

Embedded Storage: Build or Buy?

An Oracle White Paper
October 2008

Embedded Storage: Build or Buy?

INTRODUCTION

It is common software engineering practice to incorporate reusable components whenever possible into software products to focus limited engineering resources on the competitive aspects of a product. Choosing to build in-house if there is a reusable component available can be a costly mistake.

Commercial off-the-shelf software components from trusted partners can speed the development of high-quality software products.

There are many obvious situations where choosing to use a commercial off-the-shelf (COTS) software library is a foregone conclusion. Engineers would never consider rebuilding a graphics engine, network protocol stack, or encryption library. These are things better left to specialists, and the right choice in those cases is clearly to reuse, rather than reinvent.

One-Size-Fits-All isn't the case for data management requirements.

It would seem that the same reasoning should also apply to data management, but unfortunately, the case is not always as obvious. Whereas encryption, graphics rendering, and networking all have very well defined use cases, making it relatively easy to design a “one size fits all” programmatic API to enable reuse across a wide variety of applications, data management is not as easy to encapsulate. Access patterns, durability requirements, scalability, throughput, encoding, and schema are just some of the very complex dimensions wherein requirements may differ from application to application. Some common storage techniques have evolved, including relational databases with SQL and more recently XML/XQuery databases. Although highly successful across a broad spectrum of applications, these and other data models have their limits. Not all applications can afford or need the overhead of transforming data to and from such abstract representations. There are many software applications that need conventional database features such as transactions, recovery, replication, scalability, LRU data caching, concurrency/locking, indexed access, etc., but for which the SQL or XML schema and query layers are unnecessary or inappropriate.

If you need an embedded data management component, without the overhead of query processing or data transformation, consider Oracle Berkeley DB.

Oracle Berkeley DB's design provides these features as a reusable software component enabling customization in order to meet your application requirements (e.g. speed, scalability, latency, durability, availability, etc.). By using Oracle Berkeley DB, engineering teams can avoid building a solution from scratch. Oracle Berkeley DB is the only customizable database storage engine designed as a reusable software component with the maturity, features, and dependability you need for your most demanding applications. Don't re-write, reuse.

A COMPONENT FROM THE START

In the previous paragraph, we called Oracle Berkeley DB a “reusable software component”. To understand this better, we should take a quick look at its history.

Oracle Berkeley DB was designed as an embedded component from its very inception.

Oracle Berkeley DB is a descendent of the “DB 1.85” package that shipped with 4.4 BSD in 1991 to replace the proprietary `dbm` and `hsearch` packages. It provides a uniform way to access both in-memory and disk-based btree and hash storage. In 1992, Margo Seltzer and Michael Olson wrote a paper demonstrating that transaction support in DB 1.85 offered performance superior to that found in traditional relational database systems.

Adopted in a large number of industrial and academic projects, including the University of Michigan’s Lightweight Directory Access Protocol (LDAP) server, Berkeley DB 1.85 quickly became a trusted storage component. In 1996, Netscape Corporation hired the programmers from the University of Michigan to create a commercial LDAP server. In addition to the functionality provided by Berkeley DB 1.85, they needed the functionality described in Seltzer and Olson’s paper. The original Berkeley DB team of Margo Seltzer and Keith Bostic rewrote the library to include these and other new features and then commercialized the product under a dual-license open source business model¹. This was the start of Sleepycat Software.

Later, in 2003, responding to market and customer demand, Sleepycat introduced Berkeley DB XML and Berkeley DB Java Edition, expanding the family of Berkeley DB products. These two new members of the Berkeley DB product family followed in their predecessor’s design principals, providing data management components in the form of embeddable software libraries for non-relational data management.

Acquired by Oracle Corporation in 2006, Sleepycat’s Berkeley DB products are now part of the Embedded Database Group along with Oracle TimesTen and Oracle Lite. Oracle has invested significantly in Berkeley DB development and sales since the acquisition. Many Oracle products now embed Berkeley DB in ways similar to our commercial and open source customers.

EMBEDDED COMMERCIAL COMPONENTS

So how are embedded commercial components different from discrete or stand-alone components or solutions?

Oracle Berkeley DB is an embedded, reusable software component that provides a generic database storage API.

An embedded commercial off-the-shelf (COTS) component is different from a discrete component; once integrated into an application, an embedded component is transparent and can be self-managing. To do this, the embedded COTS component is not a stand-alone server or service, but rather a software library that

¹ Olson, M., Bostic, K., Seltzer, M., [Berkeley DB](#) *Proceedings of the 1999 FREENIX Conference*, June, 1999, Monterey, CA.

becomes part of the application. Oracle Berkeley DB was designed with this goal in mind.

