In industry, most applications can’t be built without having interaction with a database. Databases serve the purpose of data storage so the data can be retrieved later via either a SQL query or a database application. Almost every software application running interacts with either one or multiple databases. Therefore, the front end needs a mechanism to connect with databases, and ADO.NET serves that purpose. Most of the .NET applications that require database functionality are dependent on ADO.NET. In this chapter, we’ll cover the following:

- Understanding ADO.NET
- The motivation behind ADO.NET
- Moving from ADO to ADO.NET
- Understanding ADO.NET architecture
- Understanding the SQL Server data provider
- Understanding the OLE DB data provider
- Understanding the ODBC data provider
- Data providers as APIs

Understanding ADO.NET

Before .NET, developers used data access technologies such as ODBC, OLE DB, and ActiveX Data Object (ADO). With the introduction of .NET, Microsoft created a new way to work with data, called ADO.NET.

ADO.NET is a set of classes exposing data access services to .NET programmers, providing a rich set of components for creating distributed, data-sharing applications. ADO.NET is an integral part of the .NET Framework and provides access to relational, XML, and application data. ADO.NET classes are found in System.Data.d1l.

This technology supports a variety of development needs, including the creation of front-end database clients and middle-tier business objects used by applications, tools, languages, and Internet browsers. Hence, ADO.NET helps connect the UI, or presentation layer, of your application with the data source or database.
The Motivation Behind ADO.NET

With the evolution of application development, applications have become loosely coupled, an architecture where components are easier to maintain and reuse without relying on implementation details of other components. More and more of today’s applications use XML to encode data to be passed over network connections, and that is how different applications running on different platforms can interoperate.

ADO.NET was designed to support disconnected data architecture, tight integration with XML, common data representation with the ability to combine data from multiple and varied data sources, and optimized facilities for interacting with a database, all native to the .NET Framework.

During the development of ADO.NET, Microsoft wanted to include the following features:

- **Leverage for the current ADO knowledge**: ADO.NET’s design addresses many of the requirements of today’s application development model. At the same time, the programming model stays as similar as possible to ADO, so current ADO developers do not have to start from scratch. ADO.NET is an intrinsic part of the .NET Framework yet is familiar to ADO programmers.

  ADO.NET also coexists with ADO. Although most new .NET-based applications will be written using ADO.NET, ADO remains available to the .NET programmer through .NET COM interoperability services.

- **Support for the n-tier programming model**: The concept of working with a disconnected record set has become a focal point in the programming model. ADO.NET provides premium-class support for the disconnected, n-tier programming environment. ADO.NET’s solution for building n-tier database applications is the DataSet.

- **Integration of XML support**: XML and data access are closely tied. XML is about encoding data, and data access is increasingly becoming about XML. The .NET Framework not only supports web standards but also is built entirely on top of them.

XML support is built into ADO.NET at a fundamental level. The XML classes in the .NET Framework and ADO.NET are part of the same architecture; they integrate at many different levels. You therefore no longer have to choose between the data access set of services and their XML counterparts; the ability to cross over from one to the other is inherent in the design of both.

Moving from ADO to ADO.NET

ADO is a collection of ActiveX objects that are designed to work in a constantly connected environment. It was built on top of OLE DB (which we’ll look at in the “Understanding the OLE DB Data Provider” section). OLE DB provides access to non-SQL data as well as SQL databases, and ADO provides an interface designed to make it easier to work with OLE DB providers.

However, accessing data with ADO (and OLE DB under the hood) means you have to go through several layers of connectivity before you reach the data source. Just as OLE DB is there to connect to a large number of data sources, an older data access technology, Open Database Connectivity (ODBC), is still there to connect to even older data sources such as dBase and Paradox. To access ODBC data sources using ADO, you use an OLE DB provider for ODBC (since ADO works directly only with OLE DB), thus adding more layers to an already multilayered model.