How to create one application

Author: Frédéric Brégier under LGPL

Quick list

Here is the first attempt to start what will be later on a long documentation on how to extend OpenLSD to fit business needs. The objective right now is not to make the documentation as complete as possible but to show the main ideas.

1) Identify the extension on database for the LSDDOCUMENT table
2) Extend by implementing the corresponding classes (see example with `BusinessImpl:openlsd.business.LSDBusinessImpl.java`)
3) Extend or adapt the interface of servlet functions (see example in `WebSupport:openlsd.web.*` and in `OpenLSDWebImpl` project)
4) Finally create the new `OpenLSD-Impl.jar` that will replace the original one with your business.

OpenLSD takes a huge part of the work away from you. However, if you want to change par of the logic, you still can do it but you will have to take care about the main problem which is to obtain a new index when inserting a document in the system in an efficient way, both for parallelism (multiple import support) and for the ability to take care about deleted documents and so “defragmentation” on the storage.
I try in my current implementation to cover both problems. So take care to keep those implementations, except if you know what you do.

One story example

The example will try to cover all steps to follow on how to obtain a complete implementation for one particular application form the OpenLSD framework.
The example target is the following:

- Each document will have four data: one string for the business index, one number for the business unit id, one string for the business owner id and one date for a life time limit for the document.
- The logic of import will be of three kinds:
  - Import from Web services from applications running under Tomcat
  - Import from applications on a different server than the OpenLSD Server
  - Import from applications but after a file transfer on the OpenLSD Server
- The logic of export will be of two kinds:
  - Access through applications running under Tomcat, limited according to the Business Unit id
  - Exporting documents to enable a copy on an independent media
- The logic of obliteration of documents will be of two kinds:
  - Deletion from applications running under Tomcat, limited according to the Business Owner id
  - Automatic from life time limit in batch
- The logic of the exploitation will be as follow:
  - One LSDServer only at the beginning
  - One LSDServer later on (when the number of documents will imply to have a mirror of the storage since the save on tape duration will become too long)
  - Every day an incremental check of consistency will be done (with MD5).
  - Every week-end a full check of consistency will be done (with no MD5).

DATABASE step

Now the objectives are written, we can start the process to start from the OpenLSD framework. First, we extend the database table LSDDOCUMENT as follow:

FROM

```
CREATE TABLE "LSD"."LSDDOCUMENT"
   (   "LID" NUMBER(30,0) NOT NULL ENABLE,
       "SID" NUMBER(30,0) NOT NULL ENABLE,
       "DID" NUMBER(30,0) NOT NULL ENABLE,
       "SIZEDOC" NUMBER(30,0) DEFAULT 0 NOT NULL ENABLE,
       "IDMETIER" VARCHAR2(240 BYTE),
       "DOCDATE" TIMESTAMP (3),
       "EMPREINT" VARCHAR2(240 BYTE),
        CONSTRAINT "PK_DOCUMENT" PRIMARY KEY ("LID", "SID", "DID")
  USING INDEX COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645)
  TABLESPACE "DATA"  ENABLE,
        CONSTRAINT "FK_DOC_STO" FOREIGN KEY ("LID", "SID")
         REFERENCES "LSD"."LSDSTORAGE" ("LID", "SID") ENABLE
   ) NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645)
  TABLESPACE "DATA" ;
```

