

What You Don't Know About Data Connectivity CAN Hurt You

WHITE PAPER



Abstract

Data connectivity, the middleware that enables applications to access databases, is a critical, but often overlooked component of the IT infrastructure. These components, such as ODBC and JDBC drivers and ADO.NET providers, can have dramatic effects on application performance, reliability, and portability. Poor choices directly impact the bottom line through increased development costs, slower deployments, and missed revenue opportunities. In fact, one of the greatest impediments to application performance and scalability is the bottleneck between the application and the underlying database caused by flawed or suboptimal data connectivity.

This paper provides an overview of data connectivity, details its effect on application quality, and explores various alternative solutions.

Table of Contents

| | |
|---|-----------|
| Key Takeaways | 1 |
| Introduction..... | 1 |
| The Facts on Data Connectivity | 2 |
| What is Data Connectivity? | 2 |
| The Importance of Data Connectivity | 2 |
| Implications for the Application Development Organization | 3 |
| Business Considerations | 5 |
| Alternatives for Data Connectivity | 6 |
| Vendor-Provided Drivers/Providers | 6 |
| Native Connectivity | 10 |
| Open Source Drivers / Providers | 10 |
| Evaluating Third-Party Data Connectivity | 12 |
| Comprehensiveness | 12 |
| Proven | 13 |
| Technical Leadership..... | 13 |
| Conclusion | 14 |
| Appendix A – Database and Platform Support by Product Line | 1 |

Key Takeaways

- Data connectivity components, such as ODBC and JDBC drivers and ADO.NET providers, are the middleware that connect applications to databases. All communications with the database must flow through the data connectivity component.
- Data connectivity quality can have dramatic effects on key application attributes such as functionality, performance, scalability, security, and reliability.
- Data connectivity choices can have significant impact on development, testing, and customer support costs.
- Application developers who need to connect to multiple databases or database versions should choose database-independent data connectivity components.
- Application developers have four options when evaluating data connectivity: database vendor provided drivers, native/proprietary connectivity, open source drivers, and third-party professionally developed components.
- Database vendors offer inherently inferior solutions because they are designed to *limit* interoperability and portability. Database vendors do not focus significant R&D or testing resources on their components, so they are also limited in quality, performance, and feature support.
- Third-party components offer the best alternative for critical applications and applications that need to support multiple databases.
- When choosing a third-party component, application developers should look at three primary factors:
 - Comprehensiveness – How broad and deep is the product line?
 - Experience – How proven are the vendor and its products?
 - Technical leadership – Is the vendor a leader in the field?

Introduction

Most software applications today rely on relational databases to support the storage, access, and management of information. Databases holding customer, order, user, inventory, and financial information have become as important as the ERP, CRM, financial, business intelligence, and other enterprise systems that connect to them.

But what about the connection between applications and databases?

Under growing pressures to deploy increasingly complex applications against numerous database back-ends, many application developers overlook the importance of data connectivity – the critical middleware that connects applications to data.

This white paper discusses the role that data connectivity plays in overall application performance, scalability, interoperability, reliability, and development and maintenance cost and complexity. It then explores various alternatives for data connectivity and recommends considerations in evaluating these alternatives.

The Facts on Data Connectivity

What is Data Connectivity?

Data connectivity embodies all of the methods used to connect software applications to databases. Data connectivity is most often associated with standards-based components such as ODBC and JDBC drivers as well as ADO.NET data providers in Microsoft's .NET framework.

Data connectivity components contain specific knowledge about the inner-workings of the database that enable a software application to call and use the database's functionality. Data connectivity components perform five critical functions in IT infrastructure:

1. Provide a standard means (API) for applications to access the database.
2. Enable applications to update, insert, or delete data or to call stored procedures in the database.
3. Allow applications to call functionality or business logic that resides in the database engine.
4. Enable applications to retrieve (select) data from the database.
5. Provide communications between databases for data extraction, transformation, and loading (ETL).

