

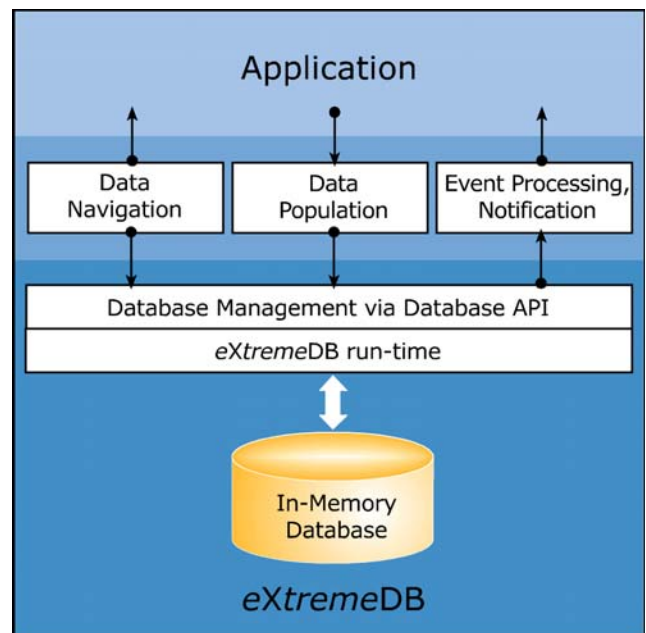
A new approach to data management for embedded devices and real-time enterprise applications. Unparalleled performance, with tools for easier integration.

Embedded system and real-time application design is increasingly complex. Performance and reliability requirements on shorter product cycles are driving the incorporation of commercially proven software components. McObject's *eXtremeDB*[™] in-memory and on-disk embedded database system combines unmatched performance, reliability and developer efficiency in an industrial-strength software package.

Today's intelligent devices – such as telecom and network infrastructure equipment, consumer electronics, and industrial controllers – process growing volumes of complex data. Managing this data requires instant responsiveness with minimal RAM and CPU demands.

Specialized enterprise systems in fields such as business intelligence, capital markets and science/engineering need real-time performance, with special features like SQL and very large database (VLDB) support.

Recognizing that traditional business-oriented database software was stretched thin trying to meet these needs, McObject built *eXtremeDB*, an in-memory database system (IMDS) that is extremely frugal in its resource use, and meets demanding requirements for performance, reliability and ease of implementation. *eXtremeDB* offers scalability into the Terabyte-plus range, with the flexibility of both SQL and native APIs. With new *eXtremeDB* Fusion, developers can even blend on-disk and in-memory data storage.



System Architecture

" *eXtremeDB* helped cut 18 programmer months from the development cycle."

- *The Boeing Company*

The Need for a Real-Time Data Management Solution

As more embedded devices become networked, their need to manipulate shared data increases exponentially. Organizations are also recognizing the competitive advantage of high performance in time-sensitive, data-intensive tasks, and are augmenting their IT infrastructure with new systems for functions ranging from securities trading to business intelligence.

Selecting a proven commercial database system for real-time enterprise and embedded systems must include consideration of price and of the vendor’s track record. High run-time performance is a must. The system must provide a programmer-friendly integration environment, resulting in a shorter development cycle and more legible/maintainable code.

For the runtime environment:

- High transaction rate
- Small footprint & compact data layout
- Transactions support ACID properties
- Direct data access
- Support for multiple data types
- Compatible with leading OSs and RTOSs
- Very large database (VLDB) support
- Choice of on-disk, in-memory or combined data storage

For development:

- Source code available
- Intuitive native API
- Built-in consistency and error checking
- Easy to learn standard functions
- Flexible and efficient data queries
- Generates maintainable code
- Compatibility with existing systems via SQL and XML interfaces

