



## **Best Practices with PervasiveOLEDB**

The Advanced Guide for using ADO and Pervasive.SQL™ 2000i

Revision 1

April 2001

---

*Pervasive Software Inc.*

Pervasive Software Inc.  
12365 Riata Trace Parkway II  
Austin, Texas 78727  
Public Relations Contact: Marian Kelley  
Telephone: 800-287-4383/ 512-231-6000  
Fax: 512-231-6010  
Internet: <http://www.pervasive.com>

© 2001 Pervasive Software Inc., Pervasive, Pervasive.SQL, Btrieve, and the Pervasive logo are registered trademarks of Pervasive Software Inc. All other product names are trademarks of their respective companies. All rights reserved worldwide.

## INTRODUCTION

---

Pervasive Software released its first OLE DB provider with the release of Pervasive.SQL 2000 in June of 1999. Since then, we have been busily creating an OLEDB provider that is richer in features, functionality, and performance—everything a developer would want from a data access method. Pervasive.SQL 2000i (Service Pack 3) is this database.

Some assume that OLE DB development is straightforward and relatively easy to master. But that is not necessarily the case. While developing the new provider and referring to the documentation for ADO, we have noticed some of the most interesting features are not very well represented. These features can double and triple performance if used correctly. Furthermore, commonly referred to ADO documentation specifically mentions “Best Practices” that can degrade performance.

Enter the “Best Practices of PervasiveOLEDB.” This paper was written to inform the developer of pitfalls that they can, and probably will, encounter.

The intended audience includes:

- Developers currently using ADO within applications that are considering PervasiveOLEDB as the underlying provider
- Developers using PervasiveOLEDB as their underlying provider who would like to further understand the implementation details of PervasiveOLEDB

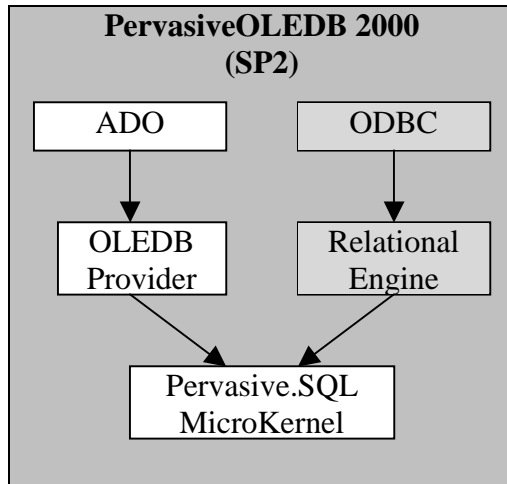
This document assumes an in-depth knowledge of ADO and programming techniques.

## ARCHITECTURE (OLEDB AT A GLANCE)

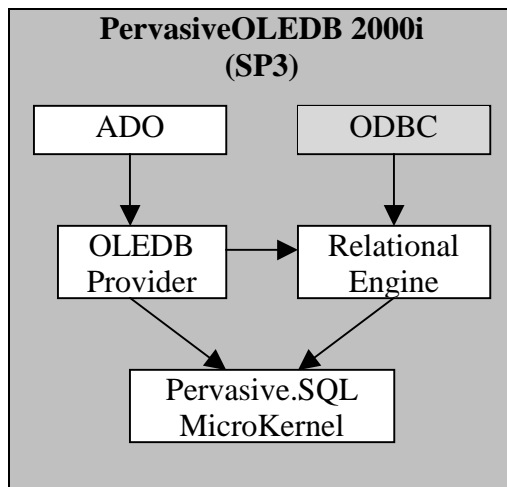
---

PervasiveOLEDB is the name of Pervasive Software’s implementation of the OLEDB Provider Specification written by Microsoft.

