An Oracle White Paper
September 2009

What's New in Oracle Data Provider for .NET 11.1.0.7.20

ORACLE

Introduction

Oracle Data Provider for .NET (ODP.NET) is a native ADO.NET data access driver for Oracle databases. ODP.NET 11.1.0.7.20 introduces new features to improve ease of development, application scalability, performance, security, and manageability. This release exposes unique Oracle database functionality, which .NET developers can now use more easily, and improves Oracle integration with the .NET Framework. Features include:

- **Oracle Streams Advanced Queuing (AQ)** – AQ provides database-integrated message queuing. ODP.NET can programmatically access AQ operations, such as message enqueue, dequeue, listen, and notification for building robust messaging applications. Oracle Developer Tools for Visual Studio can administer and manage AQ resources in the database.

- **Promotable Transactions** – Promotable transactions optimize resource usage by deferring distributed transaction use until absolutely required. ODP.NET transactions begin as local transactions with the first database connection. When a second database connection joins, ODP.NET promotes the local transaction to a distributed transaction.

- **Performance - Application Self-Tuning** – After sampling query behavior at run-time, ODP.NET dynamically tunes the statement cache size. This enhancement improves performance, reduces network usage, and saves on client and server memory usage.

- **Performance - Faster Data Retrieval and Optimized Memory Usage** – *OracleDataReader* data retrieval and *OracleDataAdapter.Fill* performance have been improved. In addition, ODP.NET now reuses the fetch array buffer, which stores data from executed statements. This feature reduces client memory usage since ODP.NET requires fewer fetch array buffers for typical applications.

- **Code Access Security** – ODP.NET now includes *OraclePermission* and *OraclePermissionAttribute* classes to help developers enforce imperative and declarative Code Access Security, respectively.

- **High Availability Event Notification and Callback** – ODP.NET receives notifications when a database, service, service member, host, or instance fails or becomes available. .NET developers can register an ODP.NET callback to notify the application when one of these events occurs and execute an event handler.

- **Programmatic Database Startup and Shutdown** – ODP.NET users with database administrator privileges can programmatically startup or shutdown a database instance.

ODP.NET 11.1.0.7.20 can connect to Oracle Database 9*i* Release 2 (9.2) or higher. The database server can be on any platform, such as Windows, Linux, or UNIX. ODP.NET supports .NET Framework 2.0 and higher. Developers can download ODP.NET for free from Oracle Technology Network (OTN).

http://www.oracle.com/technology/tech/windows/odpnet/

Oracle Streams Advanced Queuing

Oracle Streams Advanced Queuing (AQ) provides database-integrated message queuing. AQ is often used for message management and asynchronous communication among applications. For example, AQ can route a typical purchase order to a validation application, and then later to a sales recording application. AQ can then send the order to the shipping department for order fulfillment. Upon fulfillment, the order information can be routed to the billing department for payment processing. AQ and .NET can orchestrate this entire business process.

Queues act as message repositories between sender and recipient(s). AQ leverages the Oracle database to persist messages, to propagate them among queues, and to transmit messages using Oracle Net Services and HTTP(S).

Because AQ uses the database infrastructure, all operational benefits of persistence, high availability, scalability, scheduling, and reliability apply to queue data. AQ supports standard database features, such as recovery, restart, and security.

ODP.NET Oracle Streams AQ Functionality

ODP.NET 11.1.0.7.20 introduces an AQ application programmatic interface (API) that can access AQ's operational features, including enqueue messages, dequeue messages, listen for queue messages, and message notification. The ODP.NET AQ API is easy to use for building

.NET messaging applications with Oracle database. Oracle Developer Tools for Visual Studio can administer and manage AQ resources so that developers never have to leave Visual Studio when working with AQ.

ODP.NET queues are represented by *OracleAQQueue* objects. A user queue handles normal message processing. An exception queue handles unprocessed or unretrieved messages. A single message or an array of messages can be enqueued.

The AQ message itself is represented by an *OracleAQMessage* object. This object consists of control information and the message data. The message can be one of several data types: XML, Raw (OracleBinary or byte array), or user-defined type.

Developers have numerous customizable message options. They can specify message recipients, overriding the queue subscriber list. Messages can be set with different priority levels for priority based dequeuing. Messages can remain on the queue for a developer-defined set time before expiring, whereby the message is moved to the exception queue. These are some of the many choices available for managing individual queue messages.

.NET developers set enqueue options via the *OracleAQEnqueueOptions* class. For example, a message can be enqueued persistently on disk or buffered in memory. Disk storage is a high availability medium; however, memory storage is better performing.

Similarly, there are dequeue options available through the *OracleAQDequeueOptions* class. With this object, developers can search and wait for messages with matching criteria.