The Importance of Data Connectivity

Data connectivity components are the conduit through which all information and commands flow between software applications and databases. The sophistication and quality of data connectivity architecture necessarily impacts the performance, scalability, reliability, and security of IT infrastructure.

Data connectivity quality and performance is a function of architectural sophistication and intimate knowledge of the underlying database engine. For example, ODBC drivers are frequently slowed down by the use of database vendor client libraries – additional software which must be installed on the same physical server as the application. Client libraries create additional steps in the process and thus reduce performance and scalability. Client-less (or wire-protocol) ODBC drivers skip this step and thus show radical improvements in performance and

scalability. Similarly, components developed without deep knowledge of the underlying database protocol will not allow applications to make full use of database functionality.

Data connectivity components are also an important aspect of software maintainability and portability. Because all database communications must pass through the connectivity component, bugs or flaws in the component's architecture or implementation cause support problems for application owners. For example, if a JDBC driver implements certain functionality defined in the JDBC specification incorrectly, application performance could be unpredictable. With regard to portability, data connectivity architecture can either simplify or radically complicate portability between databases, database versions, and platforms. Ideally, data connectivity components should share a common architecture that makes it easy to change or upgrade the underlying database infrastructure.

Implications for the Application Development Organization

Application development organizations arguably have the most to gain (or lose) from their data connectivity choices. Strategic data connectivity increases application performance and scalability, radically simplifies database and platform portability, reduces project costs, improves developer productivity, increases functionality, and speeds deployment.

Application Performance and Scalability

Engineers and application and development managers should carefully consider data connectivity when architecting new applications and when they are looking to improve performance and scalability of existing applications. Over the past several years server processing power has increased dramatically while prices have plummeted. As a result, applications and databases are no longer constrained by hardware processing power – the bottleneck has moved to other parts of the system.

At present, one of the greatest impediments to application performance and scalability is the bottleneck between the application and the underlying database caused by flawed or suboptimal data connectivity.

1. Use of client libraries: If client libraries (additional software that must be installed on the application client) are required, architecture becomes complicated and performance and scalability suffer.
2. Use of disk caching: If a data connectivity component uses disk caching rather than in-memory processing, performance can suffer severely.
3. Verbose database communications: Performance suffers if a data connectivity component uses excessive network trips to communicate with the database.
4. Support for connection pooling/Statement pooling in Java/JDBC environments – If a data connectivity component does not support connection pooling or statement pooling, performance can suffer dramatically.

Database and Platform Portability

Data connectivity architecture can either simplify or radically complicate portability among databases, database versions, and platforms. Ideally, data connectivity components should share a common architecture that makes it easy to change or upgrade the underlying database infrastructure. Most software companies and enterprise IT organizations must support more than one database platform – and more than one version of every platform they support. This can mean also managing a myriad of data connectivity methods, driver versions, and client library packages. Adding a new database or even upgrading to a new version of the same database can create substantial development, integration, and testing work.

For example, data connectivity components designed to work with only one database will handle BLOB/CLOB data types (large binary or character objects) differently than a component designed to work exclusively with another database. Developers will spend significant time and effort on additional coding and testing for each new database that they need to support.

Standardizing and simplifying data connectivity architecture dramatically reduces the cost and complexity associated with supporting multiple database back ends. For independent software vendors in particular, this is a significant business priority.

Project Costs

The time and attention spent on building, integrating, and testing data connectivity components can have substantial impact on project costs. These costs may manifest themselves in a number of ways – development time, consulting fees, testing and debugging, and re-work costs.

For example, some JDBC drivers do not support savepoints. To support this functionality, developers will have to write and test additional code. This adds unnecessary time and cost to the project.

In addition, customer support and maintenance costs increase based on the reliability and performance of a given data connectivity choice.

Application Reliability

Data connectivity bugs can be as detrimental to application reliability as those in application business logic or at the database tier. Users depend on applications to return consistent, accurate results from database queries and for dependable results from updates, inserts, and deletes.

