Oracle Data Provider For .NET 11*g*

An Oracle Technical White Paper
October 2007

ORACLE

Oracle Data Provider For .NET 11*g*

INTRODUCTION

One of the great benefits of using Oracle products is their support for multiple programming frameworks. By supporting .NET, Java/J2EE, PHP, and C/C++ applications, all developers can use Oracle's advanced database features, providing true flexibility for development organizations. Each of Oracle's data access drivers is designed to maximize performance of its respective framework and to enable access to the latest database features.

Within the .NET realm, Oracle offers a multitude of products to develop applications with the Oracle database, including Oracle Developer Tools for Visual Studio .NET, Oracle Database Extensions for .NET, Oracle Providers for ASP.NET, and Oracle Data Provider for .NET (ODP.NET).

ODP.NET is a native .NET data access provider for Oracle databases. It offers standard ADO.NET data access for .NET Framework 1.x, 2.0, and higher. For developers who have used another ADO.NET provider, basic ODP.NET data access requires very little new to learn. It can be used with existing constructs, such as the Microsoft Data Access Application Blocks (DAAB) for .NET. As such, ADO.NET developers can start using Oracle data sources very quickly.

While ODP.NET shares common basic elements with other ADO.NET providers, ODP.NET's unique value is its tight integration with the Oracle database. ODP.NET exposes many of the database's unique capabilities, such as Real Application Clusters (RAC) tuning, advanced security, and complex data types, to .NET developers. These features allow .NET middle-tiers to take advantage of the Oracle database's unique capabilities.

This white paper focuses on ODP.NET data access and its unique capabilities, discussing .NET development features that are part of Oracle Database 11*g* and prior releases. Many ODP.NET 11*g* features are enabled with older Oracle database server releases, such as the Oracle Database 10*g* and Oracle Database 9*i*. This white paper does not cover ODP.NET features released after Oracle Database 11*g* on Windows, such as ODP.NET 11.1.0.6.20.

ORACLE DATA PROVIDER FOR .NET

The Oracle Data Provider for .NET enables fast data access performance to Oracle databases. It supports the latest database functionality available with Oracle Database 11*g*.

As with other .NET data providers, any .NET application, including C# .NET, Visual Basic .NET, and ASP.NET, can call ODP.NET. While most commonly employed in middle-tiers, ODP.NET can also be used within the database server itself via .NET stored procedures.

ODP.NET offers the best performance and greatest access to Oracle database features than any other .NET data provider. ODP.NET was designed specifically to maximize Oracle database's capabilities.

ODP.NET 11*g* introduces new performance features available in Oracle Database 11*g*, as well as enhance existing functionality in the Oracle database. As such, all developers will benefit using the latest ODP.NET version whether they are deploying new database applications or enhancing existing applications. Some of these unique Oracle features include data access performance tuning; database change notification; RAC connection pooling; XML support; native Oracle data type support; and many other features, which will be described further in this paper.

Performance

ODP.NET includes numerous performance tuning features to optimize data retrieval and data changes. Some of these tuning options include connection pooling; statement caching; using LOB data types; and employing PL/SQL associative arrays. 64-bit ODP.NET is supported on Windows x64 and Windows Itanium.

One of ODP.NET's key differentiators over other providers is its out-of-the-box performance and numerous tuning options. Under the covers, numerous optimizations automatically ensure fast .NET access to Oracle data sources without requiring any specific performance coding. In addition, ODP.NET has many tunable parameters for specific data retrieval and data update scenarios. Many of these optimizations were developed for retrieving and manipulating historically large data types, such as LOBs and REF Cursors.

Connection Pooling and Statement Caching

One of the most widely used performance optimizations is connection pooling, which is critical for applications with large numbers of users that connect and disconnect from the database. ODP.NET creates a pool of connections with tunable settings that include connection lifetime and timeout, minimum and maximum pool sizes, and the numbers of connections to increment or decrement from the pool at a time. These parameters give developers greater control over how their application handles large user populations and the changes in those populations over time. This ultimately leads to better application response time and quality of service for end users.

If a particular query or PL/SQL statement is executed multiple times, ODP.NET can use statement caching to speed statement execution. By caching the server cursor created during the initial statement execution, statement caching eliminates the need to re-parse each statement before subsequent executions. Each subsequent statement execution reuses the saved parsed information, and then executes the statement. The result set data itself is not cached, only the parsed statement information. ODP.NET will still retrieve the latest data from the

database server. Statement caching just allows these queries to be executed more quickly.

