

Embedding ObjectStore PSE Pro for Java, The Pure Java, Pure Object Database



14 Oak Park
Bedford, MA 01730 USA
Phone: +1-781-280-4000
www.objectstore.net

INTRODUCTION	3
ENABLING SOFTWARE INNOVATION WITH JAVA	4
Today's Java Database Applications	4
THE NEED FOR AN EMBEDDED JAVA DATABASE	5
Embedded Java Database Requirements	5
Traditional Database Shortcomings	6
OBJECTSTORE PSE PRO: THE PURE JAVA, PURE OBJECT DATABASE	7
Most Portable DBMS - 100% Pure Java™	7
Best Support for Java	7
Fastest DBMS	8
Most Compact DBMS	9
Most Extensible DBMS	9
Zero Database Administration	9
Part of the Most Comprehensive Object DBMS Product Family	9
EMBEDDING OBJECTSTORE PSE PRO IN JAVA SOFTWARE	9
Creating ObjectStore PSE Pro Schema	10
Creating Transactions	10
Persistence by Reachability	11
Defining and Running Queries	12
Using Indexes	13
Using Persistent Garbage Collection	13
CONCLUSION	14

Introduction

Java is profoundly affecting the computing industry. The Write Once, Run Anywhere premise of the technology allows software manufacturers to dramatically reduce time to market and development costs by reallocating software porting resources into new product development. In addition, the ubiquity of the Java Virtual Machine allows companies to deploy innovative new software solutions across the entire computing spectrum: servers, desktops, thin clients, and even devices such as hand-held computers, appliances, and medical and communications equipment and so on.

In order to fully realize the competitive edge that Java provides, companies require a new database management system (DBMS) that does not limit innovation, time to market, or the ability to deploy Java applications to any platform.

The Java database on which you standardize must:

- Run anywhere Java can
- Install and execute without administration assistance
- Require as little RAM as possible—especially for applications deployed to thin clients and devices
- Provide superior performance, extensibility and time to market advantages

There is only one DBMS that meets all these needs. ObjectStore® PSE Pro, the Pure Java, Pure Object Database, is uniquely designed to meet Java developers' requirements for an embedded DBMS.

This white paper is written for developers and IT project managers, and it describes the advantages of the ObjectStore PSE Pro Pure Java, Pure Object database architecture. The white paper answers questions including:

- What DBMS options are available to Java developers?
- How does ObjectStore PSE Pro compare with other databases in terms of:
 - Portability
 - Administration requirements
 - Java support
 - Extensibility
 - Footprint
 - Performance

For a "hands-on" experience with ObjectStore PSE Pro or to review the product documentation, please visit www.objectstore.net and download a free trial version of ObjectStore PSE Pro for Java.

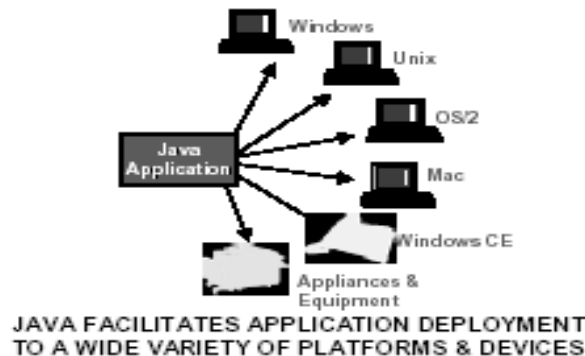
*Studies have shown that compared with relational databases, ObjectStore PSE Pro-based Java software requires **35% percent less source code and performs hundreds of times faster.****

*Refer to Java Data Management—Comparing ODBMS and RDBMS implementations, written by STR, November, 1997.

Enabling Software Innovation with Java

Software developers are flocking to Java, and arguably it has been more rapidly adopted than any other application development technology in recent history. Why is that?

- **Faster time to market.** Java's ubiquity and portability allow you to eliminate software porting costs and invest in new product development.
- **Lower maintenance costs and increased responsiveness.** Java is object-oriented, and its extensibility and framework for code reuse allow you to adapt your software to changing business, technical and competitive requirements faster and at lower cost.
- **Increased market share and revenue.** Your Java applications can be used by any customer, regardless of their computing platform. "You don't support my platform" becomes a sales objection of the past.
- **Easy entry to the embedded application market.** Java virtual machines are being embedded in a wide variety of devices, which makes it easy for you to create new applications that run in appliances, hand-held computers, Windows CE devices, medical & communications equipment and others.



