are located in the [System.Data.Odbc](http://msdn.microsoft.com/en-us/library/system.data.odbc.aspx) namespace.

The following code example shows how to include the System.Data.Odbc namespace in your applications.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl77VisualBasic');)

Imports System.Data.Odbc

C#

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl78CSharp');)

using System.Data.Odbc;

|  |
| --- |
| **Note:** |
| The .NET Framework Data Provider for ODBC requires MDAC 2.6 or a later version, and MDAC 2.8 SP1 is recommended. You can download MDAC 2.8 SP1 from the [Data Access and Storage Developer Center](http://go.microsoft.com/fwlink/?linkid=4173). |

 .NET Framework Data Provider for Oracle

The .NET Framework Data Provider for Oracle (OracleClient) enables data access to Oracle data sources through Oracle client connectivity software. The data provider supports Oracle client software version 8.1.7 or a later version. The data provider supports both local and distributed transactions. For more information, see [Transactions and Concurrency (ADO.NET)](http://msdn.microsoft.com/en-us/library/777e5ebh.aspx).

The .NET Framework Data Provider for Oracle requires Oracle client software (version 8.1.7 or a later version) on the system before you can connect to an Oracle data source.

.NET Framework Data Provider for Oracle classes are located in the [System.Data.OracleClient](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.aspx) namespace and are contained in the System.Data.OracleClient.dll assembly. You must reference both the System.Data.dll and the System.Data.OracleClient.dll when you compile an application that uses the data provider.

The following code example shows how to include the System.Data.OracleClientnamespace in your applications.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl89VisualBasic');)

Imports System.Data

Imports System.Data.OracleClient

C#

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl90CSharp');)

using System.Data;

using System.Data.OracleClient;

 Choosing a .NET Framework Data Provider

Depending on the design and data source for your application, your choice of .NET Framework data provider can improve the performance, capability, and integrity of your application. The following table discusses the advantages and limitations of each .NET Framework data provider.

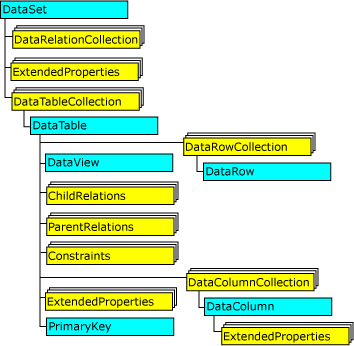
|  |  |
| --- | --- |
| **Provider** | **Notes** |
| .NET Framework Data Provider for SQL Server | Recommended for middle-tier applications that use Microsoft SQL Server 7.0 or a later version.  Recommended for single-tier applications that use Microsoft Database Engine (MSDE) or SQL Server 7.0 or a later version.  Recommended over use of the OLE DB provider for SQL Server (SQLOLEDB) with the .NET Framework Data Provider for OLE DB.  For SQL Server 6.5 and earlier, you must use the OLE DB provider for SQL Server with the .NET Framework Data Provider for OLE DB. |
| .NET Framework Data Provider for OLE DB | Recommended for middle-tier applications that use SQL Server 6.5 or earlier.  For SQL Server 7.0 or a later version, the .NET Framework Data Provider for SQL Server is recommended.  Also recommended for single-tier applications that use Microsoft Access databases. Use of an Access database for a middle-tier application is not recommended. |
| .NET Framework Data Provider for ODBC | Recommended for middle and single-tier applications that use ODBC data sources. |
| .NET Framework Data Provider for Oracle | Recommended for middle and single-tier applications that use Oracle data sources. |

 EntityClient Provider

The EntityClient provider is used for accessing data based on an Entity Data Model (EDM). Unlike the other .NET Framework data providers, it does not interact directly with a data source. Instead, it uses Entity SQL to communicate with the underlying data provider. For more information, see [EntityClient and Entity SQL](http://msdn.microsoft.com/en-us/library/bb386881.aspx).

**ADO.NET DataSets**

The [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) object is central to supporting disconnected, distributed data scenarios with ADO.NET. The **DataSet** is a memory-resident representation of data that provides a consistent relational programming model regardless of the data source. It can be used with multiple and differing data sources, with XML data, or to manage data local to the application. The **DataSet** represents a complete set of data, including related tables, constraints, and relationships among the tables. The following illustration shows the **DataSet** object model.

DataSet object model  


The methods and objects in a **DataSet** are consistent with those in the relational database model.

The **DataSet** can also persist and reload its contents as XML, and its schema as XML schema definition language (XSD) schema. For more information, see [Using XML in a DataSet (ADO.NET)](http://msdn.microsoft.com/en-us/library/84sxtbxh.aspx).

 The DataTableCollection

An ADO.NET **DataSet** contains a collection of zero or more tables represented by [DataTable](http://msdn.microsoft.com/en-us/library/system.data.datatable.aspx) objects. The [DataTableCollection](http://msdn.microsoft.com/en-us/library/system.data.datatablecollection.aspx) contains all the **DataTable** objects in a **DataSet**.

A **DataTable** is defined in the [System.Data](http://msdn.microsoft.com/en-us/library/system.data.aspx) namespace and represents a single table of memory-resident data. It contains a collection of columns represented by a [DataColumnCollection](http://msdn.microsoft.com/en-us/library/system.data.datacolumncollection.aspx), and constraints represented by a [ConstraintCollection](http://msdn.microsoft.com/en-us/library/system.data.constraintcollection.aspx), which together define the schema of the table. A **DataTable** also contains a collection of rows represented by the [DataRowCollection](http://msdn.microsoft.com/en-us/library/system.data.datarowcollection.aspx), which contains the data in the table. Along with its current state, a [DataRow](http://msdn.microsoft.com/en-us/library/system.data.datarow.aspx) retains both its current and original versions to identify changes to the values stored in the row.