The embedded nature of Oracle Berkeley DB is why most people are completely unaware of its existence, despite its pervasiveness in daily life. Berkeley DB is unique in that it has always been focused on delivering the sophisticated features of a database as a reusable programmer's library (a.k.a. software component). The ability to embed and reuse a generic database storage API sets Berkeley DB apart from other embeddable database solutions.

BENEFITS

Let us examine some of the benefits of an embedded COTS component.

➤ **TIME TO MARKET**

“When we did the analysis, Berkeley DB proved to be highly sophisticated while remaining cost effective.” – Ralph Anzarouth, Director Systems and Network Management, Mitel

Using COTS software reduces time to market and costs during design and development. Design focuses on integration rather than re-invention. Development centers on utilization of the component and testing can focus on the application, not on data management.

➤ **RESOURCES**

“With Oracle Berkeley DB, we were able to minimize storage overhead, maximize our throughput, ensure 24x6 service, and obtain the lowest latency possible for our traders” – Aleksey Dukhnyakov, Vice President, Tbricks

Effective use of available resources is critical for any software project. By reusing embedded software components, limited development personnel are available to focus on the company's core competencies and unique value proposition.

Embedded data management components can also play a key role in maximizing the utilization of hardware resources. Oracle Berkeley DB goes even further by eliminating the data transformation overhead inherent in traditional relational database solutions. By using an embedded software component, the embedding product can “do more processing with less hardware”, thus reducing the total cost of ownership or operation of the solution.

➤ **QUALITY**

“Speed, reliability and ease-of-use are important considerations for telecommunication infrastructure equipment. We're pleased that Oracle continues to improve Berkeley DB for the telecommunications industry.” – Rich Wong, SVP Products & Solutions Group, Openwave Systems

COTS software is constantly under review and maintenance, making it a robust and high quality alternative to an in-house custom-built solution. Commercial

**Oracle Berkeley DB allows you to do more
with less.**

products have generally been in the market long enough to be field-tested in mission-critical systems and proven reliable. A new in-house component, by contrast, is really only tested under fire when deployed in large, demanding customer environments. New products cannot risk failures in the field from untested components.

➤ **APPLICATION SOFTWARE LIFECYCLE**

“Oracle Berkeley DB XML is the database that is behind our ‘Zero Maintenance, 100% Confidence’ message. Other databases just can’t compete in a cloud computing business model.” – Paul McNamara, CEO Coghead

Oracle Berkeley DB is continuously tested during development, source is available for customer inspection, and serious bugs are quickly identified and patched when necessary.

Once deployed, products require on-going maintenance and enhancement until end of life. The IEEE91 definition of software maintenance is “the process of modifying the software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a change in the environment.” This is one of the primary benefits of COTS software and the single largest underestimated cost of in-house development.

Software maintenance consists of these four primary aspects:²

1. **Perfective maintenance:** changes required as a result of user requests (a.k.a. *evolutionary* maintenance)
2. **Adaptive maintenance:** changes needed as a consequence of operating system, hardware, or other external changes
3. **Corrective maintenance:** the identification and removal of faults in the software
4. **Preventative maintenance:** changes made to software to make it more maintainable

Not all aspects of the software life-lifecycle are properly budgeted when estimating in-house development costs.

All four aspects of software maintenance are precisely what define the development goals for a dedicated COTS development team. However, commonly only corrective maintenance is planned for and estimated when budgeting in-house development. This can be a costly mistake.³

In addition, there is the risk of losing key employees, the original authors or those most experienced with the software. In contrast, an embedded COTS component leverages the partner's product development and maintenance teams to provide ongoing bug fixes, upgrades and updates, ensuring a long life for the embedded components. The partner providing the software component will hire and keep specialists on staff to support the component over time, freeing you to hire employees who enhance your core competencies.

² Hiren Variava – SE682, 4/26/2004, <http://pheatt.emporia.edu/courses/2006/cs552s06/softwaremaintenance/SoftwareMaintenance.ppt>

³ Bennett, Keith H., “Software Maintenance: A Tutorial.” Software Engineering, M. Dorfman and R.H. Thayer, eds., IEEE Computer Society Press, Los Alamitos, Calif., 1997

BENEFITS TO YOUR CUSTOMER

“Berkeley DB was 20 times faster than other databases. It has the operational speed of a main memory database, the startup and shut down speed of a disk-resident database, and does not have the overhead of a client-server inter-process communication.” – Ray van Tassle, Senior Staff Engineer, Motorola

Software that uses embedded COTS components provides a complete turnkey solution for the end-user. The customer avoids having to deal with multiple vendors, as they might when using discrete COTS solutions, and avoids potential integration and configuration hassles. The net effect is a lower total cost of ownership for the customer end-user. In the customer’s eyes, there is a single contact for service and support. This enables a higher level of service to the customer, while the company's partner provides backline support for the embeddable COTS component when necessary.

