

Failover Support

DataDirect Connect[®] for ADO.NET Data Providers

In today's business environment, you need to ensure that the data on which your critical .NET applications depend is always available. This document describes why it is important that your ADO.NET data provider includes connection failover support.

The DataDirect Connect *for* ADO.NET 3.0 data providers support the following useful features that help your data remain available:

- Connection failover, which allows an application to connect to an alternate database server if the primary database server is unavailable.
- Connection retry, which defines the number of times that the data provider attempts to connect to the primary and alternate database servers after the first unsuccessful connection attempt.
- Client load balancing, which changes the order in which the servers are accessed. You can use client load balancing with connection failover to help distribute new connections in your environment so that no one server is overwhelmed with connection requests.

Of course, for connection failover, client load balancing, and connection retry to work, an underlying replication mechanism must be used to ensure data consistency.

NOTE: Code examples in this document use the ADO.NET 2.0 Common Programming Model and MetaData capabilities introduced in the Microsoft .NET 2.0 Framework. If you are using the .NET Framework 1.x or DataDirect Connect *for* .NET 2.2 data providers, refer to [Failover Support: DataDirect Connect for .NET Data Providers](#).

Connection Failover

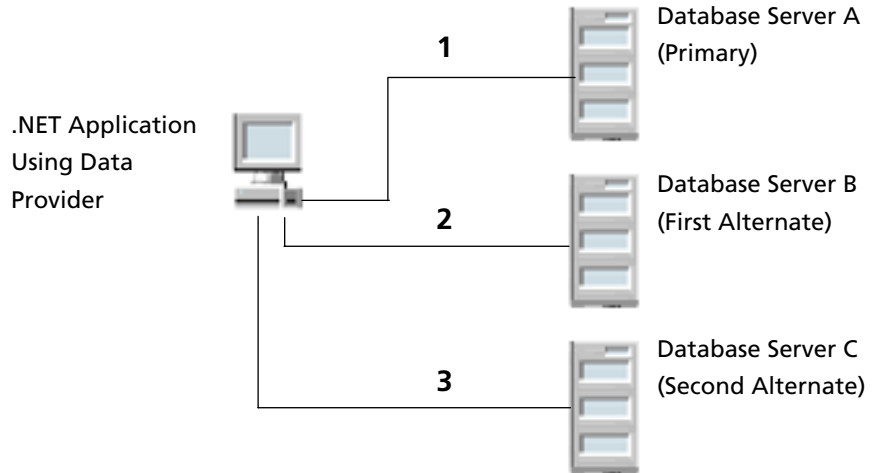
Connection failover allows an application to connect to an alternate, or backup, database server if the primary database server is unavailable, for example, because of a hardware failure or traffic overload. Connection failover ensures an open route to your data, even when server downtime occurs.

You can customize the DataDirect Connect *for* ADO.NET data providers for connection failover by configuring a list of alternate database servers that are tried if the primary server is not accepting connections. Connection attempts continue until a connection is successfully established or until all the alternate database servers have been tried the specified number of times.

For example, suppose you have an environment with multiple database servers, as shown in Figure 1: Connection Failover. Database Server A is

designated as the primary database server, Database Server B is the first alternate server, and Database Server C is the second alternate server.

Figure 1: Connection Failover



When the application requests a connection from the data provider, the data provider attempts to connect to the primary database server, Database Server A (1). If connection failover is enabled and Database Server A fails to accept the connection, the data provider attempts to connect to Database Server B (2). If that connection attempt also fails, the data provider attempts to connect to Database Server C (3).