Subscribers can listen on an *OracleAQQueue* object for relevant messages. Alternatively, developers can employ asynchronous notification with the *OracleAQQueue.MessageAvailable* event. This event provides notification if there is a queue message available for *OracleAQQueue.NotificationConsumers*, which consists of an array of consumers.

The table below illustrates some of the key ODP.NET AQ APIs.

TABLE 1. ODP.NET AQ FUNCTIONALITY AND IMPLEMENTATION EXAMPLES

AQ FUNCTIONALITY	ODP.NET EXAMPLE
Create Message	Create an *OracleAQMessage* object
Enqueue single message	Specify an *OracleAQMessage* message, an *OracleAQQueue* queue, and the *OracleAQQueue enqueue* options. Finally, call *OracleAQQueue.Enqueue*.
Enqueue multiple messages	Specify an *OracleAQMessage* array of messages in *OracleAQQueue.EnqueueArray*.
Dequeue single message	Specify the *OracleAQQueue* dequeue options and then call *OracleAQQueue.Dequeue*.
Dequeue multiple messages	Call *OracleAQQueue.DequeueArray*.
Listen for messages on Queue(s)	Call *OracleAQQueue.Listen*. To listen on multiple queues, use static *OracleAQQueue.Listen* method.
Message Notification	Use *OracleAQQueue.MessageAvailable* event and the *OracleAQQueue.NotificationConsumers* property

Promotable Transactions

Distributed transactions require orchestration among application, transaction coordinator, and multiple resource managers. Local transactions only require an application and a single resource manager. When compared to distributed transactions, local transactions have much less overhead and are preferred for greater application performance and scalability.

At design time, it may not be clear when a distributed or local transaction will be required. Depending on run time circumstances, the same transaction code may sometimes require just one database connection (i.e. local) and sometimes require more than one database connection (i.e. distributed). Normally, developers would have to design for a distributed transaction in all cases, even if a local transaction could have been used some of the time.

ODP.NET promotable transactions allow all transactions to start local. Transactions are promoted to distributed transactions only when more than one resource manager joins. Promotable transactions provide better run time optimization of system resources.

This new feature requires that the first connection be to Oracle Database 11g Release 1 or higher. Subsequent connections may connect to any other Oracle resource or version, or even another vendor's database.

No ODP.NET code change is required to enable promotable transactions. The *Promotable Transaction* setting must be set to "promotable", which is its default value. Oracle Services for

Microsoft Transaction Server 11.1.0.7.20 or higher and .NET Framework 2.0 or higher must be used on the mid-tier.

Performance

ODP.NET introduces performance enhancements that dramatically speed data access and reduce resource usage. No code change is necessary to employ these enhancements, meaning developers will see faster and more efficient ODP.NET data access just by upgrading their provider. The features include application self-tuning, faster data retrieval, and optimized memory usage.

Application Self-Tuning

ODP.NET enables dynamic application self-tuning of the statement cache. By sampling run time query execution, ODP.NET dynamically resizes the statement cache to optimize overall performance. The statement cache size is increased dynamically if frequently executed statements are not being cached. The statement cache is decreased dynamically if excessive memory usage is slowing overall application performance. This feature enhances performance, reduces network usage, and saves on client and server memory usage, while increasing developer productivity, especially if statement cache size hasn't been tuned previously.

Developers no longer need to manually optimize the ODP.NET statement cache. It is unnecessary to experiment with different cache sizes, determine which statements to cache, and later resize the cache when application behavior changes over time.

Statement caching itself allows ODP.NET to cache statement parses for frequently executed SQL and PL/SQL. Oracle retains the parsed statements in the shared pool. Because the client cursor remains open, no additional server cursor lookup is required. In addition, statement metadata remains on the client. All these improvements speed SQL and PL/SQL execution.

Before ODP.NET 11.1.0.7.20, developers would set the statement cache size at design time, thus caching a user-defined number of most recently executed statements. Over time, the least recently executed statements were removed from the cache.

Because statement cache size was set at design time, it was difficult to ensure the cache size would remain optimal at run time. The number of frequently executed statements could change over time, be it over the course of a day or over the course of a year. A cache that is too small leads to unnecessary parsing and lookup. A cache that is too large leads to excessive memory usage. ODP.NET application self-tuning overcomes these obstacles and manages the statement cache automatically.

Application self-tuning is enabled by default in the Windows Registry. Self tuning can be set with the *Self Tuning* connection string attribute; in a .NET configuration file, such as *web.config* or application configuration file; and in the Windows Registry. If the connection string attribute is

set, it overrides the .NET configuration file setting for that application. Along the same lines, setting the .NET configuration self-tuning attribute overrides the Registry setting.