When employing statement caching, SQL or PL/SQL statements should use parameters rather than literal values. Doing so takes full advantage of statement caching since parsed information from parameterized statements can be reused even if the parameter values change in subsequent executions. If literal values were used instead and those literal values changed, the parsed information could not be reused and the database would need to parse the statement anew.

By default, ODP.NET will cache the last ten executed statements. The number of statements to cache and which statements to cache can be configured at the application level or machine level.

Controlling Data Fetch Size

To tune data retrieval performance, ODP.NET can specify a set amount of data to return for each database round trip. Many times, a developer may not need to retrieve the data queried all at once. The end user may be consuming portions of the data over a period of time.

The query's data fetches can be spaced in distinct chunks defined by the developer through two ODP.NET OracleCommand properties: FetchSize and RowSize. FetchSize tells ODP.NET how much data to retrieve per database roundtrip. RowSize indicates how large each row of data is. RowSize is a read-only property that is set after a query is executed. If a developer wishes to fetch ten rows of data per database roundtrip, all that is required is to set FetchSize equal to ten multiplied by RowSize. The wonderful thing about RowSize is that its value can be determined at runtime. Consequently, if there is a schema change or a query change in the future, there is no need to modify the code to ensure ten rows of data are fetched per round trip.

Optimizing LOB Data

A similar fetch size tuning feature exists with LOB data types. These data types are used to store images and documents, which can sometimes be in the range of gigabytes. For LOB applications, performance is often a key issue due to LOB's potential data size and how LOB data is consumed. Sending gigabytes of data between the server and client can clog a network unless data retrieval is handled intelligently.

With ODP.NET, developers can specify how LOB data should be retrieved. Upon LOB query execution, developers can choose to fetch all the LOB data immediately or defer the LOB fetch until the user attempts to read the data. The end user may not need to read the data directly after the query execution. If developers choose to defer the LOB fetch, they can then specify how much data to retrieve for each LOB read call. If the end user only needs to read 10KB of data at a time, developers can retrieve just 10KB of data for each LOB read. This optimizes how networking resources between the server and client are used.

Moreover, ODP.NET developers can retrieve any portion of a LOB via random access. Perhaps the end user may need only the last 100MB of data from a 1GB LOB. Developers can tune LOB retrieval to only fetch the last 100MB without returning data from the first 900MB to the client. These tuning options provide .NET developers the flexibility to build better performing applications.

Because LOB data can often be large, by default, the LOB data fetch is deferred after a query is executed. When retrieving many sets of large LOBs, this behavior is optimal to prevent overloading the network with LOB data delivered to the client. However, for small LOBs, this behavior can be slow, causing more database round trips than is necessary.

To allow all the small LOB data to be fetched immediately, ODP.NET has an InitialLOBFetchSize property on the OracleCommand and OracleDataReader classes. If InitialLOBFetchSize is set to a value greater than zero, the initial LOB data from all the LOBs queried is fetched in one round trip up to the number of characters or bytes that is specified in this property. For instance, if InitialLOBFetchSize were set to 10 KB, then the first 10 KB of all the LOBs selected would be retrieved to the client in one database round trip. This can significantly speed up applications consuming lots of small LOBs.

Array Data

One of ODP.NET's unique features is the ability to pass arrays between the database and .NET Framework. Arrays make sharing large sets of data of the same data type between the database and client much easier. ODP.NET uses PL/SQL associative arrays in the database to pass data into and out of .NET arrays.

64-bit .NET Framework

With 64-bit .NET Framework, .NET developers have access to more scalable and high-performing hardware systems. They have a choice between AMD64 and Intel EM64T processors for Windows x64 and Itanium processors for Windows Itanium. 64-bit systems have the capability to address larger amounts of memory directly than 32-bit systems can. They include optimized hardware components for high performance computing. Beginning with the 10.2.0.3 release, ODP.NET supports both 64-bit .NET Frameworks with a native 64-bit data access driver for each platform. Developers can now deploy their ODP.NET mid-tier as a 64-bit application to take advantage of the more scalable hardware.

Performance – New for ODP.NET 11*g*

New Oracle Database 11*g* features are available that enhance ODP.NET performance. These features include a client result cache, faster LOB fetching, and faster performance with statement caching.