 The DataView Class

A [DataView](http://msdn.microsoft.com/en-us/library/system.data.dataview.aspx) enables you to create different views of the data stored in a [DataTable](http://msdn.microsoft.com/en-us/library/system.data.datatable.aspx), a capability that is often used in data-binding applications. Using a [DataView](http://msdn.microsoft.com/en-us/library/system.data.dataview.aspx), you can expose the data in a table with different sort orders, and you can filter the data by row state or based on a filter expression. For more information, see [DataViews (ADO.NET)](http://msdn.microsoft.com/en-us/library/fdcwwhez.aspx).

 The DataRelationCollection

A **DataSet** contains relationships in its [DataRelationCollection](http://msdn.microsoft.com/en-us/library/system.data.datarelationcollection.aspx) object. A relationship, represented by the [DataRelation](http://msdn.microsoft.com/en-us/library/system.data.datarelation.aspx) object, associates rows in one **DataTable** with rows in another **DataTable**. A relationship is analogous to a join path that might exist between primary and foreign key columns in a relational database. A **DataRelation** identifies matching columns in two tables of a **DataSet**.

Relationships enable navigation from one table to another in a **DataSet**. The essential elements of a **DataRelation** are the name of the relationship, the name of the tables being related, and the related columns in each table. Relationships can be built with more than one column per table by specifying an array of [DataColumn](http://msdn.microsoft.com/en-us/library/system.data.datacolumn.aspx) objects as the key columns. When you add a relationship to the [DataRelationCollection](http://msdn.microsoft.com/en-us/library/system.data.datarelationcollection.aspx), you can optionally add a **UniqueKeyConstraint** and a **ForeignKeyConstraint** to enforce integrity constraints when changes are made to related column values.

For more information, see [Adding DataRelations (ADO.NET)](http://msdn.microsoft.com/en-us/library/ay82azad.aspx).

 XML

You can fill a **DataSet** from an XML stream or document. You can use the XML stream or document to supply to the **DataSet** either data, schema information, or both. The information supplied from the XML stream or document can be combined with existing data or schema information already present in the **DataSet**. For more information, see [Using XML in a DataSet (ADO.NET)](http://msdn.microsoft.com/en-us/library/84sxtbxh.aspx).

 ExtendedProperties

The **DataSet**, **DataTable**, and **DataColumn** all have an **ExtendedProperties** property. **ExtendedProperties** is a **PropertyCollection** where you can place custom information, such as the SELECT statement that was used to generate the result set, or the time when the data was generated. The **ExtendedProperties** collection is persisted with the schema information for the **DataSet**.

 LINQ to DataSet

LINQ to DataSet provides language-integrated querying capabilities for disconnected data stored in a DataSet. LINQ to DataSet uses standard LINQ syntax and provides compile-time syntax checking, static typing, and IntelliSense support when you are using the Visual Studio IDE.

For more information, see [LINQ to DataSet](http://msdn.microsoft.com/en-us/library/bb386977.aspx).

**Side-by-Side Execution in ADO.NET**

Side-by-side execution in the .NET Framework is the ability to execute an application on a computer that has multiple versions of the .NET Framework installed, exclusively using the version for which the application was compiled. For detailed information about configuring side-by-side execution, see [Side-by-Side Execution](http://msdn.microsoft.com/en-us/library/8477k21c.aspx).

An application compiled by using one version of the .NET Framework can run on a different version of the .NET Framework. However, we recommend that you compile a version of the application for each installed version of the .NET Framework, and run them separately. In either scenario, you should be aware of changes in ADO.NET between releases that can affect the forward compatibility or backward compatibility of your application.

 Forward Compatibility and Backward Compatibility

Forward compatibility means that an application can be compiled with an earlier version of the .NET Framework, but will still run successfully on a later version of the .NET Framework. ADO.NET code written for the .NET Framework version 1.1 is forward compatible with later versions.

Backward compatibility means that an application is compiled for a newer version of the .NET Framework, but continues to run on earlier versions of the .NET Framework without any loss of functionality. Of course, this will not be the case for features introduced in a new version of the .NET Framework.

 The .NET Framework Data Provider for ODBC

Starting with version 1.1, the .NET Framework Data Provider for ODBC ([System.Data.Odbc](http://msdn.microsoft.com/en-us/library/system.data.odbc.aspx)) is included as a part of the .NET Framework. The ODBC data provider is available to .NET Framework version 1.0 developers as a Web download from the [Data Access and Storage Developer Center](http://go.microsoft.com/fwlink/?linkid=4173). The namespace for the downloaded .NET Framework Data Provider for ODBC is **Microsoft.Data.Odbc**.

If you have an application developed for the .NET Framework version 1.0 that uses the ODBC data provider to connect to your data source, and you want to run that application on the .NET Framework version 1.1 or a later version, you must update the namespace for the ODBC data provider to **System.Data.Odbc**. You then must recompile it for the newer version of the .NET Framework.

If you have an application developed for the .NET Framework version 2.0 or later that uses the ODBC data provider to connect to your data source, and you want to run that application on the .NET Framework version 1.0, you must download the ODBC data provider and install it on the .NET Framework version 1.0 system. You then must change the namespace for the ODBC data provider to **Microsoft.Data.Odbc**, and recompile the application for the .NET Framework version 1.0.

 The .NET Framework Data Provider for Oracle

Starting with version 1.1, the .NET Framework Data Provider for Oracle ([System.Data.OracleClient](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.aspx)) is included as a part of the .NET Framework. The data provider is available to .NET Framework version 1.0 developers as a Web download from the [Data Access and Storage Developer Center](http://go.microsoft.com/fwlink/?linkid=4173).

If you have an application developed for the .NET Framework version 2.0 or later that uses the data provider to connect to your data source, and you want to run that application on the .NET Framework version 1.0, you must download the data provider and install it on the .NET Framework version 1.0 system.

