Introduction

This document describe how to use the framework's components.

Embedded database use

Database creation

There are two ways to create a database. The first one uses the FastObjectDB.open(dbdirpath, dbname) method that returns an instance of the created db. If the db already exists, the method opens it instead of creating it. In order to facilitate the use of the db object reference we provide a singleton constructor FastObjectDBManager.getCurrentFODB() that uses the ResouceManager property fastobjectdb.database.path and fastobjectdb.database.name to create the db.

The dbdirpath or fastobjectdb.database.path indicates the path to the directory where FODB stores its files. One subdirectory is created per db name.

Db object collection creation

To store objects, an FODB collection must be created. A collection has to be created one time in the life cycle of the database. This means that collection creation informations are persistent inside the db, so collections are reloaded when the db is opened.

To see if a collection has been created, use the method
db.isCollectionExist(collectionName).

To create a collection use the method
db.createCollection(collectionName,ObjectClass.class);

CollectionName is the name of the collection and ObjectClass.class is the class of the object to be stored. All objects in the collection must inherit or be of the specified class.

Adding index to the collection

To retrieve object from the collection, index must be added. Index can index methods or fields of the stored objects. Indexed methods or fields must be of the type String, long, int. More index type can be added depending on the needs.

Index can be Unique or Multiple. Unique index is similar to primary key, one key per object. Multiple indexes can associate one key to several objects.

When indexes are created, unique index must be added first. To add an index, first an IndexDescriptor must be created to define index properties. Then the descriptor is added to the collection. This code creates a unique string index with 12 of btree order and 15 characters key length.
Btree order can vary from 10 to 20 depending on the collection size. It is preferable to increase it when the collection increases. 12 is a good order for collection containing less than 10 000 objects.

```
FODBStringIndexDescriptor KeyDescriptor =
newFODBStringIndexDescriptor
("KEY", FODBIndexDescriptor.UNIQUE, "getKey()", 12, 15);
db.addIndex("TESTDATA", KeyDescriptor);
```

For multiple indexes you have to specify the multiple index increment. Key index are store in an array of db object reference. If this array is too small to store all object references for one key, a new one is created increased by the multiple index increment. So to optimize the index size it is preferable to specify the average size of this array.

```
FODBStringIndexDescriptor debKeyDescriptor =
new FODBStringIndexDescriptor
("DEBKEY", FODBIndexDescriptor.MULTIPLE
, "getDebKey()", 10
,TestData.NB_BEGIN_KEY_LETTER, 5);
db.addIndex("TESTDATA", debKeyDescriptor);
```

Adding/replacing object to the FODB

To add an object to the database, use the method db.add (collectionname, object);
The object must not already exist if a unique index is associated to the collection. In this case an exception is thrown.
To replace an object use the db.replace(collectionname, object); instead.

Collection query

To retrieve object from a collection, FODB provides query facilities. First a query must be created with the method db.query() and a collection must be associated to the query. Query can only request objects inside one collection. To associate a collection use the method: q.constrain(object.class); .

Object.class is the object class associated to the collection at the creation time. Instead of the object.class, the name of the collection can be used. Then indexes that participate to the query are specified with operand and operator.
To execute the query use the q.execute(); which return an ObjectSet that contains all object found in the collection.

- Query examples:

Simple select query SELECT * FROM TESTDATA

```
Query q = db.query();
q.constrain(TestSearchData.class);
//specify an index to request.
Query subq = q.descend("getKey()");
//execute with no constrain to get all objects
ObjectSet set = q.execute();
```

- Other examples:

Query : SELECT * FROM TESTSEARCH where getKey<14

```
q = db.query();
q.constrain(TestSearchData.class);
subq = q.descend("getKey()");
subq.constrain(new Integer(14)).smaller();
set = q.execute();
```

Query: SELECT * FROM TESTSEARCH where getkey<16 OR getDizaine=2

```
q = db.query();
q.constrain(TestSearchData.class);
subq = q.descend("getKey()");
c = subq.constrain(new Integer(16)).smaller();
subq2 = subq.descend("getDizaine()");
d = subq2.constrain(new Integer(2)).equal();
c.or(d);
set = q.execute();
```

Delete object from collection

To delete an object from a collection use the method db.delete(collectionname, objecttodelete);.

The Delete method uses the first added unique index of the collection to retrievethe db object and delete it. If the collection doesn't have a unique index, objects can't be deleted. The method deleteWithId is the same but you specify the key of the first unique index added to the database instead of the object to delete.

Synchronize object collection

On the terminal side, FODB synchronisation doesn't need specific code. The synchronisation engine is integrated in the org.openmobileis.database.fastobjectdb.synchro.client.SynchroFastObjectDBManager class.
This class must be initialized with this call SynchroFastObjectDBManager.getCurrentFODB() before any call to FastObjectDBManager.
When an FODB collection is created, it's declared as synchronized. To modify this behaviour, use the FODBCollectionDescriptor class.
On the server side a synchro target connector must be implemeted to connect the synchronozation engine to the server database.
The connector must implements the class org.openmobileis.database.fastobjectdb.synchro.server.FODBSyncTarget To be use the connector has to be registered with the code FODBSynchroManager.getManager().registerCollection(new MyFODBSyncTarget());

There is three main synchronization types.

The first is the **imcremental** type that synchronize only modifications since the last synchronization.

The second is the **complete synchronization**. During this synchronization, terminal's modifications are synchronized and if everything is ok, the entire collection is synchronized in one file. We added the complete synchronization because we discover that in the case of too large server side modifications, it appears to be more efficient to synchronize the complete collection and override the collection file on the terminal in one operation rather than updating each modification on the terminal.
The getUpdateMaxNbRow() define the threshold of the number of modification that force complete synchronization. When the complete synchronisation starts the getAllCollectionObject() method is call to generate the collection file. The updateSynchroDB() is called to allow the connector to modify the synchronized collection before being send to the terminal.

The last type is the **pregenerated complete synchronisation**. This synchronization is usefull when the collection file can't be generated during the synchronization. The collection file is pregenerated and send during the synchronization. To register a pregenerated synchro target, use the FODBSynchroManager.getManager().registerSynchroTargetWrapper(new FODBOpenMSPSynchroTargetWrapper(MyFODBSyncTarget(), dbfilepath))
The org.openmobileis.database.fastobjectdb.db.test package provides an example of FODB synchronization (see the classes TestFODBSynchro and TestFODBSyncTarget).

Multithread management

FODB collection support concurrent collection access. FODB is locked at the collection level. The lock managment has two behavious :

- Single user mode : when a new transaction enter a locked collection, the previous transaction is stopped to allow the new transaction to access the collection. This mode is well suited for an application dedicated to one user at a time.
- Multi user mode : in this mode when a transaction enter a locked collection, it waits until the collection is unlocked.

The default mode is the Single user mode. To change this behaviour, set the property org.openmobileis.database.fastobjectdb.usermode in the PropertiesManager to false. PropertiesManager.getManager().setProperty("org.openmobileis.database.fastobjectdb.usermode", "false");