Oracle Database 11*g* introduces new performance optimizations, many of which .NET application developers can use without any changes to their existing client code. These new features include a client result cache, faster LOB retrievals, and enhanced statement caching.

Client Result Cache

With Oracle Database 11*g* server and client, ODP.NET applications can use the Oracle client result cache to improve response times of repeatedly executed queries.

This feature enables client-side caching of SQL query result sets in memory. The client result cache is completely transparent to ODP.NET applications, and its cache of result set data is automatically kept consistent with any session or database server side changes that would alter the results.

.NET applications calling the same query multiple times see improved performance since query results are retrieved locally. Local client processing is faster than making database round trips to re-execute a query and fetch results. If applications are frequently running the same queries, they will experience a significant performance improvement when their results are cached on the client, as well as a reduction in database server load.

On the database server, the client cache reduces the server CPU and network traffic load that would have been consumed for processing and returning the query results, thereby improving server scalability. ODP.NET statements from multiple sessions can match the same cached result set in the client process memory if they have similar schema, SQL text, bind values, and session settings. Otherwise, the query execution occurs on the server. This means that multiple ODP.NET users all have access to the same result cache, which minimizes cache redundancies and saves memory.

Because the client cache automatically stays consistent with database server data, developers do not need to write code to ensure the cache and server remain in synch. If a server change occurs that would invalidate the client-cached data, the Oracle client will automatically invalidate the cache and update it the next time the query is executed.

Faster LOB Fetching

ODP.NET 11*g* improves the performance of small-sized LOB retrieval by reducing the number of round-trips to the database required for pre-fetching the LOB data, length, and chunk size. This enhancement is available beginning with Oracle Database 11*g* for use with either traditional LOBs or SecureFiles. This enhancement is transparent to the developer. It can be used like any other LOB data type without any code changes to existing ODP.NET LOB code.

Enhanced Statement Caching

ODP.NET 11*g* enhances the existing caching statement caching infrastructure to now cache ODP.NET parameter contexts. This enhancement works with any currently supported Oracle database server version. .NET developers will see a performance improvement when executing statement-cached queries. This enhancement is transparent to developers, requiring no code changes.

Database Change Notification

A constant challenge for client side caches is keeping the data in synch with server data changes. Using database change notification, ODP.NET clients are alerted when data on the server is modified, even if there is no active connection back to the database. This allows the client to ensure its data cache stays synchronized with the database.

Database change notification enables client applications to receive notifications when DML or DDL changes are made to a database object of interest, even when the client no longer has a connection to the database server. .NET developers can now cache their data on the middle-tier without having to worry about the cached data becoming out of synch with the database. If a change happens to one of the cached data objects or rows of data, then ODP.NET will receive a notification from the database. This feature can be used in the .NET Framework 1.x, 2.0, and higher releases.

To use database change notification, the client application registers a query with the database. When a query has dependencies on underlying database objects and a change to a dependent object is committed, the database publishes a change notification to the client application. The notification only contains metadata about what data or objects changed; it does not contain the changed data. .NET developers can create a client event handler to reissue the registered query to obtain the changed data.

Database change notification is particularly useful for applications that use cached results. Traditionally, data caching by itself is effective at improving application scalability by allowing rapid access to data without making expensive roundtrips to the database. But this scalability comes with a tradeoff, as there is no longer a guarantee that the data remains consistent with the database server after the initial query. Thus, the cached client data runs the risk of becoming stale.

Database change notification solves the stale data cache problem. Although database change notification is similar to a trigger in that it responds to a particular event, a trigger takes action immediately, whereas a database notification is just an alert, not an action. It is up to the application to determine what action, if any, to undertake and when. The application can immediately refresh the stale objects, postpone the refresh, or ignore the notification. Each .NET application may want to respond differently to a particular database change. Moreover, as additional applications are added to the database, it is often easier modifying the client application's event handler than modifying a database trigger. Modifying the trigger may require re-testing how existing applications work with the new database trigger code, while modifying just the new .NET application better isolates the testing boundaries.

Web applications often cache a variety of data, not all of which needs to be updated in real time. For example, a weather forecast may only be updated periodically. End users don't need to query the database every time the web page is visited. Since many people will be requesting the same data, application performance and scalability are greatly enhanced by caching the results and retrieving the data from the cache. At some point, the weather forecast is updated and the cache must be refreshed. This may be done the instant the current weather forecast in the database server has changed.