MODIFIED TO (Business ID = IDMETIER, Business Unit Id = IDBUNIT, Business Owner Id = IDOWNER)

```
CREATE TABLE "LSD"."LSDDOCUMENT"
   (   "LID" NUMBER(30,0) NOT NULL ENABLE,
       "SID" NUMBER(30,0) NOT NULL ENABLE,
       "DID" NUMBER(30,0) NOT NULL ENABLE,
       "SIZEDOC" NUMBER(30,0) DEFAULT 0 NOT NULL ENABLE,
       "IDMETIER" VARCHAR2(240 BYTE),
       "DOCDATE" TIMESTAMP (3),
       "EMPREINT" VARCHAR2(240 BYTE),
       "IDBUNIT" NUMBER(30,0),
       "IDOWNER" VARCHAR2(150 BYTE),
       "LIMITDATE" TIMESTAMP (3),
        CONSTRAINT "PK_DOCUMENT" PRIMARY KEY ("LID", "SID", "DID")
  USING INDEX COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645)
  TABLESPACE "DATA"  ENABLE,
        CONSTRAINT "FK_DOC_STO" FOREIGN KEY ("LID", "SID")
         REFERENCES "LSD"."LSDSTORAGE" ("LID", "SID") ENABLE
   ) NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645)
  TABLESPACE "DATA" ;
```

AND

```
CREATE INDEX "LSD"."IDX_DOC_IDOWNER" ON "LSD"."LSDDOCUMENT" ("IDOWNER")
  COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645)
  TABLESPACE "INDEX" ;
```

AND

```
CREATE INDEX "LSD"."IDX_DOC_IDBUNIT" ON "LSD"."LSDDOCUMENT" ("IDBUNIT")
  COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645)
  TABLESPACE "INDEX" ;
```

AND

```
CREATE INDEX "LSD"."IDX_DOC_LIMITDATE" ON "LSD"."LSDDOCUMENT" ("LIMITDATE")
  COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS 2147483645)
  TABLESPACE "INDEX" ;
```

Data Structure Step

First we have to modify the Business implementation to fit our needs. To do that, we create a copy of `BusinessImpl` directory in Example and continue to implement it. For the safety of example, I rename `database.business` in `database.newbusiness` in the Example directory in the OpenLSD distribution.
I create also a new Eclipse project named OpenLSDImpl which will include the same classes but in `database.business` container and an associate `OpenLSD-ImplExample.jar` which must substitute the default simple `OpenLSD-Impl.jar`.
In Blue, you will find what was modified, in Black what is still the same. You have two parts:

1) ***Definition part***:
   - First you define new data (long idbunit, String idowner, …),
   - then the database column names (in fieldsext),
   - then the accessor (int ridb, int ridbunit, …) that allow to access the correct column name from fieldsext,
   - then some default SQL code as the allFieldsExt (select request based on "the business where condition" so excluding the business index),
   - the getWhereClause (to be append to the Legacy where condition with the business index),
   - the getFieldsSetExt (update values from a "business where condition"),
   - the getFieldsSetAllExt (update values from a "standard where condition", i.e. including business index in the update) ,
   - the getFieldsSetExtNULL (update to NULL values for every fields),
   - the doc_hintBusiness() (hint – for Oracle – to place in SELECT "hint" 1 FROM LSDDOCUMENT WHERE getWhereClauseBusiness())
2) ***Methods part***:
   - Two constructors: empty constructor, from another LSDBusiness object (clone), from an array of integers and an array of Strings which enables a standard constructor from any part of the value using a switch-case construction and cast to the correct type. **Take care about false cast!**
   - Getter from SQL select based on all fields (getFromResultSet)
   - A Setter for Insert command for all fields (getFieldsValues)
   - Add on PreparedStatement for the "business where condition" from getWhereClause (setToPrepStmtGetWhereClause)
   - Add on PreparedStatement for the update of values except Business index from getFieldsSetExt (setToPrepStmtGetFieldsSetExt)
   - Add on PreparedStatement for the update of values from getFieldsSetAllExt (setToPrepStmtGetFieldsSetAllExt)
   - clone, clear, equals and toString methods
   - setFromString and setFromStrings methods to fill in attributes from String or Strings (Delete, GetPaths, GetCopy and Import functions)
   - Add on toString and toCSV the new fields (toCSV is intend for database replication)