 Code Access Security

The .NET Framework data providers in the .NET Framework version 1.0 ([System.Data.SqlClient](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.aspx), [System.Data.OleDb](http://msdn.microsoft.com/en-us/library/system.data.oledb.aspx)) are required to run with FullTrust permission. Any attempt to use the .NET Framework k data providers from the .NET Framework version 1.0 in a zone with less than FullTrust permission causes a [SecurityException](http://msdn.microsoft.com/en-us/library/system.security.securityexception.aspx).

However, starting with the .NET Framework version 2.0, all of the .NET Framework data providers can be used in partially trusted zones. In addition, a new security feature was added to the .NET Framework data providers in the .NET Framework version 1.1. This feature enables you to restrict what connection strings can be used in a particular security zone. You can also disable the use of blank passwords for a particular security zone. For more information, see [Code Access Security and ADO.NET](http://msdn.microsoft.com/en-us/library/0x4t63kb.aspx).

Because each installation of the .NET Framework has a separate Security.config file, there are no compatibility issues with security settings. However, if your application depends on the additional security capabilities of ADO.NET included in the .NET Framework version 1.1 and later, you will not be able to distribute it to a version 1.0 system.

 SqlCommand Execution

Starting with the .NET Framework version 1.1, the way that [ExecuteReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.executereader.aspx) executes commands at the data source was changed.

In the .NET Framework version 1.0, [ExecuteReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.executereader.aspx) executed all commands in the context of the **sp\_executesql** stored procedure. As a result, commands that affect the state of the connection (for example, SET NOCOUNT ON), only apply to the execution of the current command. The state of the connection is not modified for any subsequent commands executed while the connection is open.

In the .NET Framework version 1.1 and later, [ExecuteReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.executereader.aspx) only executes a command in the context of the **sp\_executesql** stored procedure if the command contains parameters, which provides a performance benefit. As a result, if a command affecting the state of the connection is included in a non-parameterized command, it modifies the state of the connection for all subsequent commands executed while the connection is open.

Consider the following batch of commands executed in a call to [ExecuteReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.executereader.aspx).

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl40other');)

SET NOCOUNT ON;

SELECT \* FROM dbo.Customers;

In the .NET Framework version 1.1 and later, NOCOUNT will remain ON for any subsequent commands executed while the connection is open. In the .NET Framework version 1.0, NOCOUNT is only ON for the current command execution.

This change can affect both the forward and backward compatibility of your application if you depend on the behavior of [ExecuteReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.executereader.aspx) for either version of the .NET Framework.

For applications that run on both earlier and later versions of the .NET Framework, you can write your code to make sure that the behavior is the same regardless of the version you are running on. If you want to make sure that a command modifies the state of the connection for all subsequent commands, we recommend that you execute your command using [ExecuteNonQuery](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.executenonquery.aspx). If you want to make sure that a command does not modify the connection for all subsequent commands, we recommend that you include the commands to reset the state of the connection in your command. For example:

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl43other');)

SET NOCOUNT ON;

SELECT \* FROM dbo.Customers;

SET NOCOUNT OFF;

 Microsoft SQL Server Native Client

Microsoft SQL Server Native Client contains the SQL OLE DB provider and SQL ODBC driver in one native dynamic link library (DLL) supporting applications using native-code APIs (ODBC, OLE DB and ADO) to Microsoft SQL Server. SQL Server Native Client should be used rather than Microsoft Data Access Components (MDAC) to create new applications or enhance existing applications that need to take advantage of features that were introduced in SQL Server 2005, such as Multiple Active Result Sets (MARS), Query Notifications, User-Defined Types (UDT), and XML data type support.

 Microsoft Data Access Components (MDAC)

The .NET Framework data providers for OLE DB and ODBC require MDAC 2.6 or a later version in all versions of the .NET Framework, and MDAC 2.8 SP1 is recommended. Although this requirement introduces no side-by-side execution issues, notice that MDAC does not currently support side-by-side execution. Therefore, it is important to verify that your application will continue to function correctly with the new version before upgrading the MDAC components for your installation.

For more information about MDAC, see the [Data Access and Storage Developer Center](http://go.microsoft.com/fwlink/?linkid=4173).

 Windows Data Access Components (Windows DAC)

Windows Data Access Components (Windows DAC) 6.0 is a set of technologies included in Windows Vista to provide access to information across the enterprise. These technologies include the latest versions of the data access technologies included in MDAC: Microsoft ActiveX Data Objects (ADO), OLE DB, and Microsoft Open Database Connectivity (ODBC).

For more information about Windows DAC, see [Windows Data Access Components SDK Overview](http://go.microsoft.com/fwlink/?LinkId=120501).

**ADO.NET Code Examples**

The code listings in this topic demonstrate how to retrieve data from a database by using the following ADO.NET technologies:

* ADO.NET data providers:
  + [.NET Framework Data Provider for SQL Server](http://msdn.microsoft.com/en-us/library/dw70f090.aspx#_SqlClient#_SqlClient) (System.Data.SqlClient)
  + [.NET Framework Data Provider for OLE DB](http://msdn.microsoft.com/en-us/library/dw70f090.aspx#_OleDb#_OleDb) (System.Data.OleDb)
  + [.NET Framework Data Provider for ODBC](http://msdn.microsoft.com/en-us/library/dw70f090.aspx#_Odbc#_Odbc) (System.Data.Odbc)
  + [.NET Framework Data Provider for Oracle](http://msdn.microsoft.com/en-us/library/dw70f090.aspx#_OracleClient#_OracleClient) (System.Data.OracleClient)
* ADO.NET Entity Framework:
  + [LINQ to Entities](http://msdn.microsoft.com/en-us/library/dw70f090.aspx#_LINQ#_LINQ)
  + [Typed ObjectQuery](http://msdn.microsoft.com/en-us/library/dw70f090.aspx#_QBM#_QBM)
  + [EntityClient data provider](http://msdn.microsoft.com/en-us/library/dw70f090.aspx#_EntityClient#_EntityClient) (System.Data.EntityClient)
* [LINQ to SQL](http://msdn.microsoft.com/en-us/library/dw70f090.aspx#_LINQ2SQL#_LINQ2SQL)

 ADO.NET Data Provider Examples

The following code listings demonstrate how to retrieve data from a database using ADO.NET data providers. The data is returned in a DataReader. For more information, see [Retrieving Data Using a DataReader (ADO.NET)](http://msdn.microsoft.com/en-us/library/haa3afyz.aspx).