Today's Java Database Applications

Database application developers are leveraging Java's benefits to create a wide variety of portable new systems:

- **Embedded Systems** The ease and familiarity of Java development combined with the availability of Java on a growing number of devices is fostering a new wave of embedded database applications. For example, Java databases store network configuration information inside of telephone switches; they store readings and bioinformatics inside of medical testing equipment, and personal viewing and shopping preferences inside of interactive television devices.
- **Mobile Computing** Java is ideal for mobile computing. Platform-independent applets and applications can be deployed to local or remote users automatically through the Internet. Mobile Java clients such as order entry and scientific research applications run on- or off-line

in Web browsers or Java virtual machines and update local databases. When reconnected to the network, they exchange data with servers through Java RMI, JavaSpaces and other means.

- **Application Servers.** In addition to devices and mobile clients, Java's tight integration with server technologies like Web Servers, ORBs, database servers and others makes it a natural choice for a wide variety of new server-side ecommerce, on-line publishing, intranet and extranet applications.

As organizations shift to Java in order to build innovative, strategic systems like these, important data management issues arise.

The Need for an Embedded Java Database

Embedded Java software, mobile Java applications, portable Java servlets all share a common requirement: **access to a portable, local database.**

Devices and mobile applications require an embedded database because they're only occasionally connected to a database server, and application servers require a local database because connecting to a remote database over a network limits performance and scalability.

Embedded Java Database Requirements

In order to deliver such a wide range of applications and to take full advantage of Java's benefits, a new database standard is required. Your Java database must:

Run anywhere Java does. What's the benefit of writing a Java database application if your DBMS only runs on Windows? In order to benefit from Java's portability and deliver your application to customers regardless of their computing platform, you require a database that's equally as portable as your Java application code.

Run in a small footprint. Devices, thin clients and many mobile computing platforms are only equipped with a few megabytes of RAM and limited disk space and processing power. A palmtop computer cannot accommodate a DBMS with a 1MB footprint and still have room for the operating system and application. A database with a footprint under 500K is required, and the DBMS must keep the size of the data itself compact.

Require no administration assistance. In many cases, your target platform, especially devices, lacks an interface for administering databases, or your target users are unable to install and configure a database, or you simply don't want to incur database administration expenses for your local and remote users. These situations call for a database that automatically deploys, installs and executes--with no administration overhead.

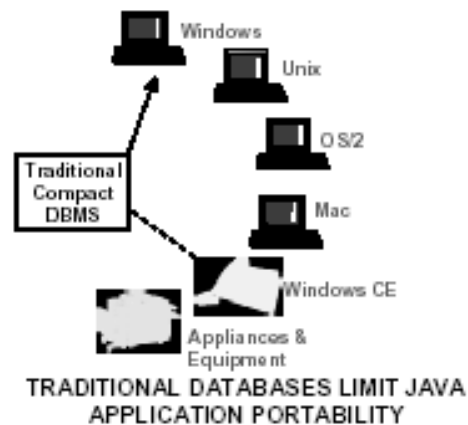
Provide superior performance and time to market. In today's competitive environment, time to market is key to establishing leadership and revenue share. Product quality is important in preserving market leadership. Your Java database must be easy to use, and deliver superior performance, or you may fail to meet your time to market and product quality goals.

These are the requirements against which you must measure your new Java database standard. Traditionally, developers have turned to Windows-based relational databases to meet requirements like these, but Java has made their utility obsolete.

Traditional Database Shortcomings

Traditional compact relational and hybrid object-relational DBMSs do not meet the embedded Java database requirements.

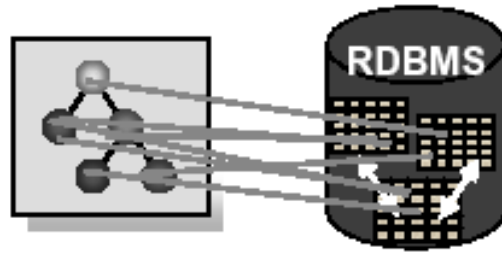
Limited portability. If your Java database is not 100% Pure Java™, then it's not portable enough for you. Traditional DBMS vendors like Sybase and Oracle and others may offer 100% Pure Java™ JDBC drivers, but their database engines only run under Windows. They may limit your ability to distribute your Java product to Unix, OS/2, and Macintosh users, which results in lost revenue potential.