Faster Data Retrieval and Optimized Memory Usage

ODP.NET runtime performance has been internally optimized to speed data retrieval using *OracleDataReader* or populating a *DataSet*. This feature provides better performance and scalability for ODP.NET applications.

In addition, ODP.NET reuses the same fetch array buffer, rather than create a new buffer for every statement executed serially. The fetch array buffer stores data for executed statements. This feature reduces client memory usage since ODP.NET requires fewer fetch array buffers for typical applications.

Developers should see faster data retrieval performance and lower memory usage just by upgrading ODP.NET.

Code Access Security

Code access security limits .NET assemblies from access to protected resources and operations. It allows developers to manage system resource permission and code group security policies. ODP.NET provides both imperative and declarative security for client applications with *OraclePermission* and *OraclePermissionAttribute* classes, respectively. These new classes enable easier and secure code access management with ODP.NET and Oracle database.

In ODP.NET 11.1.0.7.20, code access security ensures that .NET assembly access to the Oracle database can be sufficiently restricted. Only *OraclePermission* granted assemblies can connect to the Oracle database. A security exception is thrown if *OraclePermission* is not granted to the assembly. This rule applies to all the assemblies in the call stack. They all must have *OraclePermission* granted for database access.

OraclePermission provides flexible control of database access. ODP.NET developers can grant or deny permission based on specific connection string attributes and attribute values. Consider the following example:

OraclePermission.Add("Data Source=orcl;"," User Id=;Password=;", KeyRestrictionBehavior.AllowOnly);

This code segment grants connection permission for connection strings with *Data Source* set to 'orcl'; with any user id and password combination; and with no other connection string attributes. ODP.NET throws a security exception if any other connection string combination tries to connect.

Because *OraclePermission* and *OraclePermissionAttribute* classes respectively inherit from System.Data.Common.DBDataPermission and DBDataPermissionAttrubute, all the standard .NET code access security functionality is available to ODP.NET developers. Configuration can

be performed with NET Framework Configuration tool (Mscorcfg.msc) or manually by modifying application configuration file, web.config, security.config, and/or Windows Registry, such as modifying the *DemandOraclePermission* configuration attribute, which enables or disables *OraclePermission* demand.

High Availability Event Notification and Callback

Starting with ODP.NET 10.2, .NET applications could use Fast Application Notification (FAN) events to load balance ODP.NET connections across a cluster and to initiate Fast Connection Failover (FCF) when a database resource failed. FAN events provide ODP.NET information on server host, service, service member, instance, and database status changes. This information is critical for real-time connection load balancing with Oracle Real Application Clusters (RAC) and for removing invalid connections to RAC or Oracle Data Guard from the connection pool.

ODP.NET managed FAN events internally, which meant .NET developers could not design their own application logic to respond to FAN events. With ODP.NET 11.1.0.7.20, developers can now register their own .NET callbacks for FAN events, and then execute an event handler based on the event. With this feature, application developers can better manage their .NET applications in response to database component failures or additions.

When the high availability event occurs, the event handler consumes and acts on the database event data returned to the client. The event data includes the event source, instance name, service name, hostname, and source status among other information. Events are triggered whenever a server resource fails or becomes available. The server resource could be a host, service, service member, instance, or database.

To use this feature programmatically:

1. Set the *HA Events* attribute to true in the ODP.NET connection string.

2. Define an event handler, *HAEventHandler*, for the application.

3. Register the callback, *OracleConnection.HAEvent += new OracleHAEventHandler(HAEventHandler)*, with the application.

Programmatic Database Startup and Shutdown

ODP.NET users with database administrator privileges can now startup or shutdown a database programmatically. This feature is useful for ODP.NET applications that manage Oracle databases.

Starting up or shutting down a database requires database administrator privileges, SYSDBA or SYSOPER. This privilege can be set in the connection string attribute *DBA Privilege*. The *OracleDatabase* class uses a special purpose connection for only starting and shutting down a

database instance. The class contains methods that provide control over what modes to startup or shutdown the database in.

Conclusion

ODP.NET 11.1.0.7.20 includes numerous new features to improve developer productivity, application performance, scalability, and security. .NET developers can now use unique Oracle database functionality, including Oracle Streams Advanced Queuing; application self-tuning; faster data retrieval and improved memory management; high availability event notification callbacks; and programmatic database startup and shutdown. At the same time, more .NET Framework functionality is now available with promotable transactions and .NET Code Access Security. .NET developers can take advantage of the best of both worlds in Oracle Database and .NET Framework.

What's New in Oracle Data Provider for .NET
11.1.0.7.20
September 2009
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
oracle.com

Oracle is committed to developing practices and products that help protect the environment