public class LSDBusinessImpl extends LSDBusiness

```
{
/** Business index */
public String idb = null;
/** Business Unit */
public long idbunit = LSDConstants.invalide_idx;
/** Business Owner */
public String idowner = null;
/** Limit of time validity */
public Timestamp limitdate = null;
/** Extended fields for Business */
protected static final String[] fieldsext = {"IDMETIER","IDBUNIT","IDOWNER","LIMITDATE"};
/** Index for extended fields for Business */
public static final int ridb = 0;
/** Index for extended fields for Business */
```

```
public static final int ridbunit = 1;
/** Index for extended fields for Business */
public static final int ridowner = 2;
/** Index for extended fields for Business */
public static final int rlimitdate = 3;
/** Values to add to allfields (replace select *) */
public static final String allFieldsExt = ","+fieldsext[ridb]+","+fieldsext[ridbunit]+
       ","+fieldsext[ridowner]+","+fieldsext[rlimitdate]+" ";
/** Where condition in " condition " format which will be added
 * to " WHERE Legacy = ? AND condition " and that must returns 1 unique document.      */
public static String getWhereClause = fieldsext[ridb]+" = ? ";
/** Set for all fields except fields used in getWhereClause */
public static String getFieldsSetExt = ","+fieldsext[ridbunit]+
       " = ?,"+fieldsext[ridowner]+" = ?,"+fieldsext[rlimitdate]+" = ? ";
/** Set for all fields      */
public static String getFieldsSetAllExt = ","+fieldsext[ridb]+" = ?,"+fieldsext[ridbunit]+
       "= ?,"+fieldsext[ridowner]+"= ?,"+fieldsext[rlimitdate]+" = ? ";
/** Set values as NULL (or empty) including where clause */
public static String getFieldsSetExtNULL = ","+fieldsext[ridb]+" = NULL,"+fieldsext[ridbunit]+
       "= "+LSDConstants.invalide_idx+","+fieldsext[ridowner]+"=NULL,"+
       fieldsext[rlimitdate]+" = NULL ";
/** @see openlsd.database.business.LSDBusiness#getAllFieldsExt() */
public String getAllFieldsExt() {
       return allFieldsExt;
}
/** @see openlsd.database.business.LSDBusiness#getFieldsExt(int) */
public String getFieldsExt(int number) {
       return fieldsext[number];
}
/** @see openlsd.database.business.LSDBusiness#getFieldsSetAllExt() */
public String getFieldsSetAllExt() {
       return getFieldsSetAllExt;
}
/** @see openlsd.database.business.LSDBusiness#getFieldsSetExt()    */
public String getFieldsSetExt() {
       return getFieldsSetExt;
}
/** @see openlsd.database.business.LSDBusiness#getFieldsSetExtNULL() */
public String getFieldsSetExtNULL() {
       return getFieldsSetExtNULL;
}
/** @see openlsd.database.business.LSDBusiness#getWhereClause()     */
public String getWhereClause() {
       return getWhereClause;
}
/** @see openlsd.database.business.LSDBusiness#doc_hintBusiness()   */
public String doc_hintBusiness() {
       switch (LSDConstants.typeDriver) {
       case LSDConstants.typeMySQL:
              return " ";
       case LSDConstants.typeOracle:
              return " /*+ index(lsddocument IDX_DOC_LIDMETIERDATE)*/ ";
       case LSDConstants.typePostgresql:
              return " ";
       default:
              LSDConstants.logger.error("NO DRIVER CORRECTLY SPECIFIED:"+
                            LSDConstants.typeDriver);
              return " ";
       }
}
/** Default empty constructor */
public LSDBusinessImpl() {
       this.idb = null;
       this.idbunit = LSDConstants.invalide_idx;
       this.idowner = null;
       this.limitdate = null;
}
/** Constructor from data
 * @param lsdb  */
public LSDBusinessImpl(LSDBusiness lsdb) {
       if (LSDBusiness.class.getClass() == LSDBusinessImpl.class.getClass())
              this.setFromLSDBusiness((LSDBusinessImpl) lsdb);
}
/** Constructor from 1 int[] for rank and 1 String[] for value
 * @param ranks
 * @param args  */
public LSDBusinessImpl(int []ranks, String []args) {
       this.setFromStringArray(ranks, args);
}
/** Constructor from data
 * @param lsdb  */