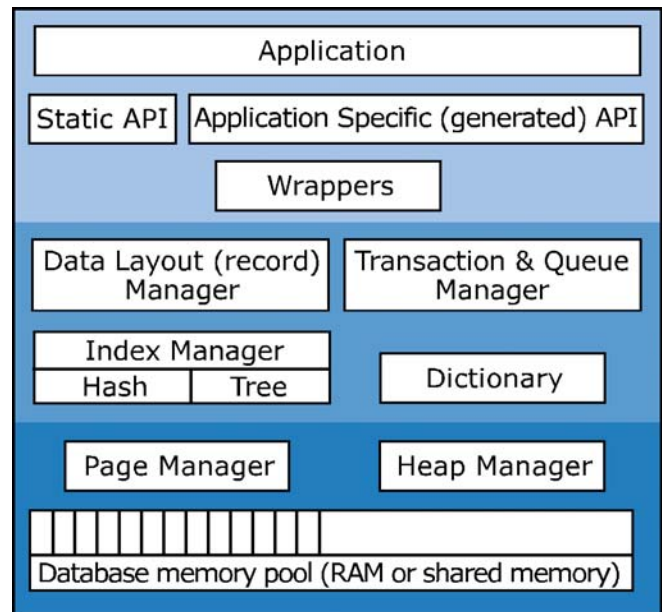
Only *eXtremeDB* from McObject addresses both sets of needs. *eXtremeDB* combines world-class data management execution with a developer-friendly design environment. Other database systems, based on a “one size fits all” model, force developers to write code that is only understandable to those familiar with a rigid, proprietary API, and impose data storage that is entirely “on-disk” or “all-in-memory” when a blended approach would be ideal. With the *eXtremeDB* product family, embedded database management reaches new levels of performance, reliability and maintainability.

The Runtime Environment

eXtremeDB was designed for performance, starting with a memory-based architecture and direct data manipulation. Typical read and write accesses require a few microseconds, or less. The engine is re-entrant, allowing for multiple execution threads. Transactions support the ACID properties, assuring transaction and database integrity.

eXtremeDB Fusion, McObject’s hybrid database system, builds on the core architecture by offering two storage modes: on-disk and in-memory. Developers can combine these two approaches *in the same application*, providing a vital tool for fine-tuning a system’s performance and data persistence.

eXtremeDB’s product family offers proven tools to obtain data persistence at runtime. Transaction logging is supported, while *eXtremeDB* High Availability provides fail-safe database processing even in the face of hardware or software failure.



eXtremeDB	Advantage
Small footprint – 100K or less depending on processor and compiler	Fits the most resource-constrained environments
In-memory database – all data is stored in main memory	No disk overhead
Direct data access – application works with data in main memory	Minimizes CPU cycles
High transaction rate	Keeps up with demanding device environments
No translation – eXtremeDB stores data in the exact form used by the application	Eliminates overheads from translating to relational representations
Data integrity – transactions support the ACID properties	Reliable implementations
Re-entrant engine	High performance even when using multiple execution threads
Choice of database interfaces – native or SQL APIs	Flexibility and performance optimization
Optional on-disk storage via eXtremeDB Fusion	Code fine-tuning for performance and persistence

The Development Environment

Developers strive to produce readable, maintainable, efficient code in the shortest time. *eXtremeDB* includes numerous features that boost development efficiency.

eXtremeDB offers a choice of database interfaces. Its efficient native API derives from the application's data model. Because functions reflect the database's purpose and schema, the compiler can catch data typing and assignment errors, resulting in more reliable run-time code. The second API is *eXtremeSQL*, a high-performance implementation of the SQL database programming language for *eXtremeDB*.

To help in debugging, *eXtremeDB* includes numerous traps in the runtime code, which can be disabled as development and testing progress. McObject provides source code, to give an in-depth understanding of *eXtremeDB* within an application. In addition, *eXtremeDB* supports virtually all data types as well as extremely efficient indexing for queries. These include hash indexes for exact match searches; tree indexes for pattern match, range retrieval and sorting; Patricia Trie indexes for network/telecom applications; r-tree indexes for geospatial lookups; and object-identifier references, for direct access. Rather than storing duplicate data, indexes contain only a reference to data, keeping memory requirements to an absolute minimum.

Intuitive Native API

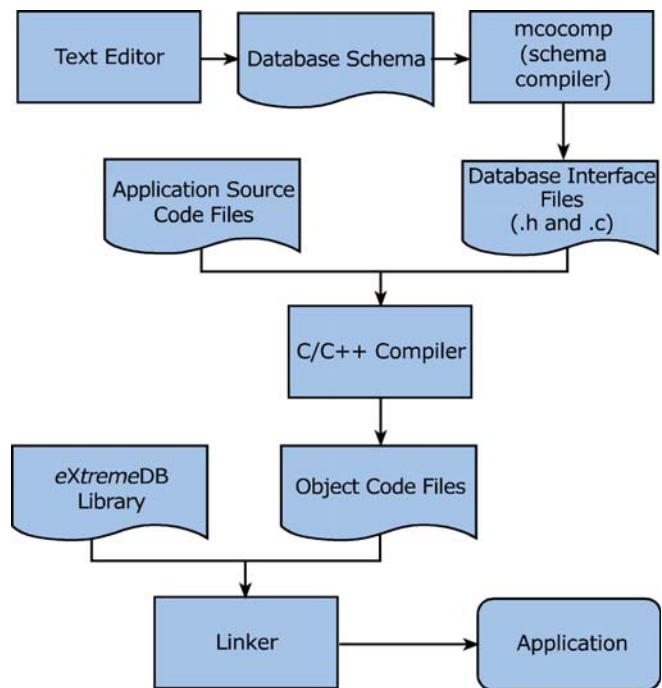
With *eXtremeDB*'s native API, the developer focuses on the data definition first, then *eXtremeDB* generates the API from this definition via the schema compiler. The result is