To ensure reliability, all tiers of the application infrastructure must work flawlessly in concert – the application, the data connectivity layer, and the database. For instance, a bug in an ODBC or JDBC driver might cause a database to return erroneous results, an error message, or no results at all.

In addition, some data connectivity components enable functionality that significantly increases application reliability – such as support for distributed transactions. In Java environments where reliability is critical, developers should look for data connectivity components that support JTA (Java Transaction API).

Developer Productivity

The more time that developers spend building, integrating, and testing data connectivity, the less time they have available for new feature development and other projects within the organization. As application development organizations seek to optimize the use of existing development and QA staff, it is important that managers look carefully at how their data connectivity choices impact resource allocation.

Functionality

Database vendors are constantly innovating and adding new features and functionality to their database products. And data connectivity standards – ODBC, JDBC, and ADO.NET are frequently updated to include new functionality. However, developers cannot automatically take advantage of either category of new functionality in their applications. Their database driver or provider must support the new database functionality or new version of the standard.

Older drivers or ADO.NET provider versions most certainly will not support new functionality – and newer drivers may or may not provide a backwards compatibility for older databases. Application developers who want to take advantage of the latest changes in database functionality and latest updates of connectivity specifications must ensure that their data connectivity method contains the requisite support. At the same time, they must ensure that their applications are fully backwards compatible with previous database releases.

Deployment Speed

Configuration headaches can dramatically slow deployment time. Given the myriad of databases and database versions available, it is not surprising that data connectivity configuration can be complicated and time consuming.

To smooth deployment, engineers and application development managers should look to streamline and standardize data connectivity. This simplifies configuration and testing, and enables much more rapid application deployment.

Business Considerations

While data connectivity is patently a technical decision, its implementation and architecture has significant business implications on cost, product revenue potential, project delivery, and customer support/satisfaction.

Application Development Costs

The quality, usability, and interoperability of data connectivity components can have significant impact on the engineering resources necessary for a given software development project. Components that require substantial development effort to integrate and test can add hundreds of thousands of dollars to the total cost of a new product or project development.

Product Revenue Potential

Data connectivity is a key factor in determining interoperability between databases and database versions. For independent software vendors, it is critical to support as many databases and versions as possible because this maximizes the available

market and associated product revenue potential. Utilizing database-independent data connectivity components enables ISVs to reach more customers while maintaining a single code base.

Project Delivery Schedules

The more time developers spend integrating and testing data connectivity the longer it takes them to complete a given project. Similarly, if engineers are focused on data connectivity, they cannot simultaneously be focused on adding or integrating new features and functionality. In either case, the application and user base suffers.

Customer Support/Satisfaction

When data connectivity components fail to perform properly, users turn to customer support. This requires time and attention to diagnose and correct problems. This is primarily true for independent software vendors, but also true for enterprise IT departments who have to support custom applications. Applications based on database vendor drivers and providers are at the mercy of support and development organizations whose focus is on the database itself, rather than the connectivity components.

Alternatives for Data Connectivity

Data connectivity choices have now shown to have significant bearing on the cost, quality, and portability of application development projects. To help the developer make sense of their choices, it helps to categorize available data connectivity components into four broad categories:

Vendor-provided drivers/providers – Drivers or providers that come bundled with the database engine.

Native connectivity – Proprietary, client library-based applications developed in the absence of industry connectivity standards.

Open source drivers/providers – Components developed and distributed through open source projects.

Third-party components – Drivers and providers developed and marketed by firms that make a business of data connectivity.

Vendor-Provided Drivers/Providers

All relational databases are shipped with drivers that allow developers to connect applications to them. These components are designed for a specific version of a specific vendor's database engine. Vendor-provided components give the requisite level functionality and performance necessary to make use of the database engine.

Because they are included with the database, vendor-provided components are often seen as the most cost-effective alternative for new application development. However, these components have several critical limitations which can have significant impacts on project success and development and support costs.

