Borland Data Providers for Microsoft .NET

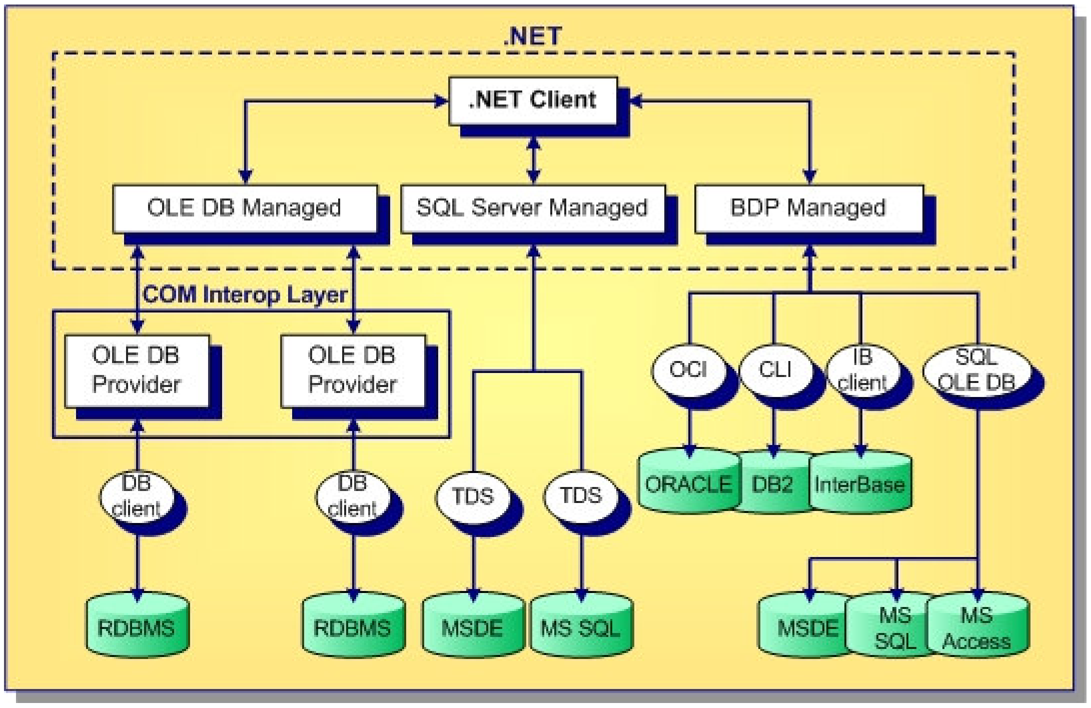
In addition to supporting the providers included in the .NET Framework, Delphi 2005 includes Borland Data Providers for Microsoft .NET (BDP.NET). BDP.NET is an implementation of the .NET Provider and connects to a number of popular databases.

This topic includes:

- Data Provider Architecture
- BDP.NET Advantages
- BDP.NET and ADO.NET Components
- Supported BDP.NET Providers
- BDP.NET Data Types
- BDP.NET Interfaces

Data Provider Architecture

Delphi 2005 supports the .NET Framework providers and the BDP.NET providers.



BDP.NET provides a high performance architecture for accessing data sources without a COM Interop layer.

The architecture exposes a set of interfaces for third-party integration. You can implement these interfaces for your own database to provide designtime, tools, and runtime data access integration into the Borland IDE. BDP.NET-managed components communicate with these interfaces to accomplish all basic data access functionality. These interfaces were implemented to wrap database-specific native client libraries by way of Platform Invoke (P/Invoke) services. Depending on the availability of managed database clients, you can implement a fully-managed provider underneath BDP.NET.

The database-specific implementation is wrapped into an assembly and the full name of the assembly is passed to the BdpConnection component as part of the connection string. Depending on the Assembly entry in the ConnectionString property, BDP.NET dynamically loads the database-specific provider and consumes the implementation for ISQLConnection , ISQLCommand , and ISQLCursor . This allows you to switch applications from one database to another just by changing the ConnectionString property to point to a different provider.

