



Version 7.0

db4o Replication System (dRS)

The *db4o Replication System (dRS)*, powered by [Hibernate](#), enables users to build applications that synchronize objects bi-directionally between distributed instances of [db4o](#)'s leading open source object database and all common relational databases such as [Oracle](#) or [MySQL](#). dRS makes [db4o](#)'s native object persistence architecture available to all [Java](#) and [.NET](#) developers while staying fully data compatible with existing IT environments using relational database technology.

db4o is the open source object database that enables Java and .NET developers to slash development time and costs and achieve unprecedented levels of performance.

The unique design of db4o's native object database engine makes it the ideal choice to be embedded in equipment and devices, in packaged software running on mobile or desktop platforms, or in real-time control systems – in short: in environments where no DBA is present.

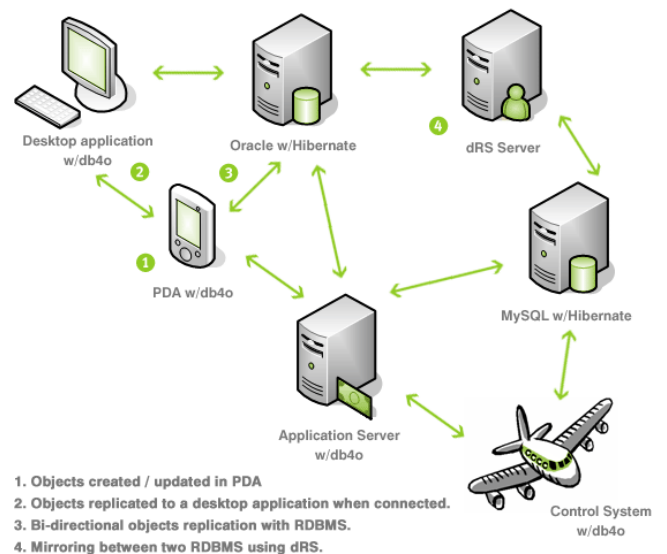
dRS is 100% object-oriented and is specifically designed for agile enterprises and manufacturers of software-enabled products in a fast changing and ever more mobile business environment. Software managers can shave off up to 90% of development cost and time in comparison to using replication through relational database technology which had been designed for server-centric data storage in the 1980s and is inherently incompatible to the object-oriented software architectures of Java and .NET.

[db4o](#) through [Hibernate](#) is superior to using [Hibernate](#) alone, since it enables the developer to fully design the software based on business logic, rather than by considerations and limitations of [Hibernate](#).

[db4o](#)'s object database is designed to be embedded in distributed applications and software-enabled devices, where resources are constrained, performance is critical, and no DBA is present. Examples of [db4o](#) customers' applications are [Indra's High Speed Train Control System](#), [Boeing's P8-A aircraft](#), or [Easterndata's mobile doorstep delivery system running on off-the-shelf PDAs](#). In many cases, these client-side [db4o](#) database instances are partially connected to server-side enterprise IT running relational databases (RDBMS) such as [Oracle](#) or [MySQL](#). The dRS provides automatic, uni- or bi-directional data synchronization between these distributed data sets upon connection.

db4o's Replication System (dRS) is available for [db4o-to-Hibernate/RDBMS](#), [db4o-to-db4o](#) and [Hibernate/RDBMS-to-Hibernate/RDBMS](#) replication. dRS version 7.0 runs on Java 1.5 or later and is able to fully replicate Java objects, while replicating .NET objects only between [db4o-to-db4o](#).

A free download under the GPL is available from the [db4o Download Center](#) (<http://developer.db4o.com/files/>), and an affordable commercial (non-GPL) license is available upon request at sales@db4o.com.



1. Objects created / updated in PDA
2. Objects replicated to a desktop application when connected.
3. Bi-directional objects replication with RDBMS.
4. Mirroring between two RDBMS using dRS.

← Bi-directional Replication →



Enabling the Agile and Mobile Enterprise

dRS's object-oriented replication approach is specifically designed for agile enterprises in fast changing and ever more mobile business environments. The dRS is optimized for easy synchronization of distributed database instances AND frequent refactoring of software code, which in the past have been difficult if not impossible to achieve at the same time.