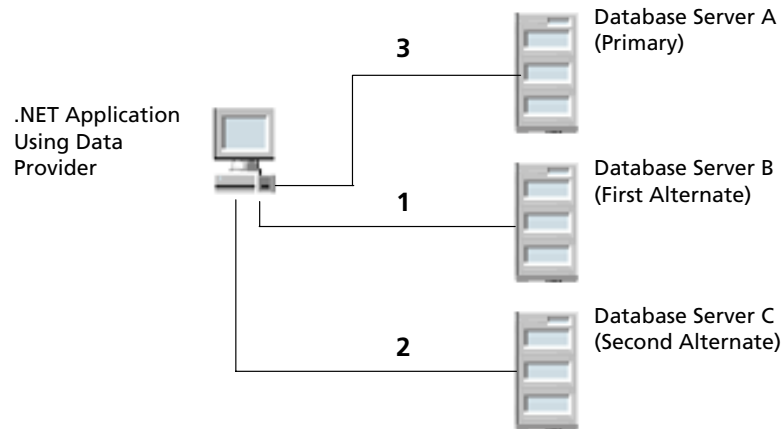
In the scenario in Figure 1, it is probable that at least one connection attempt would succeed, but if no connection attempt succeeds, the data provider can retry the primary database server and each alternate database server for a specified number of attempts. You can specify the number of attempts that are made through the connection retry feature. You can also specify the number of seconds of delay, if any, between attempts through the connection delay feature. See [“Connection Retry”](#) on page 4 for more information about connection retry and delay.

You can use client load balancing with connection failover to help distribute new connections in your environment so that no one server is overwhelmed with connection requests. Client load balancing changes the order in which the servers are accessed. For example, let us look again at the scenario shown in Figure 1. Without client load balancing enabled, connection attempts may be front-loaded, meaning that most connection attempts would try Database Server A first, then Database Server B, and so on until a connection attempt is successful. This creates a situation where Database Server A and Database Server B can become overloaded with connection requests.

When both connection failover and client load balancing are enabled as shown in Figure 2, the first connection attempt establishes the random order in which primary and alternate databases are tried. Subsequent connection

attempts use the same sequence. In this example, Database Server B is tried first (1). If that connection attempt is unsuccessful, Database Server C is tried (2), and then Database Server A (3). This makes it less likely that any one server will be so overwhelmed with connection requests that it may start refusing connections.

Figure 2: Client Load Balancing



See ["Client Load Balancing"](#) on page 5 for more information about client load balancing.

A DataDirect Connect *for* ADO.NET data provider fails over to the next alternate database server only if a successful connection cannot be established with the current alternate server. If the data provider successfully establishes communication with a database server and the connection request is rejected by the database server because, for example, the login information is invalid, then the data provider generates an error and does not try to connect to the next database server in the list. It is assumed that each alternate server is a mirror of the primary and that all authentication parameters and other related information are the same.

To configure connection failover, you specify a list of alternate database servers that are tried at connection time if the primary server is not accepting connections. To do this, you use the Alternate Servers connection string option in a connection string. Connection attempts continue until a connection is successfully established or until all the database servers in the list have been tried once (the default).

The following connection string configures the Oracle data provider to use connection failover:

```
DbProviderFactory Factory = DbProviderFactories.GetFactory("DDTek.Oracle");
DbConnection Conn1 = Factory.CreateConnection();
Conn1.ConnectionString =
    "Host=Accounting;Port=1521;User ID=scott;Password=tiger;" +
    "Service Name=ORCL;Min Pool Size=50";
    "Alternate Servers="Host=server2; Port=1521;Service Name=TEST," +
    "Host=255.210.11.25;Port=1600;Service Name = TEST2";" +
    "Load Balancing=true";
Conn1.Open();
```

Specifically, this connection string configures the data provider to use two alternate servers as connection failover servers.

Connection Retry

Connection retry defines the number of times that the data provider attempts to connect to the primary, and, if configured, alternate database servers after the first unsuccessful connection attempt. Connection retry can be an important strategy for system recovery. For example, suppose you have a power failure scenario in which both the client and the server fail. When the power is restored and all computers are restarted, the client may be ready to attempt a connection before the server has completed its startup routines. If connection retry is enabled, the client application can continue to retry the connection until a connection is successfully accepted by the server.

Connection retry can be used in environments that only have one server or can be used as a complementary feature in connection failover scenarios in environments with multiple servers. For example, the connection string in the following code fragment configures the Oracle data provider to use connection failover in conjunction with connection retry and connection retry delay:

```
DbProviderFactory Factory = DbProviderFactories.GetFactory("DDTek.Oracle");
DbConnection Conn1 = Factory.CreateConnection();
Conn1.ConnectionString =
    "Host=server1;Port=1521;Service Name=ORCL; Min Pool Size=50"; " +
    "Alternate Servers="Host=server2;Port=1521;Service Name=TEST," " +
    "Host=255.210.11.25;Port=1600;Service Name=TEST2";" " +
    "Load Balancing=true; Connection Retry Count =10;
    "Connection Retry Delay=10;";
Conn1.Open();
```

Specifically, the connection string configures the data provider to use two alternate servers as connection failover servers, to attempt to connect four additional times if the initial attempt fails, and to wait 5 seconds between attempts. When used with connection failover, the Connection Timeout connection string option applies to each connection attempt; in this example, the connection attempt lasts for 60 seconds.