Require too many system resources. On the small end of the scale, traditional DBMS footprints are in the 1MB range, and some can scale under 500K. However, the footprint issue isn't limited to the DBMS RAM requirements. Database size and the amount of code required to integrate your DBMS and Java also affect an application's ability to run on embedded devices. For example, relational and especially object-relational DBMS tend to inflate the size of your databases with duplicate data (indexes, foreign keys, etc.). In addition, you generally have to write 35% to 60% more code to convert your Java data from objects to rows (and back), which inflates the size of your application executable.

Too expensive to administer. Developers benefit from Java's application and applet packaging and deployment facilities. Traditional databases weren't designed for this paradigm and require extra development of special installation, database setup and application start-up scripts.

Poor time to market and poor performance. Java applications manage objects, and relational and object-relational DBMS manage rows. The negative impact of this mismatch is slower time to market and performance.



TRADITIONAL DATABASES ARE NOT
DESIGNED TO MANAGE JAVA OBJECT DATA

Your application requires nearly 50% more Java code just to map your Java objects into relational tables using JDBC and SQL, which results in increased development and maintenance costs and lost income and market share. Waiting for the object-to-row translation to take place, combined with the relational joins required to re-establish relationships between objects at runtime imposes significant performance limitations on the application, which may make it impossible to deploy your Java application in real-time testing and control equipment.

The fundamentally new characteristics of today's Java applications suggest that for maximum deployability, performance, and productivity, a new embedded DBMS technology is required.

ObjectStore PSE Pro: The Pure Java, Pure Object Database

ObjectStore PSE Pro is the only database designed specifically for Java developers who require an embedded database.

The ObjectStore PSE Pro Pure Java, Pure Object approach is unique and results in tremendous deployability, productivity and performance benefits across all Java-supported platforms.

Here are the key architectural features and benefits of ObjectStore PSE Pro:

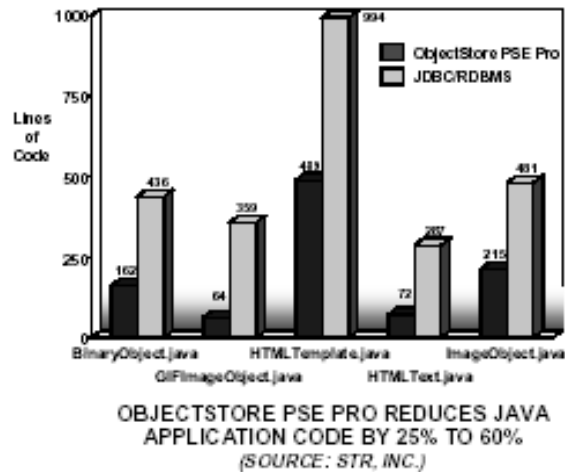
Most Portable DBMS - 100% Pure Java™

In 1996, ObjectStore PSE Pro became the first Java DBMS on the market, and in 1997, it became the first DBMS to pass Sun Microsystem's 100% Pure Java™ certification tests. As such, ObjectStore PSE Pro runs anywhere Java does, and customers use it under Windows, Unix, OS/2, Macintosh, Windows CE, and it also meets PersonalJava compatibility tests. With ObjectStore PSE Pro for Java, you can be sure that your Java application can be distributed, along with its database, to any Java platform. Today, ObjectStore PSE Pro is embedded in many other 100% Pure Java™ products including Sun's own JavaSpaces distributed Java platform.

Best Support for Java

As a true object-oriented database management system (ODBMS), ObjectStore PSE Pro manages data as objects and eliminates the mismatch between object and relational data models and the code needed to map between them each time data is stored and retrieved. Additionally, ObjectStore PSE Pro seamlessly integrates with Java so, unlike RDBMSs, there is no second language to learn for data definition and manipulation: no JDBC, no SQL. The benefits of this approach are enormous: typically 25% to 60% of the application code in a Java application using an RDBMS is unproductive mapping code.

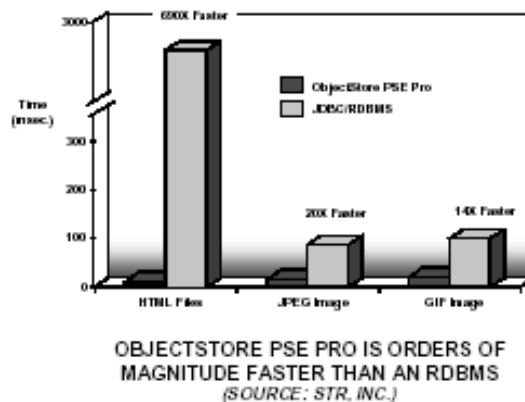
Eliminating this code reduces time to market and increases quality, performance, and maintainability. The next section, Programming with ObjectStore PSE Pro, illustrates how seamlessly the database integrates with Java.