The first iteration of PervasiveOLEDB was released with Pervasive.SQL 2000. This initial version of the provider did not support the command object. The provider only had navigational (transactional in Pervasive terms) capabilities, and the architecture resembled the following:



One of the strengths of OLEDB is the dual access method that it allows: both relational and navigational access methods are supported. PervasiveOLEDB 2000i fully supports both access methods with the inclusion of command-based recordsets. The new architecture resembles the following:



This architecture allows for different access paradigms using the same API. Furthermore, the access is coded in the same way, except that the *Open* statement is changed.

## PERVASIVE ACCESS WITH MDAC

---

### *The nitty-gritty under-the-covers view*

Now that it is possible to reach Pervasive.SQL databases with more than one provider (PervasiveOLEDB and the Microsoft ODBC to OLEDB bridge), it is necessary to describe the architecture of both. Before we open the hood, let's make some generalizations about SQL queries in order to better describe the differences of the two architectures.

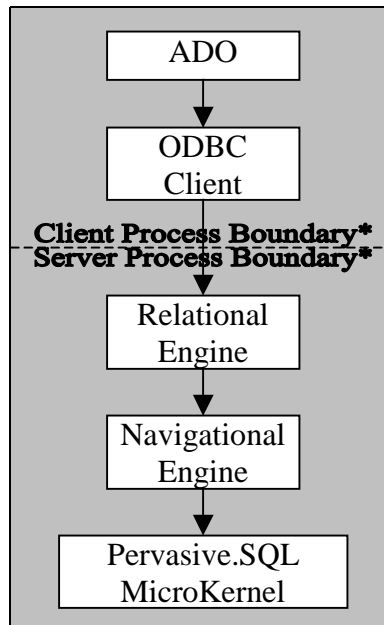
*Simple SQL query:* a query involving one table with few predicates (we could limit this definition a bit more, but for simplicity we will stay with this definition) and an index is available for each predicate column. A good

example is "SELECT \* FROM Tablename WHERE ID = 5". This will open a table and use an index to find the ID.

*Complex SQL query:* a query that must read many rows that will never be included in the resultset. An example is a predicate (WHERE clause) on a column without an index. These types of queries may try to find a column value and an index that is not available on the column. Therefore, the relational engine must check each column for the value. Another example is a query with a JOIN. In determining which rows will be considered for the final resultset, each row must be read and many may not be included in the resultset.

ODBC (thin relational client)

ODBC has been the mainstay for database access for quite some time. This architecture has been extended by ADO. Using the Microsoft ODBC to OLEDB bridge (commonly referred to as the ODBC Bridge), ADO has access to any database that has an ODBC driver.



\*You may substitute "Machine Boundary" for "Process Boundary" in a client/server architecture.

This architecture is referred to as a *thin client* because calls are passed to the server, and the bulk of the processing is done in the server process. This type of architecture is familiar to people who have used ODBC before.

The ODBC Bridge is efficient at *complex SQL queries* because the relational engine and the Pervasive.SQL MicroKernel are in the same process space. This allows the relational engine to reach the data without a process switch.

### **The Facts**

- Uses DSNs (ODBC Data Source Names) to determine the Database Name (DBName)
- If a DSN is used in the connection string and a provider is not specified, the ODBC will be the default provider

### **The Good**

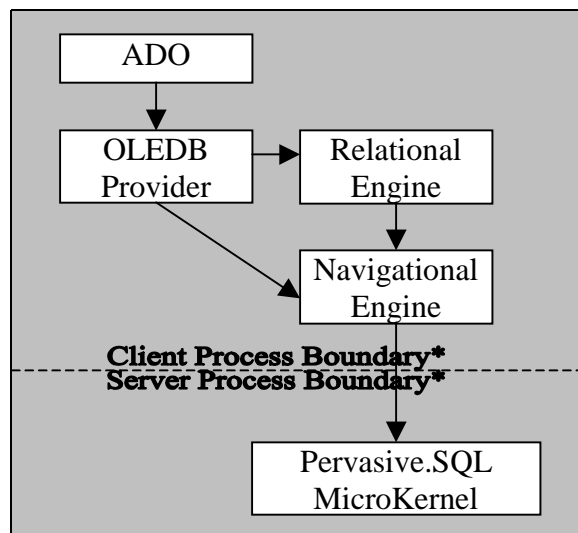
- Efficient complex SQL query processing