### SqlClient

The code in this example assumes that you can connect to the Northwind sample database on Microsoft SQL Server 7.0 or a later version. The code creates a [SqlCommand](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlcommand.aspx) to select rows from the Products table, adding a [SqlParameter](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlparameter.aspx) to restrict the results to rows with a UnitPrice greater than the specified parameter value, in this case 5. The [SqlConnection](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqlconnection.aspx) is opened inside of a using block, which ensures that resources are closed and disposed when the code exits. The code executes the command by using a [SqlDataReader](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.sqldatareader.aspx), and displays the results in the console window.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl30VisualBasic');)

Option Explicit On

Option Strict On

Imports System

Imports System.Data

Imports System.Data.SqlClient

Public Class Program

Public Shared Sub Main()

Dim connectionString As String = \_

"Data Source=(local);Initial Catalog=Northwind;" \_

& "Integrated Security=true"

' Provide the query string with a parameter placeholder.

Dim queryString As String = \_

"SELECT ProductID, UnitPrice, ProductName from dbo.Products " \_

& "WHERE UnitPrice > @pricePoint " \_

& "ORDER BY UnitPrice DESC;"

' Specify the parameter value.

Dim paramValue As Integer = 5

' Create and open the connection in a using block. This

' ensures that all resources will be closed and disposed

' when the code exits.

Using connection As New SqlConnection(connectionString)

' Create the Command and Parameter objects.

Dim command As New SqlCommand(queryString, connection)

command.Parameters.AddWithValue("@pricePoint", paramValue)

' Open the connection in a try/catch block.

' Create and execute the DataReader, writing the result

' set to the console window.

Try

connection.Open()

Dim dataReader As SqlDataReader = \_

command.ExecuteReader()

Do While dataReader.Read()

Console.WriteLine( \_

vbTab & "{0}" & vbTab & "{1}" & vbTab & "{2}", \_

dataReader(0), dataReader(1), dataReader(2))

Loop

dataReader.Close()

Catch ex As Exception

Console.WriteLine(ex.Message)

End Try

Console.ReadLine()

End Using

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl31CSharp');)

using System;

using System.Data;

using System.Data.SqlClient;

class Program

{

static void Main()

{

string connectionString =

"Data Source=(local);Initial Catalog=Northwind;"

+ "Integrated Security=true";

// Provide the query string with a parameter placeholder.

string queryString =

"SELECT ProductID, UnitPrice, ProductName from dbo.products "

+ "WHERE UnitPrice > @pricePoint "

+ "ORDER BY UnitPrice DESC;";

// Specify the parameter value.

int paramValue = 5;

// Create and open the connection in a using block. This

// ensures that all resources will be closed and disposed

// when the code exits.

using (SqlConnection connection =

new SqlConnection(connectionString))

{

// Create the Command and Parameter objects.

SqlCommand command = new SqlCommand(queryString, connection);

command.Parameters.AddWithValue("@pricePoint", paramValue);

// Open the connection in a try/catch block.

// Create and execute the DataReader, writing the result

// set to the console window.

try

{

connection.Open();

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

Console.WriteLine("\t{0}\t{1}\t{2}",

reader[0], reader[1], reader[2]);

}

reader.Close();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

Console.ReadLine();

}

}

[[Top]](http://msdn.microsoft.com/en-us/library/dw70f090.aspx#_TOP#_TOP)

### OleDb

The code in this example assumes that you can connect to the Microsoft Access Northwind sample database. The code creates a [OleDbCommand](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbcommand.aspx) to select rows from the Products table, adding a [OleDbParameter](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbparameter.aspx) to restrict the results to rows with a UnitPrice greater than the specified parameter value, in this case 5. The [OleDbConnection](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbconnection.aspx) is opened inside of a using block, which ensures that resources are closed and disposed when the code exits. The code executes the command by using a [OleDbDataReader](http://msdn.microsoft.com/en-us/library/system.data.oledb.oledbdatareader.aspx), and displays the results in the console window.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl36VisualBasic');)

Option Explicit On

Option Strict On

Imports System

Imports System.Data

Imports System.Data.OleDb

Public Class Program

Public Shared Sub Main()

' The connection string assumes that the Access

' Northwind.mdb is located in the c:\Data folder.

Dim connectionString As String = \_

"Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" \_

& "c:\Data\Northwind.mdb;User Id=admin;Password=;"

' Provide the query string with a parameter placeholder.

Dim queryString As String = \_

"SELECT ProductID, UnitPrice, ProductName from Products " \_

& "WHERE UnitPrice > ? " \_

& "ORDER BY UnitPrice DESC;"

' Specify the parameter value.

Dim paramValue As Integer = 5

' Create and open the connection in a using block. This

' ensures that all resources will be closed and disposed

' when the code exits.

Using connection As New OleDbConnection(connectionString)

' Create the Command and Parameter objects.

Dim command As New OleDbCommand(queryString, connection)

command.Parameters.AddWithValue("@pricePoint", paramValue)

' Open the connection in a try/catch block.

' Create and execute the DataReader, writing the result

' set to the console window.