Fastest DBMS

Java programs represent data as hierarchies of interrelated objects. Most databases deconstruct objects into tables for storage, and in the process lose the explicit relationships between objects. When data is accessed, relationships must be reconstructed by executing CPU-intensive SQL joins. Any developer who has worked with an RDBMS knows that performance degrades exponentially as the number of tables involved in a join increases.

In contrast, ObjectStore PSE Pro stores objects in the database directly, with their relationships intact. As a result, ObjectStore PSE Pro is able to query object data by navigating the relationships between objects instead of doing relational joins with SQL queries, which allows ObjectStore PSE Pro to run orders of magnitude faster than other databases.



Most Compact DBMS

The ObjectStore PSE Pro DBMS footprint is a mere 300K. Unlike other databases, ObjectStore PSE Pro stores your Java data to disk as objects, with inter-object references and pointers intact. There are no tables, no joins, no data duplication required to create foreign keys, nor is there any mapping code in your Java software required to convert objects to relational rows. Less data and less source code mean **smaller databases** (25% to 50% smaller than relational databases, on average) and **smaller application executables**.

Most Extensible DBMS

RDBMSs are designed to manage simple tabular data such as numbers, dates, and text. ObjectStore PSE Pro, by contrast, has been designed from the start to support any abstract Java data type. ObjectStore PSE Pro schema and data are defined by your Java class hierarchies, and extending ObjectStore PSE Pro's support for data types is as easy as creating or extending your Java object models using techniques like inheritance, composition, and aggregation.

An additional advantage to ObjectStore's seamless Java support and extensibility is that ObjectStore PSE Pro can store data defined by popular third-party class libraries such as ObjectSpace's JGL and others without extensive modification. Consequently, ObjectStore PSE Pro is a natural fit for applications that process the rich data types being used today, including multimedia, JDK collections, time series, spatial data, and many others.

Zero Database Administration

ObjectStore PSE Pro is designed from the ground up to be embedded in your application or device. The database itself is a modular group of Java classes that download, install and execute automatically on an as-needed basis just like the rest of your Java applet, application or servlet. No user intervention is ever required to administer ObjectStore PSE Pro, which simplifies your product and reduces your support costs.

Part of the Most Comprehensive Object DBMS Product Family

Application migration between devices, desktops and enterprise servers is a key concern for Java software developers. ObjectStore PSE Pro is a member of the most complete object data management product family on the market. ObjectStore PSE Pro and the enterprise edition of the ObjectStore ODBMS share a common API, which allows you to scale your Java software from devices and thin clients all the way up to enterprise-class servers--without changing your code.

In summary, the benefits of the ObjectStore PSE Pro Pure Java, Pure Object database architecture over both relational and object-relational databases make ObjectStore PSE Pro ***the fastest, most portable, and compact Java database on the market.***

Embedding ObjectStore PSE Pro in Java Software

ObjectStore PSE Pro provides the most comprehensive support for Java. Unlike other databases, ObjectStore PSE Pro doesn't require extensive modification of your Java application class hierarchy to permit objects to be stored. This transparency results in more rapid development, easier extensibility, faster migration of existing code, and higher performance at run time than other databases.

Using ObjectStore PSE Pro is simple. The following sections illustrate ObjectStore PSE Pro's seamless Java support and, in some cases, contrast it with JDBC:

- Creating ObjectStore PSE Pro Schema
- Creating Transactions
- Persistence by Reachability
- Defining & running queries
- Using indexes
- Using persistent garbage collection

Creating ObjectStore PSE Pro Schema

ObjectStore PSE Pro can store any type of object you define in Java because your Java class definitions define the structure of what the database will store. Unlike other databases, there's no need to use a second tool or language, like SQL, to define your data. Eliminating redundant data definition procedures saves you time and money.

Here's how you would prepare ObjectStore PSE Pro to store Employee objects, which contain relationships to another type of object called Manager.

```
public final class Employee {
    String name;
    int salary;
    String location;
    Manager boss;

    public Employee(String name, int salary,
        String location, Manager boss)
    {
        name = name;
        salary = salary;
        location = location;
        boss = boss;
    }
}
```

Assuming that this class is stored in a file called Employee.java, here's how to make ObjectStore PSE Pro ready to store new Employee objects or collections or arrays of Employee objects. Simply compile the class and run the following command line utility against the resulting .class file or against a batch of classes in a .zip or .jar file:

```
osjcfp -inplace Employee.class
```

The Osjcfp (ObjectStore Java Class File Postprocessor) utility annotates your Java class files with instructions to ObjectStore PSE Pro on how to store and access Employee objects. These annotations allow you to make your Java objects persistent seamlessly--without writing mapping code and without forcing you to alter your object model with vendor-supplied Java access classes.

