.NET Framework 4

**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.

http://i.msdn.microsoft.com/Global/Images/clear.gif 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**

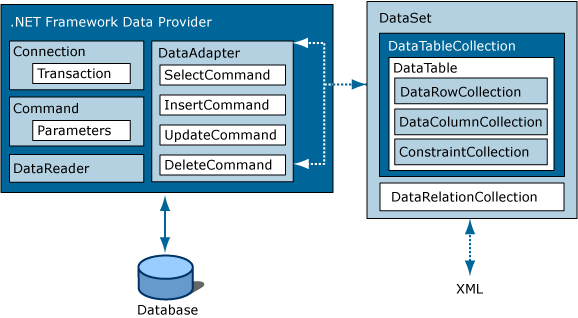
The .NET Framework Data Providers are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data. The **Connection** object provides connectivity to a data source. The **Command** object enables access to database commands to return data, modify 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**.

http://i.msdn.microsoft.com/Global/Images/clear.gif 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).

http://i.msdn.microsoft.com/Global/Images/clear.gif 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).

http://i.msdn.microsoft.com/Global/Images/clear.gif 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).

http://i.msdn.microsoft.com/Global/Images/clear.gif WCF Data Services

WCF Data Services 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 [WCF Data Services](http://msdn.microsoft.com/en-us/library/cc668792.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif 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 Technology Options and Guidelines**

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.

**Object Services**

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**

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).

**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 to SQL

LINQ to SQL is an object relational mapping (OR/M) implementation that allows you to model a SQL Server database by using .NET Framework classes. LINQ to SQL allows you to query your database by using LINQ, as well as update, insert and delete data from it. LINQ to SQL supports transactions, views, and stored procedures, providing an easy way to integrate data validation and business logic rules into your data model. You can use the Object Relational Designer (O/R Designer) to model the entity classes and associations that are based on objects in a database. For more information, see [Object Relational Designer (O/R Designer)](http://msdn.microsoft.com/en-us/library/bb384429.aspx). LINQ to SQL is supported for versions of SQL Server starting with SQL Server 2000.

 WCF Data Services

WCF Data Services deploys 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 [WCF Data Services](http://msdn.microsoft.com/en-us/library/cc668792.aspx).