### **The Bad**

- Does not provide direct navigational access

## **PERVASIVEOLEDB (THICK RELATIONAL AND NAVIGATIONAL CLIENT)**

PervasiveOLEDB is Pervasive's implementation of the OLEDB specification. Embedded within PervasiveOLEDB are both the relational engine and the navigational engine. Because the bulk of the processing is done in the client process, it can be referred to as a *thick client*.



\*You can substitute "Machine Boundary" for "Process Boundary" in a client/server architecture.

PervasiveOLEDB is efficient at *simple SQL queries* because the relational engine is in the same process as the client. The Pervasive provider also allows navigational access, which allows for faster tabular access.

### **The Facts**

- OLEDB does not use DSNs, rather it uses the DBName.

### **The Good**

- PervasiveOLEDB moves the relational engine to the client process so that the relational processing is not necessarily done on the same machine as the DBMS (MicroKernel).
- Tabular data can be accessed navigationaly.

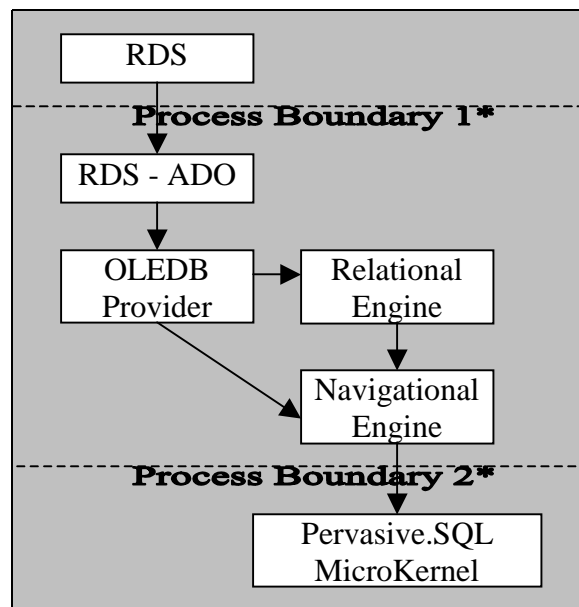
### **The Bad**

- Moving a provider to a separate machine
  - Can be costly in terms of complex SQL query performance.
  - Requires a mapped drive to the database and a server-side DBName.
- Stored Procedures are no longer executed in the server process, but rather in the PervasiveOLEDB process.

## **RDS**

---

As you may have noticed, the key difference in the architecture is where the communication between process boundaries (remoting) takes place. In the ODBC Bridge, the process boundary is above the relational engine, and in PervasiveOLEDB, the process boundary is below the navigational engine. RDS (Remote Data Services, Microsoft Technology described in MSDN) provides the additional option of remoting above the OLEDB provider layer.



Depending on the needs of your application, this approach can either be efficient or inefficient. Process Boundary 1 is usually a machine boundary (client/server boundary across a network). Process Boundary 2 is usually within the same machine. This allows RDS to intelligently remote the provider for maximum bandwidth utilization. Properties that would normally take multiple trips across a machine boundary can be efficiently determined in one trip. If bandwidth is not a concern for the application, RDS can add unnecessary marshalling overhead.