You may want to enable connection retry attempts for the primary server only and not for alternate servers. The following connection string instructs the Oracle data provider to attempt to connect to the primary server up to 10 more times if the data provider was unable to establish a connection during the initial pass. The data provider waits 10 seconds before attempting to connect again. Each connection attempt lasts for 60 seconds.

```
DbProviderFactory Factory = DbProviderFactories.GetFactory("DDTek.Oracle");
DbConnection Conn1 = Factory.CreateConnection();
Conn1.ConnectionString =
    "Host=server1;Port=1521;Service Name=ORCL;User ID=test; " +
    "Password=secret;Connection Retry Count=10;Connection Retry Delay=10; " +
    "Connection Timeout=60;";
```

Client Load Balancing

Client load balancing works with connection failover to help distribute new connections in your environment so that no one server is overwhelmed with connection requests. When both connection failover and client load balancing are enabled, the first connection attempt establishes the random order in which primary and alternate databases are tried. Subsequent connection attempts use the same sequence.

The following connection string enables load balancing for the DataDirect Connect *for* ADO.NET Oracle data provider:

```
DbProviderFactory Factory = DbProviderFactories.GetFactory("DDTek.Oracle");
DbConnection Conn1 = Factory.CreateConnection();
Conn1.ConnectionString =
    "Host=server1;Port=1521;Service Name=ORCL;" +
    "Alternate Servers=Host=AcctOracleServer;Service Name=ORCL,"+
    "Host=255.210.11.25; Service Name=ORCL";" +
    "User ID=test; Password=secret;Connection Retry Count=10; " +
    "Load Balancing=true; Connection Retry Delay=10;Connection Timeout=60"
```

Summary

DataDirect Connect *for* ADO.NET data providers provide full support for the important features of connection failover, connection retry, and client load balancing to help make your business more flexible and agile in today's computing environment, where scalability and data availability is critical. With connection failover, an application can connect to a backup database server if the primary database server is unavailable, ensuring that the data on which your critical .NET applications depend is always available. Connection retry is another important strategy for system recovery after a power failure. If connection retry is enabled, the client application can continue to retry the connection until a connection is successfully accepted by the server. Client load balancing helps distribute new connections in your environment so that no one server is overwhelmed with connection requests after a power failure. All three features require an underlying replication mechanism to ensure data consistency.

For more information about failover support in DataDirect Connect *for* ADO.NET data providers, refer to the [DataDirect Connect for ADO.NET User's Guide and Reference](#).

FOR MORE INFORMATION

800-876-3101

Worldwide Sales

Belgium (French).....	0800 12 045
Belgium (Dutch).....	0800 12 046
France	0800 911 454
Germany	0800 181 78 76
Japan	0120.20.9613
Netherlands	0800 022 0524
United Kingdom	0800 169 19 07
United States	800 876 3101

Copyright © 2006 DataDirect Technologies Corp. All rights reserved. DataDirect Connect is a registered trademark of DataDirect Technologies Corp. in the United States and other countries. DataDirect XQuery is a trademark of DataDirect Technologies Corp. in the U.S. and other countries. Java and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Other company or product names mentioned herein may be trademarks or registered trademarks of their respective companies



DataDirect Technologies is the software industry's only comprehensive provider of software for connecting the world's most critical business applications to data and services, running on any platform, using proven and emerging standards. Developers worldwide depend on DataDirect® products to connect their applications to an unparalleled range of data sources using standards-based interfaces such as ODBC, JDBC™ and ADO.NET, XQuery and SOAP. More than 300 leading independent software vendors and thousands of enterprises rely on DataDirect Technologies to simplify and streamline data connectivity for distributed systems and to reduce the complexity of mainframe integration. DataDirect Technologies is an operating company of Progress Software Corporation (Nasdaq: PRGS). For more information, visit www.datadirect.com.