Vendor database drivers are often updated by the database vendor as frequently as once a month. It is not unusual for these releases to contain inconsistent functionality from one version to the next. For ISVs, this can wreak havoc on development and QA schedules before product release. This translates into increased development man-hours and costs and adds to technical support expenses after the product release. The enterprise environment is quite similar: multiple, frequently updated driver versions increase development, testing, and support costs.

At the heart of these limitations is one key fact: database vendors are in business to develop and market database engines. They distribute drivers because they need to provide developers a means of connecting to their database.

Bearing this in mind, developers should look closely at four attributes of vendor drivers:

- Application portability
- Performance and Scalability
- Software Quality
- Feature Support

Application Portability

Vendor-provided components are designed to work with a specific version of a specific database. For instance, Oracle has 22 different versions of its ODBC driver to cover all current versions of its database engine. Not only will an application developed against an Oracle-provided driver not work with an IBM DB2 database, but it may also not work with more than one version of Oracle. Because many applications need to work with a variety of databases and database versions, this creates a problem for application developers, testers, and support personnel.

For applications that need to support multiple versions of multiple databases, the permutations and combinations present a management and versioning problem. Software developers need to build and maintain different versions of their applications for each database they intend to support. This in turn creates additional testing and retesting work and slows down project completion. Even further downstream, it also creates a support problem when application users upgrade to a new version of their database.

Additionally, since database vendors are in business to sell more database licenses, they have strong incentive to make it very hard for customers to switch to another vendor's database engine. In short, they want to create "lock-in." To achieve this, database vendors may implement proprietary extensions to data connectivity standards. This further complicates application portability. Database vendors also provide incentives for customers to upgrade to the latest versions of the database. Therefore, the latest connectivity components may only work against the current major version of the database, requiring customers to upgrade their database to utilize new components.

In some cases, there are even greater incentives against interoperability. For instance, Microsoft SQL Server includes a JDBC driver. However, Microsoft has no incentive to improve the performance and quality of a Java application because they

would prefer that customers use .NET-based applications. They have no incentive to include a high performance JDBC driver with their product. Similarly, Microsoft has a .NET provider for Oracle; however Microsoft has no incentive to provide maximum performance for an Oracle application, as they prefer that people would adopt their own SQL Server database product.

Performance/Scalability

Data connectivity is not a strategic technology for database vendors because it does not impact their revenue stream. They ship database drivers because they have to do so. As such, they focus their performance and scalability efforts on the database engine itself, not on their `drivers or providers.

Oracle ships an ADO.NET provider with its database so that Microsoft shops supporting Oracle will have the ability to access Oracle, using Oracle's components. However, Oracle has a strong commitment to Java and encourages its customers to use Java over .NET. As such, minimal efforts have been put into making their ADO.NET provider perform well, let alone scale in a typical high load environment. Customers looking to develop highly scalable, high performance applications should rely on vendor *neutral* third-party providers who have no stake in the adoption of any particular platform or database.

Software Quality

Not surprisingly, database vendors invest the majority of their testing and QA efforts on their database engine. Data connectivity components are usually only tested for specific functionality within a specific database version. These tests often do not encompass the hundreds of thousands of possible user scenarios. Nor do database vendors test to ensure that their drivers completely comply with connectivity specifications, such as ODBC or JDBC.

As a result, quality suffers and bugs are common. Poor quality may mean that databases return unpredictable results or that certain functionality simply does not work. Database vendors spend very little of their engineering resources on connectivity so bugs may not be fixed promptly, which could delay project completion.

Feature Support

It is easy to assume that database vendors would offer the most feature-rich data connectivity components because they have deep knowledge of the underlying database. However, this is frequently not the case.

Developers need to be concerned with two broad categories of data connectivity functionality:

- Functionality in the database engine
- Functionality as supported by the standard (ODBC, JDBC, etc.)