- An easy-to-learn API that is optimized for the application.
- Code that is more legible as well as easier to write and maintain.
- Compile-time type-checking that helps eliminate coding errors.

Example:

The following is a (simple) class and an example of the API to put a new value into a record in the database:

```
class Measurement{
    string measure;
    time timestamp;
    unique tree <measure, timestamp> trend;
};
Measurement_measure_put (&m, meas);
Measurement_timestamp_put (&m, value);
```



"*eXtremeDB* simplifies development and testing, especially in situations where the database must coordinate multiple processes."

- *Tyco Thermal Controls*

Progressive error detection and consistency checking features

If an application passes an invalid transaction or object handle into a runtime method, *eXtremeDB* (by default) raises a fatal exception and stops execution. In most cases, the developer can examine the call stack for the source of the coding error. The *eXtremeDB* runtime implements many verification traps and consistency checks. After application debugging, the optimized version of the *eXtremeDB* runtime is obtained by removing traps and internal checks to restore valuable clock cycles.

eXtremeDB Product Family

■ eXtremeDB Standard Edition

Ultra-fast in-memory database with a high-level data definition language, concurrent access, transactions, XML support, flexible indexing and more.

■ eXtremeDB High Availability Edition

Based on a rugged, time-cognizant two-phase commit protocol, eXtremeDB -HA enables multiple fully synchronized database instances, with automatic failover.

■ eXtremeDB Transaction Logging Edition

Adds durability to eXtremeDB in-memory databases with transaction logging, which enables recovery via a journal of database changes.

■ eXtremeSQL and ODBC

A high-performance implementation of the SQL database programming language for eXtremeDB.

■ eXtremeDB-64

The 64-bit version of eXtremeDB supports databases that are hundreds of times larger, for instant sorting and manipulation of massive data stores.

■ eXtremeDB Fusion

Combines the strengths of on-disk and all-in-memory storage in a single system. Optimize for speed and persistence, while exploiting the most cost-effective and physical space-conserving data storage.

Complex data types and efficient queries

- Supports virtually all data types, including structures, arrays, vectors and BLOBs
- Querying methods include hash indexes for exact match searches
- Tree indexes support queries for pattern match, range retrieval and sorting
- "Voluntary" indexes for program control over index population
- R-Tree and Patricia trie indexes
- Object-identifier references provide direct data access
- Autoid for system-defined object identifiers
- Rather than store duplicate data, indexes contain only a reference to data, minimizing RAM demands

"eXtremeDB gave us the performance we required and the flexibility to manage the complex data inherent in our application."

- Genesis Microchip

Target platforms

- Linux, QNX Neutrino, LynxOS, uC/OS-II
- Nucleus, INTEGRITY, eCos, uCLinux
- Windows Embedded Platforms
- VxWorks, Quadros RTXC,
- Solaris, HP-UX, Windows 98/Me/NT/2000/XP/Vista
- eXtremeDB source code for all platforms

32-bit Database specifications

- Maximum objects per database: 2,147,483,647
- Maximum classes per database: 32,767
- Maximum indexes per database: 32,767
- Maximum fields or vectors per class: 32,767
- Maximum fields per index: 32,767
- Maximum elements per vector: 32,767
- Memory requirements: As little as 50K
- Maximum databases open simultaneously: 16
- Maximum simultaneous connections per database: 64

Supported data types

- 1, 2, 4, 8-byte signed/unsigned integers
- enum
- float, double
- date, time
- char (fixed length), string (variable length)
- fixed-size array
- variable-length vector
- structs (nested to any depth)
- autoid (auto-increment)
- BLOB
- user-defined object-id and references