A CLEAR BUSINESS ADVANTAGE

“When will we be able to restart the tests?” asked the customer, “We’ve been up and running since the machine rebooted.” – Howard Chu, CTO Symas Corporation responding to a prospective customer during a demonstration of OpenLDAP, an open source product that uses Berkeley DB exclusively to store data, which was able to recover a database larger than one terabyte (1TB) in size from a simulated hardware failure in under 45 seconds. The competition required over two days.

Oracle Berkeley DB is a commercial off-the-shelf component for embedded storage and used in everything from constrained, low-powered devices to global multi-terabyte replicated systems on the most modern hardware. In all these use cases, a relational model was not a natural choice; Berkeley DB provided all the sophisticated features required without the overhead of relational, query-based, client/server layers.

Collectively, these business and technical advantages of using Berkeley DB for storage demonstrate that it is possible to deliver products that combine embedded COTS and custom in-house software delivered to end customers at a lower cost, in less time, and with greater confidence.

EMBEDDED COTS DATABASE COMPONENTS

In the previous sections, we have discussed the benefits of re-usable software components, as well as the benefit of embedded software components. Let us turn our attention specifically to embedded COTS **database** components.

Data is critical to most applications. Applications rarely exist without some amount of data that survives from one invocation to the next. Applications need to save, then later retrieve and update, some amount of information that resides on persistent storage (a disk drive, flash memory, etc.) in some predictable and managed manner. The data can be anything, from simple configuration

Oracle Berkeley DB provides scalability and advanced features that become crucial as the embedding application’s requirements evolve.

So what do you do when the data management requirements do not fit the standard relational model?

information to critical encrypted customer account records, but in almost all applications, there is a need to store and manage data.

There are cases where data models, query languages, and other pre-conceived notions are not a good fit. Just as you would never consider implementing your own relational database to manage your relational data, you should not consider implementing your own data management layer to manage your non-relational data.

SELF-STRUCTURED DATA MANAGEMENT

This additional class of data management is what we will call self-structured or custom data management. Developers programmatically tailor the underlying storage to meet specific design goals. Generally, despite being unsuited for the more traditional storage abstractions, the design will require some, if not all, of the underlying features provided by a relational database management system including scalability, throughput, concurrent access, data integrity and recoverability in the face of failures of all varieties, and many other complex conditions. A proof-of-concept or first version implementation may be satisfied by a simple formatted text file or other quickly constructed in-house storage layer, but rarely will that initial implementation support the evolving needs of your customers in the field.

COTS DATA MANAGEMENT CHOICES

These days data management decisions fall one of two major categories: relational or non-relational, essentially a decision between structured, semi-structured, or unstructured data management. For years now relational storage addressed a wide variety of data management needs. In most cases, the data fit nicely into the rows, columns, and inter-related/dependant tables that this model supports. Recently, data that do not naturally fit the relational model are cleverly adapted to do so despite a clear mismatch (object-to-relational mapping, ORM, being the primary example). Semi-structured data placed into highly structured storage with mapping in-between; a workable, yet not ideal solution.

Databases that manage XML step away from the relational model while retaining most of its benefits. XML and the XQuery standards provide a way to manage flexible, sometimes self-describing, semi-structured data for a range of data management needs not easily addressed by the relational model. Content management, where documents have some structure but are not necessarily regular and can change over time, is an excellent example. Messages sent to and from SaaS and Web2.0 products use XML as an intermediate representation where the services interacting with one another store and process that somewhat unpredictable data. XML databases exist as either network services (client/server) or embeddable COTS components. The design of Oracle Berkeley DB XML targets the COTS use cases in the form of an embedded, re-usable XML/XQuery data management component.

Oracle Berkeley DB provides the scalability and the features that you need today and tomorrow.

Data management engines are at the far end of the spectrum and manage unstructured or self-structured data. Solutions in this space tend to be embeddable COTS components. Here the programmer will tailor the format, encoding, relationships, ordering, and constraints on the data and the storage engine will manage the rest. The programmer has every opportunity to tailor operations in the database (durability, consistency, atomicity, etc.) at a fine-grained per-operation level to meet the specific design requirements of the application. Oracle Berkeley DB and Berkeley DB Java Edition follow this model, and are the most generic of storage engines available. With these products, designers and programmers have the flexibility and freedom to manage all aspects of the underlying storage to design a storage strategy and procedures that fit their exact needs without having to re-write the complex storage logic themselves.

WHY NOT SIMPLY BUILD IT IN-HOUSE

Most developers at one time or another are exposed to the basic algorithms involved in data management (hash tables, btrees, queues, etc.) and concepts at a high level. The algorithms behind hash tables, binary trees, queues, and other common data structures are well known, but clearly it takes more than that to build an effective database.