public LSDBusinessImpl(LSDBusinessImpl lsdb) {
       this.idb = lsdb.idb;
       this.idbunit = lsdb.idbunit;
```

```
		this.idowner = lsdb.idowner;
		this.limitdate = lsdb.limitdate;
	}
	/** Set from 1 int[] for rank and 1 String[] for value
	 * @param ranks
	 * @param args
	 * @return True if OK, else False */
	public boolean setFromStringArray(int []ranks, String []args) {
		boolean retour = false;
		if (ranks.length > args.length) {
			LSDConstants.logger.warn("During initialization of LSDBusinessImpl: not correct ranks and
	args:"+ranks.length+">"+args.length);
			return false;
		}
		int i, rank;
		String arg;
		for (i = 0; i < ranks.length; i++) {
			arg = args[i];
			rank = ranks[i];
			switch (rank) {
				case ridb:
					this.idb = arg;
					retour = true;
					break;
				case ridbunit:
					try  {
						this.idbunit = Long.parseLong(arg);
					} catch (NumberFormatException e) {
						LSDConstants.logger.fatal("Bad Long:"+arg+", revert to -1");
						this.idbunit = LSDConstants.invalide_idx;
					}
					retour = true;
					break;
				case ridowner:
					this.idowner = arg;
					retour = true;
					break;
				case rlimitdate:
					// Replace ; as separator of date and time by blank character
					arg = arg.replace(';', ' ');
					try {
						this.limitdate = Timestamp.valueOf(arg);
					} catch (IllegalArgumentException e) {
						LSDConstants.logger.fatal("Bad Timestamp:"+arg+", revert to NULL");
						this.limitdate = null;
					}
					retour = true;
					break;
				default:
				      LSDConstants.logger.warn("During initialization of LSDBusinessImpl: Unknown index "
						+rank+" for value "+arg);
					break;
			}
		}
		return retour;
	}
	/** @see openlsd.database.business.LSDBusiness#equals(openlsd.database.business.LSDBusiness)
	 	*/
	public boolean equals(LSDBusiness other) {
		if (LSDBusiness.class.getClass() == LSDBusinessImpl.class.getClass())
			return this.equals((LSDBusinessImpl) other);
		return false;
	}
	/** @see openlsd.database.business.LSDBusiness#setFromLSDBusiness(openlsd.database.business.LSDBusiness)
	*/
	public boolean setFromLSDBusiness(LSDBusiness lsdb) {
		if (LSDBusiness.class.getClass() == LSDBusinessImpl.class.getClass())
			return setFromLSDBusiness((LSDBusinessImpl) lsdb);
		return false;
	}
	/** Set internal value from ResultSet
	 * @param resultSet
	 * @return True if OK, else False */
	public boolean getFromResultSet(ResultSet resultSet) {
		try {
			this.idb = resultSet.getString(fieldsext[ridb]);
			this.idbunit = resultSet.getLong(fieldsext[ridbunit]);
			this.idowner = resultSet.getString(fieldsext[ridowner]);
			this.limitdate = resultSet.getTimestamp(fieldsext[rlimitdate]);
			return true;
		} catch (SQLException e) {
			LSDConstants.logger.error("SQL Exception LSDExtDbDocument");
			if (LSDConstants.LOGGING) {
				LSDDbSession.error(e);
			}
			return false;
```

```
        }
}
/** Set Values for Insert command
 * @return the String for Insert command Value part
 */
public String getFieldsValues() {
        String retour;
        if (idb == null) {
                retour = ",NULL,"+idbunit+",";
        } else {
                retour = ",'"+idb+"',"+idbunit+",";
        }
        if (idowner == null) {
                retour = retour+"NULL,";
        } else {
                retour = retour+"'"+idowner+"',";
        }
        if (limitdate == null) {
                return retour+"NULL) ";
        } else {
                return retour+"timestamp '"+limitdate+"') ";
        }
}
/** Set the value into the prepareStatement from Object for the Where Condition in getWhereClause
 * @param startrank The value to start from in the preparedStatement.setXXX call
 * @param preparedStatement
 * @return the next startrank or 0 if KO */
public int setToPrepStmtGetWhereClause(int startrank, LSDDbPreparedStatement preparedStatement) {
        try {
                preparedStatement.setString(startrank, idb);