Creating Transactions

Transactions are used to update and read ObjectStore PSE Pro data. They are fully recoverable and preserve data integrity in the event of system failure.

Here is an example of a transaction that updates an ObjectStore PSE Pro database with new Employee objects:

```
Database db = Database.open("employees.odb",
ObjectStore.UPDATE);
Transaction t = Transaction.begin(ObjectStore.UPDATE);

/* create a Set to hold all of the workers; make it a root in the
database */
Set allWorkers = new OSTreeSet(db);
db.createRoot("AllWorkers", allWorkers);

/* create some employees and add them to the set */
Manager boss = new Manager("Lauren",9000,"IRE");
allWorkers.add(new Employee("Andy",5000,"USA", boss));
allWorkers.add(new Employee("Pat",1500,"CAN", boss));

t.commit();
/*Pat & Andy were added to the set of all workers and stored
in the DBMS
Their related Manager object, boss, was also stored
automatically*/
```

In this example, the database is opened with update privileges, and an update transaction is begun.

The ObjectStore PSE Pro transaction model permits:

- Multiple concurrent read transactions (initiated by different threads or processes)
- A single update transaction (locks out all other access to the database until the transaction ends)

Persistent objects, including collections (as in the example above) are created, deleted, and updated, just as if they only existed in memory. There's no SQL, there's no need to write code to convert object data to tabular columns; it's completely seamless, which simplifies your development task.

After the collection called `allWorkers` is updated with new employee objects, the collection is stored in the database as a "root" object under a name ("AllWorkers") by which it can be retrieved later. Root objects can be individual objects or collections of objects. Finally, the changes are stored and the transaction is closed. Notice that with ObjectStore PSE Pro, explicit update instructions were not required for related objects. The `allWorkers` collection and `Employee` and `Manager` objects were all stored with a single command, which leads us to *persistence by reachability*.

Persistence by Reachability

Java applications comprise many interrelated objects, and the work required to explicitly store every different object type to the database can be tedious. ObjectStore PSE Pro supports the unique concept of "persistence by reachability," which automatically ensures the storage of all objects related to a persistent object (for example, a customer object's related salesperson and order objects).

Persistence by reachability dramatically reduces the amount of code you have to write and maintain. Compare the example above where one "createRoot" command resulted in the

automatic storage of a JDK Set object and its two related Employee objects and their related Manager object, with the JDBC equivalent:

```

Db = DriverManager.getConnection(
    "jdbc:odbc:LocalServer;database=Employee",
    "User1","");
db.setAutoCommit;

Manager boss = new Manager("Laureen",9000, "IRE");
statement=con.prepareStatement("INSERT INTO " +
"EMPLOYEE (emp_name, emp_salary, emp_location) " +
"VALUES (?, ?, ?)");
statement.setString(1,boss.name);
statement.setInt(2,boss.salary);
statement.setString(3,boss.location);
statement.executeUpdate();

Employee andy = new Employee("Andy",5000,"USA", boss);
statement=con.prepareStatement("INSERT INTO " +
"EMPLOYEE (emp_name, emp_salary, emp_location, " +
"emp_manager) VALUES (?, ?, ?, ?)");
statement.setString(1,andy.name);
statement.setInt(2,andy.salary);
statement.setString(3,andy.location);
statement.setString(4,andy.manager.name);
statement.executeUpdate();

Employee pat = new Employee("Pat",1500,"CAN", boss);
statement=con.prepareStatement("INSERT INTO " +
"EMPLOYEE (emp_name, emp_salary, emp_location, " +
"emp_manager) VALUES (?, ?, ?, ?)");
statement.setString(1,pat.name);
statement.setInt(2,pat.salary);
statement.setString(3,pat.location);
statement.setString(4,pat.manager.name);
statement.executeUpdate();

```

WITH OBJECTSTORE PSE PRO, YOU CAN REPLACE
THIS JDBC WITH 3 LINES OF CODE

Defining and Running Queries

Now you have an ObjectStore PSE Pro database that is populated with Employee and Manager objects. How do you search the database? With ObjectStore PSE Pro, there's no need to use SQL or JDBC. You simply use the Object Data Management Group (ODMG)-compliant ObjectStore PSE Pro query interface and native Java arithmetic, conditional and relational operators to compare object data and function variables.