With respect to functionality in the database, this is where the majority of the database vendors focus their efforts. New features in the databases are what entice customers to upgrade their versions. However, many database vendors implement

these features in a native syntax that may not work against other databases. Often database vendors provide extended “hooks” in their database drivers that are outside the standard data access specifications. This severely limits interoperability and makes it impossible to call similar functionality in another database without recoding the application. For example, vendors may require different methods for calling stored procedures or utilize different methods of specifying parameter markers.

On the other hand, database vendors often do not support the most recent versions of connectivity standards. In fact, database vendors are typically slow to fully adopt new standards, as a proprietary strategy encourages more customer lock-in.

A glaring example of lacking functionality is Oracle’s JDBC driver, which is not JDBC 3.0 compliant. This means that it does not support key functionality such as transactional savepoints, prepared statement pooling, BLOB/CLOB updates, and multiple open result sets. The Oracle JDBC driver does not pass the J2EE JDBC certification test suite (CTS) and is not J2EE compatible, meaning it does not meet the quality levels imposed by the Java Community Process, leaving applications at high risk for quality issues and failure. The table below highlights the functionality that may be lacking in a vendor-provided JDBC driver.

| If your application uses this feature: | This feature impacts: | Limitations or Additional Work Required |
|--|------------------------------------|--|
| Connection Pooling | Performance | Connection Pooling, a JDBC 2.0 feature that some database vendor drivers still do not support, is critical for performance. If you’re running your application outside of a J2EE application server, you’ll have to write your own pool manager. |
| Statement Pooling | Performance | Statement Pooling is critical for performance. Some application servers don’t support Statement Pooling (e.g., SunOne). And if the application runs outside of an application server, developers will have to write their own statement pooling code for every statement they execute. |
| JTA for Distributed Transaction Support | Reliability | Distributed Transactions are essential when reliability is important - especially for transactions accessing more than one database. Plan on extremely difficult coding efforts to simulate distributed transactions. Otherwise, developers will have to disable distributed transactions when running against databases for drivers that do not support JTA. |
| BLOB/CLOB Data Types | Consistency and Portability | The ability to handle large binary or character objects is a driver function. If the database vendor driver doesn’t support large objects, applications will not be able to either. Developers who must connect to more than one database will be impacted by the fact that different database vendor drivers do not handle large objects consistently. Plan on extra coding to adjust for this. |

| | | |
|------------------------------|--|--|
| Multiple Open Results | Simplifies Development Process | The ability to support Multiple Open Results is a necessary feature in the JDBC 3.0 and J2EE 1.4 specifications. If the database vendor driver doesn't support it, developers will have to check their code carefully to ensure that only single result sets are used. |
| Savepoints | Simplifies Development Process | Savepoints are another JDBC 3.0 and J2EE 1.4 specification requirement. When using a database vendor driver that does not support Savepoints, developers will have to write separate JDBC statements that set a Savepoint every time one is needed. |
| Binary Streaming | Consistency and Portability | Binary streaming is a consistency issue when accessing more than one database. To work around the lack of support for binary streaming, developers will have to write custom code to implement it. |
| Batch Queries | Performance, Consistency, and Portability | With database vendor drivers, if the application needs to access more than one database, developers will be writing custom code that can handle the proprietary Batch Query behaviors implemented by these different drivers. |
| SQLState Handling | Consistency and Portability | With database vendor drivers, if the application must access more than one database, developers may have to write custom code to make the SQLState information consistent. Otherwise, error handling will be confusing and inconsistent |

Native Connectivity

In the past, many organizations felt that standardizing on a single database meant that open standards were not necessary and that writing applications to native database libraries gave them backwards compatibility and performance gains. However, in a dynamic marketplace with changing user and budget requirements, many organizations have found that they need to support additional databases or to upgrade to new versions. When this happens, developers need to rewrite existing native applications. This creates additional development, testing, and maintenance costs. Today, most organizations have adopted a standards-based approach to data connectivity – which gives them significant interoperability advantages. With the maturation and widespread adoption of the ODBC standard, the question of performance and architectural flexibility became a moot point.