        } catch (SQLException e) {
                LSDConstants.logger.error("Bad Prep Statement");
                preparedStatement.realClose();
                return 0;
        }
        return (startrank+1);
}
/** Set the value into the prepareStatement from Object for the set in GetFieldsSetExt
 * @param startrank The value to start from in the preparedStatement.setXXX call
 * @param preparedStatement
 * @return the next startrank or 0 if KO */
public int setToPrepStmtGetFieldsSetExt(int startrank, LSDDbPreparedStatement preparedStatement) {
        try {
                preparedStatement.setLong(startrank, idbunit);
                preparedStatement.setString(startrank+1, idowner);
                preparedStatement.setTimestamp(startrank+2, limitdate);
        } catch (SQLException e) {
                LSDConstants.logger.error("Bad Prep Statement");
                preparedStatement.realClose();
                return 0;
        }
        return (startrank+3);
}
/** Set the value into the prepareStatement from Object for the set in GetFieldsSetAllExt
 * @param startrank The value to start from in the preparedStatement.setXXX call
 * @param preparedStatement
 * @return the next startrank or 0 if KO */
public int setToPrepStmtGetFieldsSetAllExt(int startrank, LSDDbPreparedStatement preparedStatement) {
        try {
                preparedStatement.setString(startrank, idb);
                preparedStatement.setLong(startrank+1, idbunit);
                preparedStatement.setString(startrank+2, idowner);
                preparedStatement.setTimestamp(startrank+3, limitdate);
        } catch (SQLException e) {
                LSDConstants.logger.error("Bad Prep Statement");
                preparedStatement.realClose();
                return 0;
        }
        return (startrank+4);
}
/** Clear all data in the object */
public void clear() {
        this.idb = null;
        this.idbunit = LSDConstants.invalide_idx;
        this.idowner = null;
        this.limitdate = null;
}
/** Clone the current object but only copy values
 * @return The new object as a clone */
public LSDBusiness clone() {
        LSDBusinessImpl newbusiness = new LSDBusinessImpl(this);
        return newbusiness;
}
/**
 * @param lsdb
 * @return True if OK, else False
```

```
 * @see openlsd.database.business.LSDBusiness#setFromLSDBusiness(openlsd.database.business.LSDBusiness) */
public boolean setFromLSDBusiness(LSDBusinessImpl lsdb) {
	this.idb = lsdb.idb;
	this.idbunit = lsdb.idbunit;
	this.idowner = lsdb.idowner;
	this.limitdate = lsdb.limitdate;
	return true;
}
/** Comparison of two LSDBusinessImpl object on primary key (Business Index)
 * @param other
 * @return True if Business index are the same, else False */
public boolean equals(LSDBusinessImpl other) {
	return this.idb.equals(other.idb);
}
/** Object to String
 * @return the string that displays this object
 * @see java.lang.Object#toString() */
public String toString() {
	return " IDM:"+idb+";BUnit:"+idbunit+";Owner:"+idowner+";Lim:"+limitdate.toString();
}
/** (non-Javadoc)
 * @see openlsd.database.business.LSDBusiness#setFromStringForDelete(java.lang.String)
 */
public void setFromString(String line) {
	int []ranks = {LSDBusinessImpl.ridb };
	String []values = {line};
	this.setFromStringArray(ranks,values);
}
/** (non-Javadoc)
 * @see openlsd.database.business.LSDBusiness#setFromStringForImport(int, java.lang.String[])
 */
public int setFromStrings(int curargs, String[] args) {
	int []ranks = {LSDBusinessImpl.ridb,LSDBusinessImpl.ridbunit,LSDBusinessImpl.ridowner,
		LSDBusinessImpl.rlimitdate};
	String []newargs = {args[curargs],args[curargs+1],args[curargs+2],args[curargs+3]};
	this.setFromStringArray(ranks, newargs);
	return (curargs+4);
}
/** (non-Javadoc)
 * @see openlsd.database.business.LSDBusiness#toCSV()
 */
@Override
public String toCSV() {
	return idb+";"+idbunit+";"+idowner+";"+limitdate.getTime();
}
}
```