### **The Facts**

- Using the Remote Server and Remote Provider in the connection string will initiate an RDS connection

### ***The Good***

- Efficient use of property caching eases the pain of remoting a recordset across machine boundaries

### ***The Bad***

- Can result in unwanted and unnecessary process boundaries
- Concurrency issues are aggravated due to disconnected nature

## **WHICH ARCHITECTURE SUITS YOUR APPLICATION?**

---

### *Understanding the options*

When developing an application, it is best to understand the implications of any given architecture. With the introduction of command-based recordsets in PervasiveOLEDB, many developers have concluded that the new provider will fit seamlessly within their current architecture. While PervasiveOLEDB does provide developers with more flexibility in designing high-performance solutions, it remains unlikely that a single architecture is suited to every possible application. It is always best to know the options available and the advantages of each before determining which architecture is right for an application.

Bullets indicate ideal situations for each architecture.

## **PERVASIVEOLEDB**

---

- Any application that needs fast, positioned reads, updates, or deletes on tabular data (ISAM applications, Btrieve application)

PervasiveOLEDB allows the use of `adCmdTableDirect` which enables direct navigational access. (Please see `adCmdTableDirect` later in this document.)

- Server-side business logic, single/double tier applications (single machine application)

Moving the data away from the client can negatively affect performance on low bandwidth networks, especially with complex SQL queries.

- High bandwidth, low latency, client/server applications where simple SQL processing should be on the client side

Using PervasiveOLEDB moves the SQL processing to the client side, freeing the server's processing for other services.

## **ODBC BRIDGE**

---

- Client/server applications with large resultsets, particularly with non-indexed search criteria

The ODBC Bridge is inherently good at client/server applications because the interface is built around this paradigm. The bridge efficiently returns data by returning large batches at a time.

Furthermore, given that the relational engine (when using the ODBC bridge) is in the same address space, any SQL query that results in frequent communication with the navigational engine will generally be more efficient.

The client is not affected by the data processing, rather the DBMS takes the brunt of the work.

## RDS

---

- Distributed application with latency concerns

RDS opens the provider on the server side, and marshals the data to client. RDS keeps the information on the client that would otherwise negatively affect performance. Also, RDS moves data in large chunks to optimize the data movement across the network.

## PERFORMANCE TECHNIQUES IN ADO

---

There are techniques that can improve performance that are not well documented in MSDN (or the documentation is specific to Microsoft's implementation). The following sections are directed towards Pervasive's implementation.

### Opening the Table Directly - AdCmdTableDirect

#### *A table or a resultset?*

Using AdCmdTableDirect opens the table directly without going through a relational engine. Here's the use:

```
rs.open "Demodata", "Provider=PervasiveOLEDB;Data Source=Keever",  
adOpenDynamic, adLockOptimistic, adCmdTableDirect
```

Now... we've got "Demodata" opened up without the relational engine—what good does that do? Well, the access is very fast, as there is no relational engine overhead. Also, instead of using *Find* to get to a row, you can use *Seek*. *Seek* uses the engine's "GetEqual" command to find the row. *Find* forces ADO to fetch every row in the table until it finds a match for the *Find* criteria. Furthermore, if you use *Seek*, you can use the same recordset for multiple queries. Instead of using eight command objects to open eight recordsets, you can keep one recordset open for eight *Seeks*. This eliminates the overhead necessary for opening and closing the recordsets and the overhead of the SQL parsing.

However, adCmdTableDirect is not the default CommandTypeEnum. Actually, it's more confusing than that; it's not listed as an option when Intellisense tries to complete the open command.

Since most OLEDB providers do not support dual (navigational and relational) access, most do not make an implementation distinction between *adCmdTable* and *adCmdTableDirect*. Perhaps because of this common limitation, ADO does not default to *adCmdTableDirect*. Using this value when opening a recordset should not affect performance with providers that do not directly support *adCmdTableDirect* (since ADO will map the call to *adCmdTable*), but can provide significant performance advantages when working with providers that do allow direct table access, such as PervasiveOLEDB.