Open Source Drivers / Providers

There are several open source communities producing and distributing data connectivity components on an *ad hoc* basis. Like other open source projects, these components are developed and tested by communities of developers who are not directly paid for their efforts.

These drivers are often available for little or no up-front cost and they do address some of the limitations of vendor-provided components. However, they replace them with some additional, and very serious, limitations:

- Software quality
- Performance and scalability
- Customer support
- Feature support
- Hidden costs

Software Quality

Open source projects depend on volunteer developers to build and test software. Unlike large open source projects like Linux or Apache, open source database driver projects do not boast thousands of developers building and testing new functionality. As such, there are usually no dedicated resources for testing and test plans lack rigidity.

This is particularly true of advanced features or functions because fewer developers spend time on them or have the resources available to them for testing these implementations.

Perhaps of even greater concern is the development process itself. Open source development of connectivity components typically includes the reverse-engineering of an already commercially successful component. This contributes to problems supporting any issues that might come up in the customer's deployment because the developer does not have access to the original source code.

Performance and Scalability

Data connectivity is complex and performance and scalability optimizations are extremely resource intensive for development. Because open source projects do not have focused resources on performance and scalability, these projects usually lag well behind their commercial software counterparts.

Customer Support

Open source projects do not offer formal, service-level driven customer support. Users of open source data connectivity components must rely on Internet message boards and newsgroups for customer support. The response time and quality of support inherent in these methods of customer support are simply not adequate for business critical applications.

Feature Support

As a general rule, open source projects support the most commonly used database and connectivity standard features. Advanced features may or may not be supported – and are usually not. In addition, support for new database versions or new revisions of data connectivity standards often lags well behind the release.

Hidden Costs

While the price of an open source driver is free, the cost is frequently very high.

Given the performance and feature support limitations, developers must frequently modify open source drivers to meet their needs. This creates additional development, testing, maintenance, and support costs.

Evaluating Third-Party Data Connectivity

When performance, scalability, portability, and quality are important, third-party data connectivity components are the best choice for software developers. Third-party components are most likely to support required features without forcing lock-in to a specific database platform or database version.

Within the category of third-party components, developers and IT managers should look closely before selecting a vendor. There are three key factors to examine before making a choice:

- How comprehensive is the vendor's product line?
- How proven is the vendor and its products?
- Is the vendor a technical leader or a follower?

Comprehensiveness

Database independence is a key reason for choosing a third-party data connectivity component. To make the most of database independence, a third-party vendor needs to offer a wide range of support for databases, database versions, connectivity standards, and platforms. Otherwise the advantage is lost. Replacing a database vendor driver with a point solution from a third-party is a marginal improvement in the best case. To see significant value, developers should look for a third-party component vendor who can deliver connectivity to service the widest possible range of requirements.

Questions to Ask

- What databases and which versions does the vendor support?
- Which platforms does the vendor support?
- Which connectivity standards does the vendor support?
- Does the vendor support features I need for my application?
- What is the vendor's policy on backwards compatibility/supporting new releases?
- What level of database interoperability does the vendor provide?

Proven

Data connectivity choices can have enormous impacts on the performance, reliability, and scalability of an application. It's critical that developers and development managers look closely at third-party components and the companies behind them. The key is to find a vendor whose products go through rigorous testing and have been deployed in thousands of environments.

Questions to Ask

- How many customers does the vendor have? Who are they?
- How are the vendor's products being used by customers?
- How are new products tested?
- How are products tuned and optimized for performance and scalability?
- What kind of customer support does the vendor offer?
- How long has the vendor been in business?
- Does the vendor consistently pass rigorous industry standard certification tests?

Technical Leadership

Data connectivity is both critical and complicated. Great data connectivity depends on deep knowledge of connectivity standards as well as the underlying database engines and the protocols used to communicate with them. Developers and development managers should look for a vendor that has strong relationships with the top database vendors and significant ties to major standards committees. This ensures that the vendor has the knowledge and relationships to be quick to market with fully functional products. It is also critical that a vendor employs engineers, testers, customer support professionals, and development managers that are experts in data connectivity.

