Welcome

This documentation introduces the db4o Replication System for .NET (dRS.NET).

dRS.NET provides replication functionality to periodically synchronize databases that work disconnected from each other, such as remote autonomous servers or handheld devices synchronizing with central servers.

Before you start, please make sure that you have downloaded the latest dRS.NET distribution from the download area of the db4objects website.

We invite you to join the db4o community in the public db4o forums to ask for help at any time. You may also find the db4o knowledgebase helpful for keyword searches.

1. Roadmap

This section describes the features supported in current release and features planned in upcoming releases.

1.1. Supported Features

- db4o to db4o replication

- Array Replication
- Inheritance hierarchies
- Standard Primitive Types
- One-To-One Relations
- One-To-Many Relations
- Many-To-One Relations

- Queries for changed objects
- Parent-to-Child traversal along changed objects
- Collection Replication
- Replication Event System
- Replication of deleted objects

1.2. Planned Features

- NHibernate replication
- Instant replication

2. Requirements

dRS.NET is provided as source code. In order to compile you will need load the solution into Visual Studio 2005.

dRS.NET is dependant on the .NET 2.0.

Testing the installation
The drs.tests project contains unit tests for testing dRS.NET.

3. First Steps

To get started as simply as possible, we are going to reuse the Pilot class from the db4o tutorial.

```
public class Pilot {
    string name;
    int points;
}
```

3.1. Simple Replication

First, we will also use db4o to db4o replication, as it is the simplest and requires the least initial setup.

When replicating objects to and from a db4o database, we need to enable UUIDs and VersionNumbers. The chapter on db4o replication explains more about these settings. We need to call these settings before opening our database.

```
//Replication requires UUIDs and VersionNumbers
Db4o.Configure().GenerateUUIDs(int.MaxValue);
Db4o.Configure().GenerateVersionNumbers(int.MaxValue);
```

Open the source db4o database

```
ObjectContainer handheld = Db4o.OpenFile("handheld.yap");
```

And let's store an object or two:

```
source.Set(new Pilot("Scott Felton", 200));
source.Set(new Pilot("Frank Green", 120));
```

Great. Now we need our destination database:

```
//Open the destination db4o database
ObjectContainer desktop = Db4o.OpenFile("desktop.yap");
```

Now, it's about time we start replicating some data. Let's set up our replication process to replicate some data from our handheld, to our desktop:

```
ReplicationSession replication = Replication.Begin(handheld,
desktop);
```

The ReplicationSession object contains all the functionality that the replication system requires to be able to comprehensively replicate modified data from one database to another.

Now that we've got the ReplicationSession set up and ready, let's get to the meat of this process:

```
//Query for objects changed from db4o
ObjectSet changed =
replication.ProviderA().objectsChangedSinceLastReplication();

//Iterate changed objects, replicate them
while (changed.HasNext())
    replication.Replicate(changed.Next());
```

This code will get a list of all of the updated objects, and then replicate them to the destination database. Easy!

But wait, there's one important last step:

```
//commit all of our changes to both databases.
replication.Commit();
```

Ok. NOW we're done. Now the data has been copied, and the transaction is commited. All of our changes are safely commited in case of any failures.

3.2. Bi-Directional Replication

Our previous example copied all new or modified objects from our handheld device to our desktop database. What if we want to go the other way? Well, we only have to add one more loop:

```
ObjectSet changed =
replication.ProviderB().ObjectsChangedSinceLastReplication();

while(changed.HasNext())
```

```
    replication.Replicate(changed.Next());
```

Now our handheld contains all of the new and modified records from our desktop.

Wow, wasn't that easy? Now, if there had been any modifications made to the destination datbase, the two are now both in sync with eachother. Congratulations, you've now syncronized 2 db4o databases!

3.3. Selective Replication

What if our handheld dosn't have enough memory to store a complete set of all of our data objects? Well, then we should check our objects before replicating them by modifing the while loop:

```
ObjectSet changed =
replication.ProviderB().ObjectsChangedSinceLastReplication();

while (changed.HasNext()){
    Pilot p = (Pilot)changed.Next();
    if(p.name.StartsWith("S"))
        replication.Replicate(p);
}
```

Now, only Pilots whose name starts with "S" will be replicated to our handheld database.

4. db4o Replication

In order to use replication, the following configuration settings have to be called before a database file is created or opened:

```
Db4o.configure().generateUUIDs(Integer.MAX_VALUE);
Db4o.configure().generateVersionNumbers(Integer.MAX_VALUE);
```

UUIDs are object IDs that are unique across all databases created with db4o. That is achieved by having the database's creation timestamp as part of its objects' UUIDs. The db4o UUID contains two parts. The first part, contains an object ID. The second part identifies the database that originally created this ID.