Web Step

Here you find some hits on how to include OpenLSD support in your servlets.
I will take the option that Oracle is your database software support. The same can be done using MySQL or PostGreSQL.

By default, one Tomcat instance has to have only one database for OpenLSD Server instance. Many OpenLSD Server can be used within one database, but from one Tomcat server, only one database can be access since there are some global data that are shared within the OpenLSD Framework.
Another way to say that, you can use only one LSDBusiness implementation in one database within one Tomcat instance.
You have then two choices, using a pool of database connections or not.

WEB GET

You can optimize the initialization part in the init part of the servlet, which is executed only once. By this way, the xml configuration step is done once and not each time there is a connection. (1 and 2a)
If you use no database pool of connections, then you have to use the second solution to create the connection by OpenLSD itself. (1 and 2b)
Obviously, the best solution is to use a database pool of connections.

INIT only once

1) Init global context: in the general initialization of the servlet (done only once)
Load log4j config file and openlsd-db config file from file real path and execute the initialization step.
The third argument contains the class that will be used as the default prefix class name in the logger.

```
LSDWebCommon.initialize(prefix+log4jfile, prefix+conflsddb,
      servletClassName.class);
```

INIT Database from existent connection from a pool of database connections

2a) Init LSD and Database context:
Set Oracle as default driver

```
LSDConstants.typeDriver = LSDConstants.typeOracle;
```

Once you get your own database connection, you can pass it trough the initAdmin call, such that OpenLSD will use this connection and so as to not open a new one.
If the Web function is in read mode (get), the first argument must be filled with the connection.
If in write mode (import), the second argument must be filled, the first one should be null.
The third argument contains a default `ArrayList<String>` that will contain plain text in case of error.

```
LSDDbAdmin admin = LSDWebCommon.initAdmin(connread,null,erreurs);
if (admin == null) {
    // There is an error since admin is null so you can take an action.
    return;
}
```

No call to `admin.close()` is necessary since the connection is not create by LSD. But if you want that OpenLSD take care of closing and getting the exception for you, you can still call admin.close().

INIT without a pool of database connections

2b) Init LSD and Database context:
Set Oracle as default driver

```
LSDConstants.typeDriver = LSDConstants.typeOracle;
```

As you don't get your own database connection, the initAdmin call will take one for you, such that OpenLSD can use this connection and so open a new one each time.

The first argument contains a default `ArrayList<String>` that will contain plain text in case of error.

```
LSDDbAdmin admin = LSDWebCommon.initAdmin(erreurs);
if (admin == null) {
    // There is an error since admin is null so you can take an action.
    return;
}
```

A call to `admin.close()` is necessary since the connection is create by LSD before the servlet is done (take care of error handling calling admin.close()).

Continue with dynamic part

3) Init OpenLSD Server identification:

Here you get the IpPort object that represents the OpenLSD Server on which you want to interact.
First arg is the `ArrayList<String>` for error handling.
Second arg is the previous `LSDDbAdmin` object if not null.
Third arg is the Legacy id (long number) on which you want to interact (id of the OpenLSD Server)
Fourth arg is the type of client you want (user, compressed, admin or protected)

```
LSDDbIpPort ipport = LSDWebCommon.getIpPort(erreurs, admin, legacy,
      LSDDbIpPort.fuser); // or fusercomp
if (ipport == null) {
      // There is an error since ipport is null
      return;
}
```

4) Init LSDBusiness object:

Here you create the `LSDBusiness` object as your implementation needs it.
You need to value for Get only the business index.

```
LSDBusinessImpl lsdb = new LSDBusinessImpl();
lsdb.idb = sidm;
```

5) Call to the LSD Web support function:

Here you call the real function.
Get: `ArrayList<String>` for error handling, `LSDDbAdmin` object, legacy id, `LSDBusiness` object, the `LSDDbIpPort` object and the `HttpServletResponse` object.
Behind, from the `HttpServletResponse` object, we get the `ServletOutputStream`.
All blocks of the needed file are returns directly to the user.
If everything is OK, it returns true.

```
if (! LSDWebGet.getDocument(erreurs, admin, legacy, lsdb,
      ipport, response)) {
      // There is an error since return is false
      return;
}
```

PoolGet: Same Get but from an internal pool of LSD Server connections to minimize connections.

```
if (! LSDWebGet.getPoolDocument(erreurs, admin, legacy, lsdb,
      ipport, response)) {
      // There is an error since return is false
      return;
}
```

Or use the Exception version (both on pooled and not pooled versions):

```
try {
      if (LSDWebGet.getPoolDocumentException(erreurs, admin, legacy, lsdb,
                  ipport, response) == 0) {
            ipport = null;
// Closing the database Connection (can be done elsewhere the servlet part)
            admin.close();
            admin = null;
            return;
      }
} catch (LSDNoConnexionException e) {
      quitInError(request,response,"No Connecxion to OpenLSD Server");
} catch (LSDServletOutputStreamException e) {
```

```
            quitInError(request,response,"Servlet Error");
      } catch (LSDBadRequestException e) {
            quitInError(request,response,"Bad Request Error");
      } catch (LSDInternalException e) {
            quitInError(request,response,"Internal Error");
      } catch (LSDNotFoundException e) {
            quitInError(request,response,"No document found");
      } catch (LSDUnknownException e) {
            quitInError(request,response,"Unknown Error");
      }
```

6) Enable shutdown in case of Pool of OpenLSD connections (LSDWebGet.getPoolDocument):

In **`destroy()`** function, you have to call the destructor of the OpenLSD connections pool.

```
public void destroy() {
      LSDWebGet.closeAll();
      super.destroy();
}
```

WEB PUT

Most of the things are the same.
Most of the things are the same. Scenario 1-2b is exactly the same, so only the 1-2a is show again.

1) Init global context: in the general initialization of the servlet (done only once)

Load log4j config file and openlsd-db config file from file real path and execute the initialization step.
The fourth argument contains the class that will be used as the default prefix class name in the logger.

```
LSDWebCommon.initialize(prefix+log4jfile, prefix+conflsddb,
      servletClassName.class);
```

2a) Init LSD and Database context:

Set Oracle as default driver

```
LSDConstants.typeDriver = LSDConstants.typeOracle;
```

Once you get your own database connection, you can pass it trough the initAdmin call, such that OpenLSD will use this connection and so as to not open a new one.
As the mode is in write (import), the second argument must be filled, the first one should be null.
The third argument contains a default ArrayList<String> that will contain plain text in case of error.

```
LSDDbAdmin admin = LSDWebCommon.initAdmin(null,connwrite,erreurs);
if (admin == null) {
    // There is an error since admin is null so you can take an action.
    return;
}
```

No call to `admin.close()` is necessary since the connection is not create by LSD. But if you want that OpenLSD take care of closing and getting the exception for you, you can still call admin.close().