Questions to Ask

- What relationships does the vendor have with Oracle, Microsoft, and IBM? What is the nature of these relationships?
- How and when does the vendor get information about new database releases, features, and architectural changes?
- What standards bodies does the vendor participate in? What role does it play on these committees?
- How and when does the vendor get information about changes and modifications to data connectivity standards?
- Who are the top data connectivity professionals on staff? What are their credentials?
- How many engineers does the vendor have on staff?

- How much does the vendor spend annually on research and development?
- How does the vendor train its customer support personnel?
- Does the vendor provide any training to its customers?

Conclusion

Data connectivity is a critical, but often overlooked, component of successful application development. Data connectivity choices can have dramatic effects on application performance, scalability, portability, and reliability. In addition, these choices can have significant impacts on the time and resources necessary to develop and test an application, and therefore on project costs and delivery schedules. Application developers and managers should closely examine their connectivity choices as early as possible in a development effort.

While all major database vendors ship drivers with their products, these components have significant limitations. Database vendors are focused on selling databases so they offer connectivity components that will facilitate this goal while diverting minimal resources from core database development. Vendor provided components are developed and tested for a specific version of a specific database and are purposely designed to make it difficult to switch between databases. Because database vendors do not invest extensively in connectivity, vendor provided components often lack the quality, performance and scalability required for critical applications.

Third-party components are the best choice for application developers and managers. Buyers should seek components from vendors with a comprehensive product line, a long track record of delivering proven solutions, and a strong leadership position in the data connectivity market.

Appendix A – Database and Platform Support by Product Line

| Product Line | Oracle | IBM DB2 | Microsoft SQL Server | Sybase | Informix | Progress |
|---------------------------------------|---|--|---|---|--|--|
| DataDirect Connect for ODBC | Windows, Linux HP-UX IPF, HP-UX PA Risc, Solaris, AIX, Irix, Tru64, Mac | Windows, Linux HP-UX PA Risc, Solaris, AIX | Windows, Linux HP-UX IPF, HP-UX PA Risc, Solaris, AIX, Irix, Tru64, Mac | Windows, Linux HP-UX IPF, HP-UX PA Risc, Solaris, AIX, Irix, Tru64, Mac | Windows, Linux HP-UX IPF, HP-UX PA Risc, Solaris, AIX, Irix, Tru64 | Windows, Linux, HP-UX PA Risc, Solaris, AIX (Available via DataDirect SequeLink) |
| DataDirect Connect64 for ODBC | Windows, HP-UX IPF, AIX | Windows, HP-UX IPF, AIX | Windows, HP-UX IPF, AIX | Windows, HP-UX IPF, AIX | Windows, HP-UX IPF, AIX | |
| DataDirect Connect for JDBC | All Operating Platforms | Window, Unix, Linux, z/OS, OS/390, AS/400 | All Operating Platforms | All Operating Platforms | All Operating Platforms | Windows, HP-UX, AIX, Tru64, Solaris (Available via DataDirect SequeLink) |
| DataDirect Connect for ADO.NET | Windows (98, 2000, ME, XP, Server 2003) | Windows (98, 2000, ME, XP, Server 2003) | Windows (98, 2000, ME, XP, Server 2003) | Available via DataDirect SequeLink | Available via DataDirect SequeLink | |

For complete information about supported databases and platforms, and for additional product offerings, please visit www.datadirect.com

FOR MORE INFORMATION

DataDirect Technologies is the leading provider of components for connecting software to data, providing standards-based technology that ensures consistent behavior and performance across diverse environments such as J2EE, .NET, Web, and client/server. With the most comprehensive support for ODBC, JDBC, ADO, ADO.NET and XML, DataDirect Technologies provides the easiest experience connecting software to data.

datadirect.com

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 |

800-876-3101

www.datadirect.com

info@datadirect.com