BDP.NET Advantages

BDP.NET provides a number of advantages:

- Unified programming model applicable to multiple database platforms
- High performance data-access architecture
- Open architecture, which supports additional databases easily
- Portable code to write once and connect to any supported databases
- Consistent data type mapping across databases where applicable
- Logical data types mapped to .NET native types
- No need for a COM Interop layer, unlike OLE DB
- Lets you view live data as you design your application
- Extends ADO.NET to provide interfaces for metadata services, schema creation, and data migration
- Rich set of component designers and tools to speed database application development

Delphi 2005 extends .NET support to additional database platforms, providing a consistent connection architecture and data type mapping.

BDP.NET and ADO.NET Components

The DataSet is an in-memory representation of one or more DataTables. Each DataTable in a DataSet consists of DataColumns and DataRows. The DataSet is generated as a result of an SQL query that you supply to the provider. You can navigate the DataSet like you would any standard relational table. BDP.NET providers encapsulate implementation details for each database type, yet allow you to customize your SQL statements and manage the result sets with complete flexibility.

BDP.NET includes several designtime components that you can place onto a Windows Form or Web Form. A set of designers are also provided to help you build your data connections, DataSets, relations, and other elements.

The primary components that are most useful, particularly if you decide to implement your own database-specific provider, are:

- BdpConnection —establishes a database connection
- BdpCommand —includes a set of methods and properties for SQL and stored procedure execution
- BdpDataReader —retrieves data
- BdpParameter —supports runtime parameter binding
- BdpTransaction —supports transaction control
- BdpDataAdapter —provides and resolves data
- BdpCopyTable —migrates table structures, primary keys, and data
- ISQLMetaData —retrieves metadata
- ISQLSchemaCreate —includes methods for creating, dropping, and altering database objects

For more information, click on the link for each component, or search for the components in the API reference documentation in this Help.

Supported BDP.NET Providers

BDP.NET includes providers for a number of industry-standard databases. These are shown in the following table, along with their corresponding namespaces.

Database	Namespace

InterBase	`Borland.Data.Interbase`
Oracle	`Borland.Data.Oracle`
IBM DB2	`Borland.Data.Db2`
Microsoft SQL Server	`Borland.Data.Mssql`
Microsoft Access	`Borland.Data.Msacc`
Sybase	`Borland.Data.Sybase`

The BDP.NET components, metadata access, and designers are defined under the following namespaces:

- `Borland.Data.Provider`
- `Borland.Data.Common`
- `Borland.Data.Schema`
- `Borland.Data.Design`

BDP.NET Data Types

BDP.NET maps SQL data types to .NET Framework data types, eliminating the need for you to learn a database-specific type system. Every attempt has been made to implement consistent type mappings across database types, allowing you to write one set of source that you can run against multiple databases. You can achieve a similar effect with the .NET Framework data providers by communicating with their interfaces directly and by using untyped ancestors. However, once you use strongly typed accessors, your application becomes less portable. BDP.NET does not support any database-specific typed accessors. For more information, see the BDP.NET Data Types topic.

BDP.NET Interfaces

You can extend BDP.NET to support other DBMSs by implementing a subset of the .NET Provider interface. BDP.NET generalizes much of the functionality required to implement data providers. While the .NET Framework gives you the capabilities to create individual data providers for each data source, Borland has simplified the task by offering a generalized set of capabilities. Instead of building separate providers, along with corresponding DataAdapters, DataReaders, Connection objects, and other required objects, you can implement a set of BDP.NET interfaces to build your own data source plug-ins to the Borland Data Provider.

Building plug-ins is a much easier task than building a completely new data provider. You build an assembly that contains the namespace for your provider, as well as classes that encapsulate provider-specific functionality. Much of the functionality you need to connect to, execute commands against, and retrieve data from your data source has already been defined in the Borland Data Provider interfaces.

Related Information

ADO.NET Overview
BDP.NET Component Designers
BDP.NET Data Types