### ADO Implementation of CommandTypeEnum

Using the default, `adCmdUnknown`, ADO will try to open a table using a command-based recordset. For example, let's say we use the following:

```
rs.open Darin, "Provider=PervasiveOLEDB;Data Source=Demodata",  
adOpenDynamic, adLockOptimistic, adCmdUnknown
```

ADO will try to open a command-based recordset using variations of `adCmdTable` (this maps to "SELECT \* FROM Darin"), `adCmdText` (pass "Darin" directly to the engine as if it's a SQL command), and `adStoredProc`. It will not try to open the recordset directly.

## Conclusion

Need fast, positioned access on non-relational data? Use `adCmdTableDirect`: faster opens, with the ability to use navigational access.

## CursorType

*When to use Static and when to use Dynamic? (Pervasive does not currently support KeySet)*

Use Static – **only** when a snapshot of the table is completely necessary.

Use Dynamic – in all other situations!

PervasiveOLEDB creates a temporary table for each static cursor that is opened. (Open a table, and look in the directory that holds the database. Notice the \*.tmp file? That's it!) As you scroll through the original table, the temptable gets a copy of the each row you have seen. If you perform *MoveNext* operations through half the table, the temptable will be half the size of the original.

This implementation enforces the integrity of the static cursor without locking other users out of the data. (Keeping the static copy in memory is not an option, resultsets can include millions of rows.) However, there are performance implications. For each movement, there is a read and an insert (read the original, insert to the temp). Other operations have a one-to-one correlation.

The default CursorType is static, so be sure to change the type when opening the recordset, unless you specifically need a static cursor.

## Conclusion

Use a dynamic cursor unless the situation requires a static cursor. Be aware, the default cursor type is static.

## CursorLocation

*When to use adUseClient?*

Microsoft's Client Cursor Engine is useful in many situations. Microsoft has a well-written technical article about the engine in MSDN (article name: Cursor Service for OLE DB). The basic premise of the engine is to read the entire table into memory, and then do any manipulations by using SQL commands on the data store. This is particularly useful in distributed and/or disconnected environments. Unfortunately, there are also some drawbacks to this implementation.

### 1. Latency

Since the engine must read the entire table, prepare to wait for a while if the table is large. This is inconvenient if minor updates are being done to a table. It is best to use a server-side cursor for positioned updates, inserts, or deletes.

### 2. Only Static Cursors are Supported

The data that is viewed in the recordset is actually cached data, and not the data from the data store. Therefore, dynamic cursors are not available.

### 3. Updating a Row with Similar Data

Try this: create a table. Add some information to the table. Make two rows identical. Create a VB project that has a client-side connection. Open the table. Update one of the rows that has the exact same data. Doesn't work, does it? Here's why:

The update statements are parameterized sql statements in this form:

```
"UPDATE Table SET Col2Change = ? WHERE Col1=? AND Col2 IS null AND ..."
```

If two rows happen to be the same, big problems! Microsoft's Client Cursor Engine will update the rows, then return an error.

## Conclusion

- Using the client cursor engine with large tables can wear on your patience.
- Don't expect a dynamic cursor from the engine.
- When using the Client Cursor Engine, always have a non-duplicate key in the table.
- Read the cursor engine technical article provided by Microsoft. It will answer many questions.

## RecordCount

*I just want to know the number of rows!!*

In order to determine the *recordcount*, ADO must fetch every row and count how many rows were returned. So if you use *recordcount*, realize that you might be unintentionally slowing down your application.

On the other hand, for tables that are created from commands, there is sometimes no way for an engine to "know" the number of rows. For example, consider a *Complex SQL query* like a JOIN. How many rows does it have? There is no way to tell except by constructing the resultset and counting the rows.

Many developers may have noticed that a particular provider returns the recordcount very fast. If you intercepted all the method calls, you might notice that this provider does not use the previous sequence to return the number of rows. When possible to return the exact number of rows, the provider uses an undocumented interface. Since this interface is not in the OLEDB specification and ADO is known to change frequently, it is impossible for Pervasive to mimic this interface. Furthermore, we believe in conforming to specification and keeping our provider as interoperable as possible.