Try

connection.Open()

Dim dataReader As OleDbDataReader = \_

command.ExecuteReader()

Do While dataReader.Read()

Console.WriteLine( \_

vbTab & "{0}" & vbTab & "{1}" & vbTab & "{2}", \_

dataReader(0), dataReader(1), dataReader(2))

Loop

dataReader.Close()

Catch ex As Exception

Console.WriteLine(ex.Message)

End Try

Console.ReadLine()

End Using

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl37CSharp');)

using System;

using System.Data;

using System.Data.OleDb;

class Program

{

static void Main()

{

// The connection string assumes that the Access

// Northwind.mdb is located in the c:\Data folder.

string connectionString =

"Provider=Microsoft.Jet.OLEDB.4.0;Data Source="

+ "c:\\Data\\Northwind.mdb;User Id=admin;Password=;";

// Provide the query string with a parameter placeholder.

string queryString =

"SELECT ProductID, UnitPrice, ProductName from products "

+ "WHERE UnitPrice > ? "

+ "ORDER BY UnitPrice DESC;";

// Specify the parameter value.

int paramValue = 5;

// Create and open the connection in a using block. This

// ensures that all resources will be closed and disposed

// when the code exits.

using (OleDbConnection connection =

new OleDbConnection(connectionString))

{

// Create the Command and Parameter objects.

OleDbCommand command = new OleDbCommand(queryString, connection);

command.Parameters.AddWithValue("@pricePoint", paramValue);

// Open the connection in a try/catch block.

// Create and execute the DataReader, writing the result

// set to the console window.

try

{

connection.Open();

OleDbDataReader reader = command.ExecuteReader();

while (reader.Read())

{

Console.WriteLine("\t{0}\t{1}\t{2}",

reader[0], reader[1], reader[2]);

}

reader.Close();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

Console.ReadLine();

}

}

[[Top]](http://msdn.microsoft.com/en-us/library/dw70f090.aspx#_TOP#_TOP)

### Odbc

The code in this example assumes that you can connect to the Microsoft Access Northwind sample database. The code creates a [OdbcCommand](http://msdn.microsoft.com/en-us/library/system.data.odbc.odbccommand.aspx) to select rows from the Products table, adding a [OdbcParameter](http://msdn.microsoft.com/en-us/library/system.data.odbc.odbcparameter.aspx) to restrict the results to rows with a UnitPrice greater than the specified parameter value, in this case 5. The [OdbcConnection](http://msdn.microsoft.com/en-us/library/system.data.odbc.odbcconnection.aspx) is opened inside of a using block, which ensures that resources are closed and disposed when the code exits. The executes the command by using a [OdbcDataReader](http://msdn.microsoft.com/en-us/library/system.data.odbc.odbcdatareader.aspx), and displays the results in the console window.

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl42VisualBasic');)

Option Explicit On

Option Strict On

Imports System

Imports System.Data

Imports System.Data.Odbc

Public Class Program

Public Shared Sub Main()

' The connection string assumes that the Access

' Northwind.mdb is located in the c:\Data folder.

Dim connectionString As String = \_

"Driver={Microsoft Access Driver (\*.mdb)};" \_

& "Dbq=c:\Data\Northwind.mdb;Uid=Admin;Pwd=;"

' Provide the query string with a parameter placeholder.

Dim queryString As String = \_

"SELECT ProductID, UnitPrice, ProductName from Products " \_

& "WHERE UnitPrice > ? " \_

& "ORDER BY UnitPrice DESC;"

' Specify the parameter value.

Dim paramValue As Integer = 5

' Create and open the connection in a using block. This

' ensures that all resources will be closed and disposed

' when the code exits.

Using connection As New OdbcConnection(connectionString)

' Create the Command and Parameter objects.

Dim command As New OdbcCommand(queryString, connection)

command.Parameters.AddWithValue("@pricePoint", paramValue)

' Open the connection in a try/catch block.

' Create and execute the DataReader, writing the result

' set to the console window.

Try

connection.Open()

Dim dataReader As OdbcDataReader = \_

command.ExecuteReader()

Do While dataReader.Read()

Console.WriteLine( \_

vbTab & "{0}" & vbTab & "{1}" & vbTab & "{2}", \_

dataReader(0), dataReader(1), dataReader(2))

Loop

dataReader.Close()

Catch ex As Exception

Console.WriteLine(ex.Message)

End Try

Console.ReadLine()

End Using

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl43CSharp');)

using System;

using System.Data;

using System.Data.Odbc;

class Program

{

static void Main()

{

// The connection string assumes that the Access

// Northwind.mdb is located in the c:\Data folder.

string connectionString =

"Driver={Microsoft Access Driver (\*.mdb)};"

+ "Dbq=c:\\Data\\Northwind.mdb;Uid=Admin;Pwd=;";

// Provide the query string with a parameter placeholder.

string queryString =

"SELECT ProductID, UnitPrice, ProductName from products "

+ "WHERE UnitPrice > ? "

+ "ORDER BY UnitPrice DESC;";

// Specify the parameter value.

int paramValue = 5;

// Create and open the connection in a using block. This

// ensures that all resources will be closed and disposed

// when the code exits.

using (OdbcConnection connection =

new OdbcConnection(connectionString))

{

// Create the Command and Parameter objects.

OdbcCommand command = new OdbcCommand(queryString, connection);

command.Parameters.AddWithValue("@pricePoint", paramValue);

// Open the connection in a try/catch block.

// Create and execute the DataReader, writing the result

// set to the console window.

try

{

connection.Open();

OdbcDataReader reader = command.ExecuteReader();

while (reader.Read())

{

Console.WriteLine("\t{0}\t{1}\t{2}",

reader[0], reader[1], reader[2]);

}

reader.Close();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

Console.ReadLine();

}

}

[[Top]](http://msdn.microsoft.com/en-us/library/dw70f090.aspx#_TOP#_TOP)

### OracleClient

The code in this example assumes a connection to DEMO.CUSTOMER on an Oracle server. You must also add a reference to the System.Data.OracleClient.dll. The code returns the data in an [OracleDataReader](http://msdn.microsoft.com/en-us/library/system.data.oracleclient.oracledatareader.aspx).