To receive database change notifications requires the database administrator to grant the CHANGE NOTIFICATION privilege to the application user. After connecting to the database, .NET users can then register their specific queries of interest for change notification. The developer creates a .NET client-side event handler to direct what the application should do upon receiving a database change notification. Usually, the event handler will re-query the database server and refresh the cache.

The following ODP.NET classes are used when building change notification applications:

- OracleDependency – Creates a dependency between an application and an Oracle database. It enables the application to receive a notification of a data change (for example, UPDATE statement), schema change (for example, ALTER TABLE), or global event (for example, database shutdown). The OnChange event handler in this class provides the client logic for what to do after the notification is received.
- OracleNotificationEventArgs – Provides all the event details when a change notification occurs.
- OracleNotificationRequest – Specifies the characteristics of a notification request and its notification, such as the notification registration's timeout value.

Oracle's database change notification has a number of features not available on SQL Server. Oracle supports all types of joins, whereas SQL Server does not support queries containing either outer joins or self-joins. SQL Server does not support notifications for statements using views, whereas Oracle database change notification supports views with the exceptions of fixed views (for example, V$ tables) and materialized views. SQL Server notifications also require explicit column references, while Oracle Database notifications support both SELECT * and explicit column references.

SQL Server notification handlers are not persistent. When a SQL Server notification is published, its notification handler is removed from the database. If the notification handler is still needed, the application must register a new notification handler. Oracle database change notification provides the option to persist the registration even after repeated changes. This is possible by setting OracleNotificationRequest.IsNotifiedOnce to false.

Real Application Clusters (RAC)

ODP.NET has two connection pooling features for RAC databases: 1) ODP.NET connections are automatically load balanced across existing nodes based on real-time cluster metrics and 2) severed database connections are automatically removed from the connection pool.

RAC is a cluster database with a shared cache architecture that overcomes the limitations of traditional shared-nothing and federated database approaches. The RAC cluster database is hosted on multiple computing nodes permitting better scalability and higher availability than a single computer server by itself. Because neither special hardware nor .NET application changes are required, RAC can be built with commodity hardware as the back end for existing applications without requiring any coding changes.

ODP.NET has always supported data access for RAC transparently. To improve ODP.NET connection management based on real-time database workload information, Oracle introduced two connection pool properties for ODP.NET 10.2. The first feature, runtime connection load balancing, improves load balancing workload across RAC instances, especially after nodes are added or removed from a cluster. The second feature, fast connection failover, automatically removes severed RAC connections from the connection pool.

With runtime connection load balancing, how ODP.NET connections are allocated is based on the database's load balancing advisory and service goal during runtime. Load balancing distributes work across all of the available RAC database instances.

In general, connections are created infrequently and exist for a long duration. Work comes into the system with high frequency, uses these connections from the pool, and exists for a relatively short duration. The load balancing advisory directs which RAC instances ODP.NET should allocate incoming work to provide the optimal service quality. Load balancing minimizes the need to relocate the work later to a different instance and ensures existing jobs can complete quickly.

The metric by which this work is distributed across instances is determined by the service goal. The database administrator sets the service goal for either service time or throughput.

The service time metric is based on how quickly the database can complete tasks, essentially its response time. The load balancing advisory data is based on the elapsed time for work done in the service as well as available bandwidth to the service. Service time is most useful for database applications where work tasks complete at different rates, such as an internet shopping system.

The throughput metric, too, tracks how quickly database tasks are completed, allocating more work to nodes where tasks are completing the fastest. Unlike service time, the throughput metric is intended to gauge the efficiency of each instance. It measures the amount of processor resources available to take on additional tasks. When a new operation comes in, it can be directed to the node with the most processor time available. This metric is best used for relatively uniform tasks, such as executing homogenous batch processes.

By using these service goals, feedback is built into the system. Work is routed to provide the best service times globally, and routing responds gracefully to changing

system conditions. When a RAC node is added to or removed from the cluster, load balancing allows work to more quickly distribute itself evenly across all the nodes to account for the system change. End users will then experience less disruption or slowed service. In a steady state, the system approaches equilibrium with improved throughput across all of the RAC instances.

To use this feature in ODP.NET, the application must enable connection pooling and set the connection pool parameter, Load Balancing, to true.