As an example, the business object "customer" (containing customer account information) would need synchronization between a salesperson's partially connected PDA and the backend enterprise server. Parent-child relationships in classes provide a natural description of what makes up the object "customer" and what needs to be replicated along with a parent object. To accomplish a complete synchronization of this business object, the developer just needs to connect the two databases instances to query for updated "customer" objects and then synchronize them -- with just one line of code:

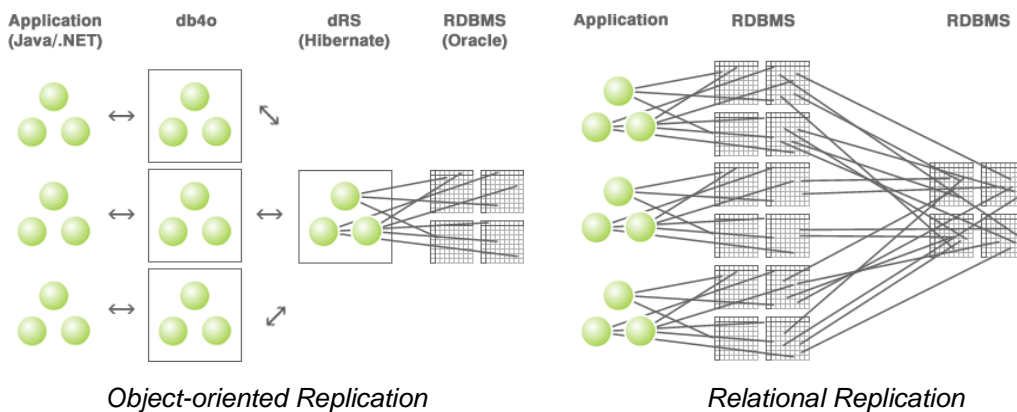
```
//Start a new ReplicationSession
ReplicationSession replication = HibernateReplication
    .begin(db4oProvider, hibernateProvider);

//Query for objects changed from db4o (Provider A)
ObjectSet changedObjects = replication.providerA()
    .objectsChangedSinceLastReplication();

//Iterate changed objects, replicate them
while (changedObjects.hasNext()) {
    Customer customer = (Customer) changedObjects.next();
    replication.replicate(customer);
}

//commit the replication session
replication.commit();
```

In conventional, relational technology, the business object "customer" could comprise many dozens of relational tables which all would need to be referenced by hard-wired, static application code, consisting of many lines including non-native SQL strings that make refactoring even more cumbersome.



While it has been argued (ref. Scott Ambler, "[Agile Techniques for Object Databases](#)") that the object-relational impedance mismatch poses a huge constraint on the ability to refactor software by itself, this problem is further aggravated by orders of magnitude once distributed database instances have to be held in synchronization.

Using db4o's native object-oriented persistence approach, developers are no longer constrained by their persistence solution to increase productivity through agile development. Thanks to its automatic schema evolution, db4o automatically adapts to any updated object model and hence enables more frequent refactoring of software even within highly distributed data architectures.

As a result, the business can adapt more easily to an ever faster changing environment without seeing IT costs explode and software quality and maintainability decrease. Instead of a "Never change a running system" policy, businesses deploying db4objects' native object persistence solutions are able to stay ahead of competition by meeting emerging customer demands faster and provide more feature rich, differentiating software solutions, especially for mobile applications.

Unique: Object-oriented Synchronization Conflict Resolution

One of the unique features of the dRS is its application-data-driven conflict resolution. The application data model, which should be the sole repository for data-oriented business logic, can be used to solve conflict resolution, rather than relying on secondary application code repositories.

Synchronization conflicts occur when an object is simultaneously modified in disconnected database instances since their last replication session. The user may then choose to model business rules that automatically resolve these conflicts by overwriting changes from one side to another, skipping the conflicted object, stopping replication to avoid data corruption and/or embedding a notification routine to trigger further user action or other business rules.

```
ReplicationEventListener listener = new ReplicationEventListener() {  
    public void onReplicate(ReplicationEvent event) {  
        if (event.isConflict()) {  
            //Override the record with PDA's copy (Provider A)  
            event.overrideWith(event.stateInProviderA());  
        }  
    }  
};
```

As an example, the business object "customer" could have a modified telephone number in both the salesperson's partially offline PDA and the back office CRM database. Business logic could demand that the salesperson has "ownership" of the customer and could overwrite all backoffice changes, unless certain cases apply (e.g., certain rights for the backoffice users or a certain type of transaction such as a user-self-service transaction, which could be "superior" to the salesperson's ownership). This business logic should not be hard-wired (as in relational replication technology) in the application itself, but be stored together with the object to make it "smart" enough to automatically synchronize itself respecting all relevant and most recent business rules.



Setup

The setup of the *db4o Replication System* for operation with relational databases is straightforward. Simply download the db4o core and the *db4o Replication System* including Hibernate (if not already present) from the db4o Download Center (<http://developer.db4o.com/files/>).

Install and configure Hibernate to connect to your relational database, and create Hibernate mapping files for persistent classes. If you already have a relational database with data, dRS will update the schema to keep track of the version of objects. If your database is blank, the System will create tables for storing the objects automatically.

Next, start a replication session. Pass in the objects that you want to replicate. Then commit -- the two databases are now in sync. For more details, please refer to the bundled user guide in the distribution and to the many examples that will get you up and running in less than 10 minutes.