## Conclusion

Limit the use of RecordCount.

## Data-Bound Controls

*The lack of control*

Data-Bound Controls are useful for non-programmatic access to data. They can be useful in getting data to controls with little effort. However, in return for the ease of development, you have to relinquish control over how the data will be accessed. This can result in unintended performance degradation.

While looking at the method calls in PervasiveOLEDB from a data-bound grid control, the grid performs the following:

GetNextRows (one row)  
GetData (on the row)  
ReleaseRows (release the previous row)

Repeat for each row in the table (I had 100,000 rows.. it took a while).

Using the same data-bound grid on a different provider (the provider and the grid were created by the same company), the performance is acceptable. So why the difference? Because they had to make a decision based on a specific implementation and a generic implementation. The specific can be optimized, and the generic just has to work. Also, the defaults for the grid were specific to the database. By changing the cursor type to dynamic (instead of static) and changing the CommandTypeEnum to adCmdTableDirect, performance improved.

However, creating my own data-bound control, using a text box and a few buttons, I was able to create a much more efficient solution.

## Conclusion

When using data-bound controls, be aware of the performance implications. You may want to consider using a custom-built solution for better performance.

## FINAL THOUGHTS

---

Where to go from here

For further information regarding PervasiveOLEDB, visit [www.pervasive.com/componentzone](http://www.pervasive.com/componentzone) .

If you have questions regarding PervasiveOLEDB, Pervasive's developers frequently review [www.pervasive.com/devtalk](http://www.pervasive.com/devtalk) .

For more information about Pervasive.SQL 2000i, pricing or support, please contact one of our sales offices or visit our web site at <http://www.pervasive.com> .

**Corporate Headquarters**

**Pervasive Software Ltd.**

12365 Riata Trace Parkway, Building II  
Austin, TX 78727  
Phone: 800-287-4383 or 512-231-6000  
Fax: 512-231-6010

**International Offices**

**Pervasive Software Ltd.**

110A High Street  
Chesham  
Buckinghamshire, England HP5 1EB  
Phone: +44-1494-791119  
Fax: +44-1494-793929

**Pervasive Software European**

Service and Support  
Bessenveldstraat 25A  
B-1831 Diegem, Belgium  
Phone: +32-2-710-1660  
Fax: +32-2-718-0331

**Pervasive Software SARL**

Immueble Atria  
21, Avenue Edouard Belin  
F-92 500 Rueil Malmaison,  
France  
Phone: +33-1-55-47-17-00  
Fax: +33-1-55-47-17-07

**Pervasive Software GmbH**

Frankfurter Strasse 151d  
D-63303 Dreieich, Germany  
Phone: +49-6103-9622-00  
Fax: +49-6103-9622-05

**Pervasive Software, N.V.**

Airport Boulevard Office Park  
Bessenveldstraat 25A  
B-1831 Diegem, Belgium  
Phone: +32-2-718-0330  
Fax: +32-2-718-0331

**Pervasive Software Co., Ltd.**

World Trade Center, 33F  
2-4-1 Hamamatsu-cho  
Minato-ku, Tokyo 105-6124  
Japan  
Phone: +81-3-5405-2261  
Fax: +81-3-5405-2269

<http://www.pervasive.com>

[info@pervasive.com](mailto:info@pervasive.com)

[salesupport@pervasive.com](mailto:salesupport@pervasive.com)

[http://www.pervasive.com/support/Email\\_Support.taf](http://www.pervasive.com/support/Email_Support.taf)

---

© 2001 Pervasive Software Inc., Pervasive, Pervasive.SQL, Btrieve, and the Pervasive logo are registered trademarks of Pervasive Software Inc. All other product names are trademarks of their respective companies. All rights reserved worldwide.

*Disclaimer: The performance demonstrated in this document is for reference purposes only and does not constitute a performance guarantee even if under similar configurations.*