The second configurable feature for RAC connection pooling, fast connection failover, allows ODP.NET to automatically free resources associated with severed connections that were caused by a downed RAC service, service member, or node. Without this feature, if a RAC node failed, the connection pool would retain connection resources that were no longer valid. End users may attempt to use these severed connections from the pool. Without a way to identify these connections, administrators would have to reset all the ODP.NET connections in the pool every time some part of the RAC cluster failed.

These severed ODP.NET connections are now automatically cleaned up without requiring any administrator intervention at runtime. To enable this ODP.NET feature, use connection pooling and set the connection pool parameter, HA Events, to true.

XML Features

ODP.NET provides support for Oracle XML DB and interoperating with System.XML. ODP.NET can work with schema and non-schema-based XML with support for native Oracle XML data types using OracleXMLType and OracleXMLStream.

Because XML is a popular language for data integration and web services, many .NET programmers incorporate it as an integral part of their applications. XML is a key part of both the Oracle database and .NET Framework. ODP.NET allows developers to exploit these two technologies together: Oracle XML DB and .NET System.XML services.

XML DB is Oracle's high-performance, native XML storage and retrieval technology available within the database server. It provides a unique ability to store and manage both structured and unstructured data under a standard W3C XML data model. XML DB provides complete transparency and interchangeability between the XML and SQL metaphors. ODP.NET exposes XML DB's functionality to .NET clients, allowing developers to share and make changes to XML between the database and .NET. This support extends to both schema and non-schema-based XML to provide flexibility for differing application requirements. Moreover, ODP.NET contains two native XML data types, OracleXMLType and OracleXMLStream, for easier client XML data management. These features allow easier XML management between Oracle XML DB and Microsoft's System.XML services.

System.XML is a set of interfaces for manipulating XML data sets from .NET data providers. ODP.NET interoperates with the System.XML programming interfaces, feeding data via the ODP.NET DataAdapter interfaces. One of the key distinctions between XML DB and System.XML is that the former provides XML services where the data resides, on the database server; the latter manipulates XML

on the client side. As such, ODP.NET provides greater choice for programmers to pick the XML technology that best fits their project requirements.

With ODP.NET, relational and object-relational data in the Oracle database can be accessed as XML and delivered into a Microsoft .NET environment. Client XML changes can be made and saved back to the server as XML or relational data. ODP.NET includes support for schema-based OracleXMLType, as well as non-schema-based XML.

ADO.NET 2.0

ODP.NET supports ADO.NET 2.0's features, including provider factories, connection string builder, schema discovery APIs, and DataAdapter batch updates.

Beginning with version 10.2.0.2, ODP.NET provides support for ADO.NET 2.0. ADO.NET 2.0 introduces a new level of abstraction for the Data Access Layer (DAL). Instead of using provider-specific classes that implement a generic interface, ADO.NET 2.0 offers the DbCommon classes, which inherit from the System.Data.Common namespace and allow developers to use factory classes. Using database provider factory classes is an easy way to create one set of generic data access code to any database.

The OracleClientFactory class supports the creation of all of the classes in System.Data.Common. Because these concrete classes inherit from the DbCommon abstract base classes, developers can write generic DAL code by using DbCommon base class object names. There are some areas of the DAL that remain data source-specific, including the connection string, SQL, and stored procedure calls.

A key addition to ADO.NET 2.0 is improved connection string management. The DbConnectionStringBuilder class has a dictionary that maps generic parameter names to provider specific parameter names. The OracleConnectionStringBuilder class inherits from and extends the generic DbConnectionStringBuilder class to expose Oracle-specific connection string properties. Programmers can use OracleConnectionStringBuilder dynamically to set connection string parameters at runtime and/or obtain connection string parameters from the app.config file. This feature enables easier management of connection strings, especially those with a large number of attributes.

The ADO.NET 2.0 schema discovery APIs provide a generic way to retrieve metadata from any data source. Developers can obtain Oracle metadata by using an OracleConnection.GetSchema method call. There are five types of common metadata that can be exposed: MetaDataCollections, Restrictions, DataSourceInformation, DataTypes, and ReservedWords. In addition, there is additional ODP.NET-specific data source information that can be retrieved. With schema discovery, developers have a simple way to retrieve metadata from any Oracle data source.