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl45VisualBasic');)

Option Explicit On

Option Strict On

Imports System

Imports System.Data

Imports System.Data.OracleClient

Public Class Program

Public Shared Sub Main()

Dim connectionString As String = \_

"Data Source=ThisOracleServer;Integrated Security=yes;"

Dim queryString As String = \_

"SELECT CUSTOMER\_ID, NAME FROM DEMO.CUSTOMER"

Using connection As New OracleConnection(connectionString)

Dim command As OracleCommand = connection.CreateCommand()

command.CommandText = queryString

Try

connection.Open()

Dim dataReader As OracleDataReader = \_

command.ExecuteReader()

Do While dataReader.Read()

Console.WriteLine(vbTab & "{0}" & vbTab & "{1}", \_

dataReader(0), dataReader(1))

Loop

dataReader.Close()

Catch ex As Exception

Console.WriteLine(ex.Message)

End Try

End Using

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl46CSharp');)

using System;

using System.Data;

using System.Data.OracleClient;

class Program

{

static void Main()

{

string connectionString =

"Data Source=ThisOracleServer;Integrated Security=yes;";

string queryString =

"SELECT CUSTOMER\_ID, NAME FROM DEMO.CUSTOMER";

using (OracleConnection connection =

new OracleConnection(connectionString))

{

OracleCommand command = connection.CreateCommand();

command.CommandText = queryString;

try

{

connection.Open();

OracleDataReader reader = command.ExecuteReader();

while (reader.Read())

{

Console.WriteLine("\t{0}\t{1}",

reader[0], reader[1]);

}

reader.Close();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

[[Top]](http://msdn.microsoft.com/en-us/library/dw70f090.aspx#_TOP#_TOP)

 Entity Framework Examples

The following code listings demonstrate how to retrieve data from a data source by querying entities in an Entity Data Model (EDM). These examples use the [Northwind model](http://msdn.microsoft.com/en-us/library/bb399306.aspx). For more information, see [Introducing the Entity Framework](http://msdn.microsoft.com/en-us/library/bb399567.aspx).

### LINQ to Entities

The code in this example uses a LINQ query to return data as Categories objects, which are projected as an anonymous type that contains only the CategoryID and CategoryName properties. For more information, see [LINQ to Entities Overview](http://msdn.microsoft.com/en-us/library/bb386992.aspx).

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl68VisualBasic');)

Option Explicit On

Option Strict On

Imports System

Imports System.Linq

Imports System.Data.Objects

Imports NorthwindModel

Class LinqSample

Public Shared Sub ExecuteQuery()

Using context As NorthwindEntities = New NorthwindEntities()

Try

Dim query = From category In context.Categories \_

Select New With \_

{ \_

.categoryID = category.CategoryID, \_

.categoryName = category.CategoryName \_

}

For Each categoryInfo In query

Console.WriteLine(vbTab & "{0}" & vbTab & "{1}", \_

categoryInfo.categoryID, categoryInfo.categoryName)

Next

Catch ex As Exception

Console.WriteLine(ex.Message)

End Try

End Using

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl69CSharp');)

using System;

using System.Linq;

using System.Data.Objects;

using NorthwindModel;

class LinqSample

{

public static void ExecuteQuery()

{

using (NorthwindEntities context = new NorthwindEntities())

{

try

{

var query = from category in context.Categories

select new

{

categoryID = category.CategoryID,

categoryName = category.CategoryName

};

foreach (var categoryInfo in query)

{

Console.WriteLine("\t{0}\t{1}",

categoryInfo.categoryID, categoryInfo.categoryName);

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

[[Top]](http://msdn.microsoft.com/en-us/library/dw70f090.aspx#_TOP#_TOP)

### Typed ObjectQuery

The code in this example uses an [ObjectQuery<(Of <(T>)>)](http://msdn.microsoft.com/en-us/library/bb345303.aspx) to return data as Categories objects. For more information, see [Object Queries (Entity Framework)](http://msdn.microsoft.com/en-us/library/bb896241.aspx).

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl74VisualBasic');)

Option Explicit On

Option Strict On

Imports System

Imports System.Data.Objects

Imports NorthwindModel

Class ObjectQuerySample

Public Shared Sub ExecuteQuery()

Using context As NorthwindEntities = New NorthwindEntities()

Dim categoryQuery As ObjectQuery(Of Categories) = context.Categories

For Each category As Categories In \_

categoryQuery.Execute(MergeOption.AppendOnly)

Console.WriteLine(vbTab & "{0}" & vbTab & "{1}", \_

category.CategoryID, category.CategoryName)

Next

End Using

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl75CSharp');)

using System;

using System.Data.Objects;

using NorthwindModel;

class ObjectQuerySample

{

public static void ExecuteQuery()

{

using (NorthwindEntities context = new NorthwindEntities())

{

ObjectQuery<Categories> categoryQuery = context.Categories;

foreach (Categories category in

categoryQuery.Execute(MergeOption.AppendOnly))

{

Console.WriteLine("\t{0}\t{1}",

category.CategoryID, category.CategoryName);

}

}

}

}

[[Top]](http://msdn.microsoft.com/en-us/library/dw70f090.aspx#_TOP#_TOP)

### EntityClient

The code in this example uses an [EntityCommand](http://msdn.microsoft.com/en-us/library/system.data.entityclient.entitycommand.aspx) to execute an Entity SQL query. This query returns a list of records that represent instances of the Categories entity type. An [EntityDataReader](http://msdn.microsoft.com/en-us/library/system.data.entityclient.entitydatareader.aspx) is used to access data records in the result set. For more information, see [EntityClient Provider for Entity Framework](http://msdn.microsoft.com/en-us/library/bb738561.aspx).