3) Init OpenLSD Server identification:

Here you get IpPort objects that represent the OpenLSD Server on which you want to interact, based on two client: either (admin,protected) or (compressed,protected) according to the not compressed/compressed protocol you want.

```
LSDDbIpPort ipport = LSDWebCommon.getIpPort(erreurs, admin, legacy,
      LSDDbIpPort.fadmin); // or fusercomp
if (ipport == null) {
      // There is an error since ipport is null
      return;
}
LSDDbIpPort ipport = LSDWebCommon.getIpPort(erreurs, admin, legacy,
      LSDDbIpPort.fprotected);
if (ipport == null) {
      // There is an error since ipport is null
      return;
}
```

4) Init LSDBusiness object:

Here you create the `LSDBusiness` object as your implementation needs it.
For Put you should include all the values that are needed according to your business logic during import.

```
int [] rank = { LSDBusinessImpl.ridb,LSDBusinessImpl.ridbunit,
      LSDBusinessImpl.ridowner, LSDBusinessImpl.rlimitdate};
String [] args = {sidm,sidbunit,sidowner,slimitdate};
LSDBusinessImpl lsdb = new LSDBusinessImpl(rank, args);
```

5) **Call to the LSD Web support function**:

Here you call the real function.

Put: `ArrayList<String>` for error handling, `LSDDbAdmin` object, legacy id, the `LSDDbIpPort` object for put, the `LSDDbIpPort` object for control (protected), `LSDBusiness` object, and the file path.
Behind, the file is imported though a local access from OpenLSD Server (Servlet server is installed on the same server than OpenLSD Server).
If everything is OK, it returns true.

```
boolean result = LSDWebImport.putFileFromWeb(erreurs, admin, legacy,
      ipport1, ipport2, lsdb, uploadedFile);
if (result) {
      //OK, go for inform the user
      …
} else {
      //KO
      return;
}
```

PutNet: Same Put but the file is imported from the net, i.e. the Servlet server is not on the same server than OpenLSD Server.

```
boolean result = LSDWebImportNet.putFileFromWeb(erreurs, admin, legacy,
      ipport1, ipport2, lsdb, uploadedFile);
if (result) {
      //OK, go for inform the user
      …
} else {
      //KO
      return;
}
```

Jar implementation Step

With a new project, you can create your own OpenLSD-Impl.jar that will replace the original one with your implementation, so as to keep the 4 other jars (Common, Server, Client and Web). Et voilà!

Adapt functions to Business needs

Once database and primary functions are adapted, there are some extra functions that need to be adapted, or even created. We get back to our story to see if this is OK or not.

Data

- Each document will have four data: one string for the business index, one number for the business unit id, one string for the business owner id and one date for a life time limit for the document.
    - OK

The logic of import

- Import from Web services from applications running under Tomcat
    - OK using adapted OpenLSDWebImpl services, even if we probably need to manage the web page, the authentication and so on… but this is out of the scope of OpenLSD Framework.
- Import from applications on a different server than the OpenLSD Server
    - OK using ImportNet functions
- Import from applications but after a file transfer on the OpenLSD Server
    - OK using ImportBlock or Import functions after the file transfer is finished

The logic of export

- Access through applications running under Tomcat, limited according to the Business Unit id
    - OK using adapted OpenLSDWebImpl services, even if authentication and others points should be done but this is out of the scope of OpenLSD Framework.
- Exporting documents to enable a copy on an independent media
    - OK using adapted GetCopy function

The logic of obliteration of documents

- Deletion from applications running under Tomcat, limited according to the Business Owner id
    - OK using adapted OpenLSDWebImpl services, even if authentication and others points should be done but this is out of the scope of OpenLSD Framework.
- Automatic from life time limit in batch
    - One procedure will have first to get all Idbusiness of documents that are out of time in a list, then call Delete with a file containing this list, so OK.

The logic of the exploitation

- One LSDServer only at the beginning
    - OK, right now!
- One LSDServer later on (when the number of documents will imply to have a mirror of the storage since the save on tape duration will become too long)
    - **By using the Multiple Legacies support, this can be done by exchanging usual functions by ML ones.** ***Web support needs to be improved. So partially OK (web support missing as of chosen between several ML OpenLSD Servers, but using standard Web support on Primary or Secondary OpenLSD Servers is OK).***
- Every day an incremental check of consistency will be done (with MD5).
    - OK, using the consistency check function.
- Every week-end a full check of consistency will be done (with no MD5).
    - OK, using the same consistency check function with –nomd5 option.