When working with large .NET DataSets, developers will want to minimize the number of database round trips required to perform updates. By using the OracleDataAdapter.UpdateBatchSize property, programmers can specify the number of rows to update per roundtrip. To update all the modified rows in one

roundtrip, UpdateBatchSize can be set to zero. This batch processing support allows DataSet updates to be performed much more quickly.

Native Oracle Types

ODP.NET supports native Oracle database types, such as REF Cursors, within the .NET environment. These native data types can provide better performance and more flexibility for data retrieval.

Microsoft has introduced a set of unified data types among the different .NET programming languages. With ODP.NET, .NET programmers have access to .NET data types as well as Oracle data types. Oracle data types can be fully manipulated within a .NET application and interoperate with .NET data types. Oracle native types provide advanced functionality to store and manipulate data structures from the database, such as XML, result sets, images, text, or Microsoft Office documents. Even with scalar types, such as OracleDecimal, the equivalent to the .NET decimal type, Oracle types provide additional functionality. In the example of OracleDecimal, this data type provides a higher level of precision of 38 digits than the .NET decimal with a 28 digit precision.

ODP.NET supports the gamut of advanced Oracle types within the .NET environment, including REF Cursors, XMLType, LOBs (CLOBs, BLOBs, NCLOBs), BFILEs, LONGs, RAWs, LONG RAWs, and N-data types. One of the limitations of using third-party .NET data providers is that users may be limited in Oracle data type functionality. For example, in ODP.NET, multiple result sets returned as REF Cursor output parameters from a stored procedure can be accessed in an arbitrary way. One can read the results of the second REF Cursor without retrieving the results of the first. With other .NET providers, data may have to be accessed in a linear manner where the first result set's data must be retrieved prior to accessing the second's. This would negatively affect performance.

Beginning with ADO.NET 2.0, Oracle data types can be stored natively within the .NET DataSet. Previously, only .NET data types could be used in DataSet. By setting OracleDataAdapter.ReturnProviderSpecificTypes to true, the DataSet will be populated with ODP.NET-specific data types when OracleDataApdater.Fill is called. This feature can improve performance because it saves .NET from conducting data type conversions.

Other Major Features

Other major ODP.NET features include being able to use any type of PL/SQL; local and distributed transactions; and internationalized applications with Unicode.

ODP.NET exposes many other Oracle database features, including PL/SQL, transactions, and Unicode support.

ODP.NET users can fully execute PL/SQL stored procedures and functions in the database. PL/SQL can be packaged or non-packaged, or even exist as anonymous PL/SQL within .NET. In ODP.NET, anonymous PL/SQL is generally employed to batch a set of SQL statements and execute the statements in one database round trip. Batching statements is a useful performance optimization technique.

ODP.NET can participate in transactional applications with the Oracle database as the resource manager. ODP.NET employs the Microsoft Distributed Transaction Coordinator (DTC) to oversee a transaction in a Windows environment. The

Oracle Services for Microsoft Transaction Server (OraMTS) act as a proxy among ODP.NET, DTC, and the Oracle database in order to coordinate these transactions. OraMTS provides a robust architecture for ODP.NET programmers to have their transactional applications maintain high availability and high scalability. In .NET Framework 2.0 and higher, ODP.NET 10.2.0.3 supports both local and distributed transactions via the System.Transactions namespace.

ODP.NET has full Unicode support so that .NET users can globalize their applications easily in multiple written languages. This globalization support allows developers to create one set of code for multiple culture/language settings. ODP.NET globalization extracts the client computer's national language setting to display information in a locale-specific format. For example, a browser set to Japanese will have currency displayed in yen. Without additional coding, that same application can be deployed in Germany to show currency in euros. This makes developing applications for multiple locales easier and faster since no additional coding is necessary.

CONCLUSION

In Oracle Database 11*g*, ODP.NET offers new performance capabilities for Oracle database developers that build upon its existing performance, scalability, ease of use, and security features. As new Oracle database features and new .NET Framework features are introduced, ODP.NET continues to provide support for these new features.

For more information about ODP.NET and Oracle on .NET in general, visit

OTN ODP.NET Product Center:
http://www.oracle.com/technology/tech/windows/odpnet/

OTN .NET Developer Center: http://otn.oracle.com/dotnet

Oracle Data Provider For .NET 11*g*
October 2007
Author: Alex Keh

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com