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl79VisualBasic');)

Option Explicit On

Option Strict On

Imports System

Imports System.Data

Imports System.Data.Common

Imports System.Data.EntityClient

Imports NorthwindModel

Class EntityClientSample

Public Shared Sub ExecuteQuery()

Dim queryString As String = \_

"SELECT c.CategoryID, c.CategoryName " & \_

"FROM NorthwindEntities.Categories AS c"

Using conn As EntityConnection = \_

New EntityConnection("name=NorthwindEntities")

Try

conn.Open()

Using query As EntityCommand = \_

New EntityCommand(queryString, conn)

Using rdr As DbDataReader = \_

query.ExecuteReader(CommandBehavior.SequentialAccess)

While rdr.Read()

Console.WriteLine(vbTab & "{0}" & vbTab & "{1}", \_

rdr(0), rdr(1))

End While

End Using

End Using

Catch ex As Exception

Console.WriteLine(ex.Message)

End Try

End Using

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl80CSharp');)

using System;

using System.Data;

using System.Data.Common;

using System.Data.EntityClient;

using NorthwindModel;

class EntityClientSample

{

public static void ExecuteQuery()

{

string queryString =

@"SELECT c.CategoryID, c.CategoryName

FROM NorthwindEntities.Categories AS c";

using (EntityConnection conn =

new EntityConnection("name=NorthwindEntities"))

{

try

{

conn.Open();

using (EntityCommand query = new EntityCommand(queryString, conn))

{

using (DbDataReader rdr =

query.ExecuteReader(CommandBehavior.SequentialAccess))

{

while (rdr.Read())

{

Console.WriteLine("\t{0}\t{1}", rdr[0], rdr[1]);

}

}

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

[[Top]](http://msdn.microsoft.com/en-us/library/dw70f090.aspx#_TOP#_TOP)

 LINQ to SQL

The code in this example uses a LINQ query to return data as Categories objects, which are projected as an anonymous type that contains only the CategoryID and CategoryName properties. This example is based on the Northwind data context. For more information, see [Getting Started (LINQ to SQL)](http://msdn.microsoft.com/en-us/library/bb399398.aspx).

Visual Basic

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl87VisualBasic');)

Option Explicit On

Option Strict On

Imports System

Imports System.Collections.Generic

Imports System.Linq

Imports System.Text

Imports Northwind

Class LinqSqlSample

Public Shared Sub ExecuteQuery()

Using db As NorthwindDataContext = New NorthwindDataContext()

Try

Dim query = From category In db.Categories \_

Select New With \_

{ \_

.categoryID = category.CategoryID, \_

.categoryName = category.CategoryName \_

}

For Each categoryInfo In query

Console.WriteLine(vbTab & "{0}" & vbTab & "{1}", \_

categoryInfo.categoryID, categoryInfo.categoryName)

Next

Catch ex As Exception

Console.WriteLine(ex.Message)

End Try

End Using

End Sub

End Class

C#

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl88CSharp');)

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using Northwind;

class LinqSqlSample

{

public static void ExecuteQuery()

{

using (NorthwindDataContext db = new NorthwindDataContext())

{

try

{

var query = from category in db.Categories

select new

{

categoryID = category.CategoryID,

categoryName = category.CategoryName

};

foreach (var categoryInfo in query)

{

Console.WriteLine("vbTab {0} vbTab {1}",

categoryInfo.categoryID, categoryInfo.categoryName);

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

**What's New in ADO.NET**

The following features are new in ADO.NET with the Service Pack 1 release of the .NET Framework version 3.5. These include the Entity Framework as well as improvements to LINQ to DataSet and LINQ to SQL. The .NET Framework Data Provider for SQL Server (SqlClient) has been enhanced to provide support for SQL Server 2008.

 SqlClient Support for SQL Server 2008

Starting with the .NET Framework version 3.5 Service Pack (SP) 1, the .NET Framework Data Provider for SQL Server ([System.Data.SqlClient](http://msdn.microsoft.com/en-us/library/system.data.sqlclient.aspx)) provides full support for all the new features of the SQL Server 2008 Database Engine. You must install the .NET Framework 3.5 SP1 (or later) to use these new features with SqlClient. For more information about .NET Framework support for SQL Server 2008, see [New Features in SQL Server 2008 (ADO.NET)](http://msdn.microsoft.com/en-us/library/bb675218.aspx). For more information about new Database Engine features in SQL Server 2008, see [What's New (SQL Server 2008)](http://go.microsoft.com/fwlink/?LinkId=98359) in SQL Server 2008 Books Online.

 ADO.NET Data Platform

The Data Platform is a multi-release strategy to decrease the amount of coding and maintenance required for developers by enabling them to program against conceptual entity data models. In the .NET Framework version 3.5 Service Pack (SP) 1, the ADO.NET Entity Framework provides the following Data Platform components:

|  |  |
| --- | --- |
| **Component** | **Description** |
| Entity Data Model (EDM) | A design specification that defines application data as sets of entities and relationships. Data in this model supports object-relational mapping and data programmability across application boundaries.  EDM data types and relationships are defined in an conceptual model. This is an XML schema written in conceptual schema definition language (CSDL). The conceptual model is used to build programmable classes that represent application data. Developers can extend these objects as required to support various application needs. For more information, see [Entity Data Model](http://msdn.microsoft.com/en-us/library/bb387122.aspx). |
| Object Services | Allows programmers to interact with the conceptual model through a set of common language runtime (CLR) classes. These classes can be automatically generated from the conceptual model or can be developed independently to reflect the structure of the conceptual model. Object Services also provides infrastructure support for the Entity Framework, including services such as state management, change tracking, identity resolution, loading and navigating relationships, propagating object changes to database modifications, and query building support for Entity SQL. For more information, see [Object Services Overview (Entity Framework)](http://msdn.microsoft.com/en-us/library/bb386871.aspx) |
| LINQ to Entities | A language-integrated query (LINQ) implementation that allows developers to create strongly-typed queries against the Entity Framework object context by using LINQ expressions and LINQ standard query operators. LINQ to Entities allows developers to work against a conceptual model with a very flexible object-relational mapping across Microsoft SQL Server and third-party databases. For more information, see [LINQ to Entities](http://msdn.microsoft.com/en-us/library/bb386964.aspx). |
| Entity SQL | A text-based query language designed to interact with an Entity Data Model. Entity SQL is an SQL dialect that contains constructs for querying in terms of higher-level modeling concepts, such as inheritance, complex types, and explicit relationships. Developers can also use Entity SQL directly with Object Services. For more information, see [Entity SQL Language Reference](http://msdn.microsoft.com/en-us/library/bb399560.aspx). |
| EntityClient | A new .NET Framework data provider used for interacting with an Entity Data Model. EntityClient follows the .NET Framework data provider pattern of exposing [EntityConnection](http://msdn.microsoft.com/en-us/library/system.data.entityclient.entityconnection.aspx) and [EntityCommand](http://msdn.microsoft.com/en-us/library/system.data.entityclient.entitycommand.aspx) objects that return an [EntityDataReader](http://msdn.microsoft.com/en-us/library/system.data.entityclient.entitydatareader.aspx). EntityClient works with the Entity SQL language, providing flexible mapping to storage-specific data providers. For more information, see [EntityClient and Entity SQL](http://msdn.microsoft.com/en-us/library/bb386881.aspx). |
| ADO.NET Data Services | The ADO.NET Data Services framework is used to deploy data services on the Web or on an intranet. The data is structured as entities and relationships according to the specifications of the Entity Data Model. Data deployed on this model is addressable by standard HTTP protocol. For more information, see [ADO.NET Data Services Framework](http://msdn.microsoft.com/en-us/library/cc668792.aspx). |
| Entity Data Model Tools | The Entity Framework provides command-line tools, wizards, and designers to facilitate building EDM applications. The EntityDataSource control supports data binding scenarios based on the EDM. The programming surface of the EntityDataSource control is similar to other data source controls in Visual Studio. For more information, see [Entity Data Model Tools](http://msdn.microsoft.com/en-us/library/bb399249.aspx). |

For more information about the ADO.NET Entity Framework, see [Introducing the Entity Framework](http://msdn.microsoft.com/en-us/library/bb399567.aspx).

 LINQ to DataSet

LINQ to DataSet provides LINQ capabilities for disconnected data stored in a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx). For more information, see [LINQ to DataSet Overview](http://msdn.microsoft.com/en-us/library/bb399399.aspx).

 LINQ to SQL

LINQ to SQL supports queries against an object model that is mapped to the data structures of a Microsoft SQL Server database without using an intermediate conceptual model. Each table is represented by a separate class, tightly coupling the object model to the database schema. LINQ to SQL translates language-integrated queries in the object model into Transact-SQL and sends them to the database for execution. When the database returns the results, LINQ to SQL translates the results back into objects. Beginning with the .NET Framework version 3.5 SP1, LINQ to SQL supports many of the new SQL data types introduced with Microsoft SQL Server 2008. For more information, see [LINQ to SQL](http://msdn.microsoft.com/en-us/library/bb386976.aspx).

**Securing ADO.NET Applications**

Writing a secure ADO.NET application involves more than avoiding common coding pitfalls such as not validating user input. An application that accesses data has many potential points of failure that an attacker can exploit to retrieve, manipulate, or destroy sensitive data. It is therefore important to understand all aspects of security, from the process of threat modeling during the design phase of your application, to its eventual deployment and ongoing maintenance.

The .NET Framework provides many useful classes, services, and tools for securing and administering database applications. The common language runtime (CLR) provides a type-safe environment for code to run in, with code access security (CAS) to restrict further the permissions of managed code. Following secure data access coding practices limits the damage that can be inflicted by a potential attacker.

Writing secure code does not guard against self-inflicted security holes when working with unmanaged resources such as databases. Most server databases, such as SQL Server, have their own security systems, which enhance security when implemented correctly. However, even a data source with a robust security system can be victimized in an attack if it is not configured appropriately.

**Data Type Mappings in ADO.NET**

The .NET Framework is based on the common type system, which defines how types are declared, used, and managed in the runtime. It consists of both value types and reference types, which all derive from the [Object](http://msdn.microsoft.com/en-us/library/system.object.aspx) base type. When working with a data source, the data type is inferred from the data provider if it is not explicitly specified. For example, a [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx) object is independent of any specific data source. Data in a DataSet is retrieved from a data source, and changes are persisted back to the data source by using a DataAdapter. This means that when a DataAdapter fills a [DataTable](http://msdn.microsoft.com/en-us/library/system.data.datatable.aspx) in a DataSet with values from a data source, the resulting data types of the columns in the DataTable are .NET Framework types, instead of types specific to the .NET Framework data provider that is used to connect to the data source.

Likewise, when a DataReader returns a value from a data source, the resulting value is stored in a local variable that has a .NET Framework type. For both the Fill operations of the DataAdapter and the Get methods of the DataReader, the .NET Framework type is inferred from the value returned from the .NET Framework data provider.

Instead of relying on the inferred data type, you can use the typed accessor methods of the DataReader when you know the specific type of the value being returned. Typed accessor methods give you better performance by returning a value as a specific .NET Framework type, which eliminates the need for additional type conversion.

**Retrieving and Modifying Data in ADO.NET**

A primary function of any database application is connecting to a data source and retrieving the data that it contains. The .NET Framework data providers of ADO.NET serve as a bridge between an application and a data source, allowing you to execute commands as well as to retrieve data by using a **DataReader** or a **DataAdapter**. A key function of any database application is the ability to update the data that is stored in the database. In ADO.NET, updating data involves using the **DataAdapter** and [DataSet](http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx), and **Command** objects; and it may also involve using transactions.