Most products begin with a few simple requirements, but quickly balloon as customers exercise the product pushing it in unanticipated directions. Design requirements may not include scalability, concurrency, fault tolerance, recoverability, efficiency, data integrity, online backups, redundancy, out-of-space management, and many other features found in commercial relational databases until after the first version ships. However, these features are crucial for most customers, can be a huge competitive advantage, and are usually on the roadmap for future versions of the product.

The subtleties of these more sophisticated features are highly complex and time consuming to implement, debug and maintain. As with the encryption, graphics, and network libraries mentioned above these are things best left to specialists, done once and reused. In this situation, your resources can work on competitive advantage rather than areas of the product better left to a COTS software solution.

BERKELEY DB

Berkeley DB products are the right COTS solutions for data management hidden within your application. In all cases where you need sophisticated tailored semi-structured XML, Java object graph persistence, or unstructured customizable data management, you should first consider Oracle Berkeley DB. Its flexibility allows you to use it in environments ranging from a small phone or other constrained device to a globally distributed service requiring fast scalable performance with minimal processing and administrative overhead. Oracle Berkeley DB is the right choice for data management solutions where you find that you are asking the

question, “Nothing seems to fit, why not simply build our own file-based storage layer?”

Let us briefly compare the build versus buy pros and cons of using Oracle Berkeley DB in the following table:

	Oracle Berkeley DB	Custom Built
	Your product embeds Berkeley DB as a reusable embedded data management component, taking advantage of its features and functionality.	Your engineering team designs and builds a custom embedded data management component for your products’ particular needs.
Pros	<ul style="list-style-type: none"> You will be embedding a robust and widely deployed component with years of engineering expertise behind it. Out of the box functionality built into Berkeley DB eliminates weeks, if not months, from your engineering cycle to build a custom component. A well-tested component eliminates the need for additional QA cycles to test a custom built component. Backed by Oracle Corporation and 24x7 Support. Lower total cost of ownership over time when you factor in custom built engineering, QA, support, and other overhead. Built-in scalability and features support your products evolving requirements. Maintained and actively engineered by Oracle, providing future upgrades and added features when needed without developing them yourself. 	<ul style="list-style-type: none"> Meets the exact needs for your product requirements without any added overhead or features not yet required in this version. No third party licensing costs or fees. Flexibility to change the custom built component over time based on needs. In-house engineering and deliverables versus relying on external, third party timelines and deliverables, if required features are not already available in Berkeley DB
Cons	<ul style="list-style-type: none"> Berkeley DB may seem to be overly complex for version 1.0 of your product. Licensing costs to acquire the rights to use the software. Ramp up time and cost to learn Berkeley DB and understand how to utilize and tune the various features for maximum effectiveness. 	<ul style="list-style-type: none"> Costs related to resources and man-hours spent on a data management component outside your products domain and your team’s expertise. Potential for component feature creep, due to additional customer demand could exceed your initial team’s ability to deliver a timely solution. Spending your critical, highly qualified personnel to focus on non-core, non competitive features which are available in a COTS component. Documentation and knowledge transfer of a custom built component to reduce risk of single point of engineering failure. Long term engineering, QA, and support costs to support a custom built component. Potential costs if all of the required functionality cannot be built in-house for the custom embedded data management component.

CONCLUSION

Building great products depends on many things, most of all it requires a focus on the core aspects of your product's advantage in the market. Aspects of the solution that are not core are an on-going cost and a risk, not just in the first stages of development, but over time as the product matures and the market grows. Choosing embedded COTS software components from trusted experts can accelerate your implementation, provide advanced features earlier, eliminate concerns related to growth, and focus resources where they are most effective. Hidden, and underestimated in-house development costs can quickly become costly, cumbersome, and hard to fix. Working with a COTS partner bounds costs, eliminates risk, and future-proofs your product.

Oracle Berkeley DB products are the best COTS semi- or unstructured embedded data management components on the market today. When an application's embedded data management requirements seem to be outside the realm of well-known existing data management solutions, consider evaluating Berkeley DB before investing heavily in in-house development projects.

More information about the Oracle Berkeley DB products is available online. Downloads of the products, including complete source code, are also available for commercial evaluation or use in open source projects.

Oracle: <http://www.oracle.com/database/berkeley-db>

Oracle OTN: <http://www.oracle.com/technology/products/berkeley-db>

Documentation for each product is available online:

<http://www.oracle.com/technology/documentation/berkeley-db/db>

<http://www.oracle.com/technology/documentation/berkeley-db/je>

<http://www.oracle.com/technology/documentation/berkeley-db/db>

If you are evaluating Oracle Berkeley DB for commercial use, please contact us at berkeleydb-info_us@oracle.com



Embedded Storage: Build or Buy?
October 2008
Author: Greg Burd

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

Copyright © 2008, Oracle. All rights reserved.

This document is provided for information purposes only and the contents hereof are subject to change without notice.

This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.