The following example searches the Employee database for wealthy Employees:

```
// Search for wealthy US employees
Transaction t = Transaction.begin(ObjectStore.READONLY);
Set allWorkers = (Set) db.getRoot("AllWorkers");
Query myQuery = new Query(Employee.class,
    "salary >= 100000 && location = 'USA'");
Set wealthyOnes = myQuery.select(allWorkers);
Iterator iter = wealthyOnes.iterator();
while (iter.hasNext()) {
    Employee e = (Employee) iter.next();
    System.out.println (e.name + " earns: " + e.salary + "
        manager = " + e.manager.name);
}
t.commit();
```

In this example, a read-only transaction is started (it can be processed with other read-only transactions, concurrently), and the collection of Employees is retrieved from the database by its root, "AllWorkers." Next, a query object is defined using Employee class fields and native Java operators. The query is executed against the allWorkers collection and the result set is automatically stored in a new collection object called, wealthyOnes. Finally, the application uses standard JDK 1.2 collection iterators to print data from each Employee object in the wealthyOnes set using Employee field names or accessor methods instead of result set column numbers as you do with JDBC.

Using Indexes

Indexes accelerate queries by providing them with optimized access to data. ObjectStore PSE Pro supports ordered and unordered indexes on both fields and methods. Indexes are maintained using a simple API—here's an example of how an index can be used to improve the performance of our Employee example:

```
Transaction t = Transaction.begin(ObjectStore.UPDATE);
Set allWorkers = (Set) db.getRoot("AllWorkers");

allWorkers.addIndex(Employee.class, "salary", true, true);

Set wealthyOnes = (Set) myQuery.select(allWorkers);

t.commit();
```

In order to accelerate searches on Employee salary data, an index to the allWorkers collection is defined using the ObjectStore PSE Pro addIndex method. Parameters include the name of the class, the data to index and Boolean values that indicate whether the index is sorted and whether duplicate data can be stored in the index. Queries can be instructed to use the new index, which allows ObjectStore PSE Pro to quickly identify all the Employee objects that meet the query criteria without having to read each Employee object from the database.

Using Persistent Garbage Collection

Java's automatic storage management features are one of its key advantages over a language such as C++. Java removes the burden of managing storage (allocating and removing objects), which eliminates one of the most difficult classes of bugs, dangling references, to track down.

The PSE Pro persistent garbage collector (GC) extends these benefits to persistent data. The PSE Pro persistent GC collects unreferenced Java objects in a PSE Pro database, which frees storage associated with objects that are unreachable. The result is less effort for you and more robust applications.

The PSE Pro persistent GC is independent of the Java VM GC, and it can be initiated programmatically through an API or as a utility from the command line. The persistent GC performs its job in two major phases. In the mark phase, the GC identifies the unreachable objects. In the sweep phase, the GC frees the storage used by the unreachable objects.

The PSE Pro GC operates on a database at a time. It locks the entire database, so applications cannot access the database while a GC is in progress. The Java application can improve performance by deferring object deletion and space reclamation to off hours.

ObjectStore PSE Pro is the only database designed for Java developers by Java developers. It offers the most comprehensive and seamless support for Java data types and programming constructs, which results in the highest productivity, performance benefits of any Java database.

Conclusion

Java is revolutionizing embedded software development with its ubiquity and its Write Once, Run Anywhere™ technology. The shift to Java is also resulting in demand for a new database standard that must:

- Run anywhere Java does
- Require little system overhead in order to run on devices and thin clients
- Install and execute without requiring any administration assistance
- Provide superior performance, data type support and time to market advantages

ObjectStore PSE Pro for Java, with its *Pure Java, Pure Object Database* architecture, uniquely meets the data management needs of Java developers building embedded applications. Its capability for high-performance data delivery, portability, seamless Java integration, and its 300K footprint allow organizations to deliver new applications to market more quickly and with higher quality than other DBMSs.

ObjectStore PSE Pro provides Java developers with a significant competitive edge, and you can get started with ObjectStore immediately by calling us or visiting our Web site and downloading a trial version of ObjectStore PSE Pro today!

ObjectStore is a registered trademark of Progress Software Corporation in the U.S. and other countries. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. Any other trademarks or service marks contained herein are the property of their respective owners.