The replication system will use this version number to invisibly tell when an object was last replicated, and if any changes have been made to the object since it was last replicated. An object's version number indicates the last time an object was modified. It is the database version at the moment of the modification.

4.1. Simple Replication

Now suppose we have opened two ObjectContainers from two different databases called "handheld" and "desktop", that we want to replicate. This is how we do it:

```
ObjectContainer handheld = db4o.OpenFile("handheld.yap");
ObjectContainer desktop = db4o.OpenFile("desktop.yap");

ReplicationSession replication = Replication.begin(handheld,
desktop);

ObjectSet changed =
replication.providerA().objectsChangedSinceLastReplication();

while (changed.hasNext())
    replication.replicate(changed.next());

replication.commit();
handheld.close();
desktop.close();
```

We start by opening two ObjectContainers. The next line, creates the ReplicationSession. This object contains all of the replication-related logic.

After creating the session, there is an interesting line:

```
ObjectSet changed =
replication.providerA().objectsChangedSinceLastReplication();
```

This line of code will get the provider associated with the first of our sources (the handheld ObjectContainer in this case). Then it finds all of the objects that have been updated or created. The new/modified objects will be returned in an enumerable ObjectSet.

After that comes a simple loop where the resulting objects are replicated one at a time.

The replication.commit() call at the end is important. This line will save all of the changes we have made, and end any needed transactions. Forgetting to make this call will result in your replication changes being discarded when your application ends, or your ObjectContainers are closed.

The #commit() calls also mark all objects as replicated. Therefore, changed/new objects that are not replicated in this session will be be marked as replicated.

4.2. Replicating Existing Data Files

As we learned in the previous section, Db4o.configure().generateUUIDs() and Db4o.configure().generateVersionNumbers() (or its objectClass counterparts) must be called before storing any objects to a data file because db4o replication needs object versions and UUIDs to work. This implies that objects in existing data files stored without the correct settings can't be replicated.

Fortunately enabling replication for existing data files is a very simple process:
We just need to use the Defragment tool in com.db4o.tools (source code only) after enabling replication:

```
Db4o.configure().objectClass(Task.class).enableReplication(true);
new Defragment().run(currentFileName(), true);
```

After a successful defragmentation our data files are ready for replication.

5. Advanced Topics

In this chapter, we will look into several advanced replication features.

5.1. Events

Imagine you have just passed in an object that contains a huge List of objects. You only want to replicate the root object but not the list. What you can do is to make use of the event system of dRS.NET to stop traversal at the list.
Here is how:

First, you need to implement the ReplicationEventListener interface and pass it to Replication.

```
Sample Code - get a nice example tp put in here.
```

5.1.1. Resolving conflicts

The event system of dRS can also be used to resolve conflicts. When there is a conflict, You can choose to override the result with either the object from provider A, provider B or null. If you choose null, then the object will not be replicated.

```
Sample Code - get a nice example tp put in here.
```

5.2. Deleted Objects Replication

In addition to replicating changed/new objects, dRS.NET is able to replicate deletions of objects. When an object is deleted since last replication in provider A and you would like to replicate this changes to provider B, you can make use of the dRS.NET to handle that.

You could replicate deletions as follow:

```
replication.replicateDeletions(Car.class);
```

dRS.NET traverses every Car object in both providers. For instance, if a deletion is found in one provider, the deletion will be replicated to the other provider. During the traversal, ReplicationEvent will be generated as usual, you can listen to them. By default, the deletion will prevail. You can choose the counterpart of the deleted object to prevail if required.

Note that the deletions of a Parent will not be cascaded to child objects. For example, if a Car contains a child object, e.g. Pilot, Pilot will not be traversed and the deletions of Pilot will not be replicated.

6. License

6.1. dRS.NET

db4objects Inc. supplies the db4o Replication System for .NET (dRS.NET) under the General Public License (GPL).

Under the GPL dRS.NET is free to be used:
- for development,
- in-house as long as no deployment to third parties takes place,
- together with works that are placed under the GPL themselves.

You should have received a copy of the GPL in the file dRS.license.txt together with the dRS distribution.

6.2. Bundled 3rd Party Licenses

The dRS.NET distribution comes with several 3rd party libraries, located in the /lib/ subdirectory together with the respective license files.