OpenLSD HowTo

Author: Frédéric Brégier under LGPL

PRESENTATION **1**
STANDARD EXTERNAL JAR VERSIONS 2
SPECIFIC EXTERNAL JAR VERSIONS 2
THE GLOBAL LOGIC 3
OPENLSD PROJECT IS ORGANIZED IN DIFFERENT DIRECTORIES 3
OPENLSD IS THEN SUBDIVIDED IN MULTIPLE JARS 3
SOME OTHER DIRECTORIES ARE PRESENT IN OPENLSD 4
OPENLSDWEBIMPL CONTAINS THE FOLLOWING EXAMPLES 4
OPENLSD CONTAINS THE FOLLOWING EXAMPLE SCRIPTS 4
- *Admin functions:* *4*
- *Check functions:* *5*
- *Document handling functions:* *5*
- *Import functions:* *5*
- *Delete function:* *6*
- *Multiple support functions:* *6*

HOW TO INSTALL OPENLSD **8**
HOW TO INSTALL TOMCAT LSD INSTANCE WITH ORACLE **10**
HOW TO MODIFY CONFIGURATION FILES **11**
CONTEXT.XML 11
OPENLSD.CFG 12
LEGACY<N>.XML 15
OPENLSDDB.CFG 16
SLF4J.XML 17
ADAPT ANY VALUE AS WANTED IN OPENLSD.COMMON.LSDCONSTANTS 19
HOW TO EXTEND TO A NEW DATABASE SUPPORT **21**

Presentation

OpenLSD is based on 3 eclipse packages. For each of them, you have to open them in eclipse as separate projects (OpenLSD, OpenLSDWebImpl, OpenLSDC).

1. OpenLSD main project
2. OpenLSDWebImpl sub project for web support (using servlet in Tomcat)
3. OpenLSDC which proposes some optimization written in C but is not mandatory (getFileFromPath and openlsddu available for Windows, AIX)

A fourth project OpenLSDImpl is included and is about an example of extension-rewrite of the Business part so as to explain how to use the OpenLSD Framework.

Optionally there is the module fast_md5 which enable optimization to compute MD5 using a C library (".dll" under windows, ".so" under Linux or AIX as tested until now).

The project depends on external jars. Most of them are the standard up to date version. Some of them are using one specific version or adapted version. All of them are in ALLJARS.zip.

Standard external jar versions

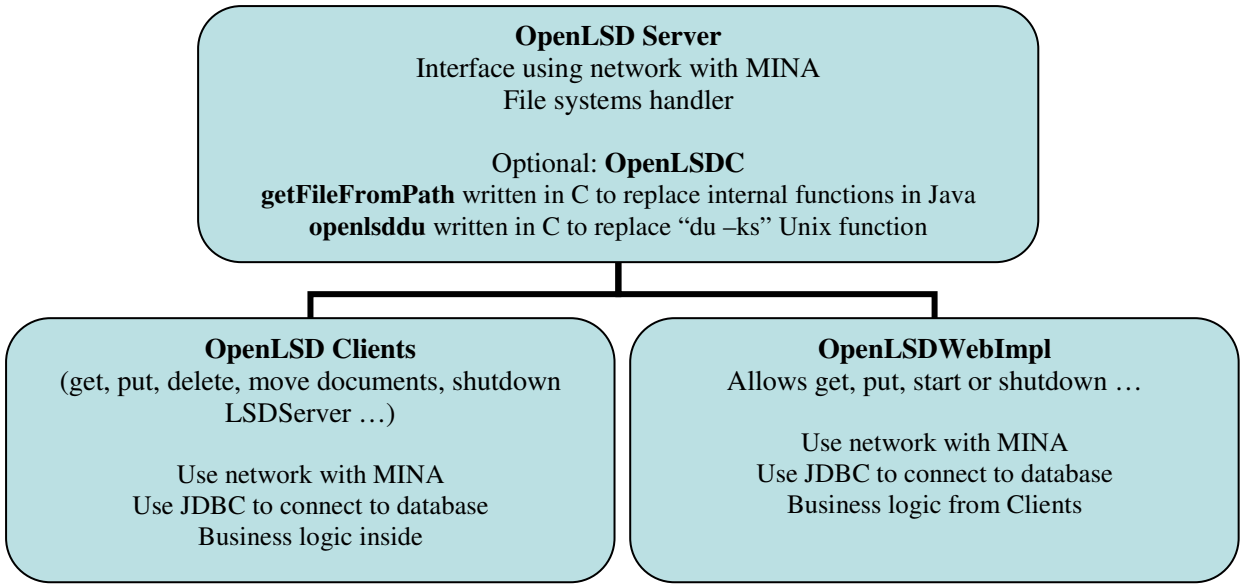
- Commons apache libraries:
  - commons-fileupload-1.2.jar
  - commons-io-1.3.2.jar
  - commons-codec-1.3.jar
  - commons-lang-2.3.jar
- XML support:
  - dom4j-1.6.1.jar
  - jaxen-1.1.1.jar
- MINA and relevant libraries:
  - jzlib-1.0.7.jar (compression support)
  - slf4j-api-1.5.0.jar (log support)
  - logback-access-0.9.8.jar
  - logback-classic-0.9.8.jar
  - logback-core-0.9.8.jar
- Factory resource for JDBC from Tomcat:
  - commons-el.jar
  - naming-factory.jar
  - naming-factory-dbcp.jar
  - naming-resources.jar
- Oracle support:
  - ojdbc14.jar or ojdbc5.jar or ojdbc6.jar
  - nls_charset12.jar or orai18n.jar
- PostGreSQL support:
  - postgresql-8.2-505.jdbc4.jar (JDK6)
  - postgresql-8.2-505.jdbc3.jar (JDK4 and JDK5)
- MySQL support:
  - mysql-connector-java-5.0.4-bin.jar

Specific external jar versions

- MINA and relevant libraries:
  - Mina-2-M2 (as today): Mina-2-Pre-M2.jar and Mina-2-Pre-M2-ASW-Common.jar (this is an extension for HTTP support using MINA where OpenLSD only uses the common part for the HTTP codec): both fixed validated version for this project
- OpenLSD support:
  - fast_md5.jar and the relative “so” or “dll” library in C if wanted (modified version for the java version). The Native library is available for Windows, Linux (from the original author) and for AIX 5.3.

Versions of JAR are constructed with JDK6. If you want to use JDK5, it works but you have to download source files of dependant project to get JAR in JDK5 format.

The global logic



No database is needed for OpenLSD Server since it does not assign index, those are given by the client through the message. The OpenLSD Client is responsible to take care about the business logic and also the database support (whatever the database is) and to assign one new index when necessary. The main reason of this organization is to enable a stable development of the Server part, whatever the Client does.
For one application, one team will have therefore to only focus on the client part and to start by extending classes from OpenLSD to meet their wishes.

OpenLSD project is organized in different directories

- **Server**: this part is the kernel part for OpenLSD Server
- **ServerApp**: this part is some applications that should be started from the same server that runs the OpenLSD Server itself.
- **ClientSupport**: this part is the kernel part for OpenLSD Client
- **ClientApp**: this part are for applications that run as Clients (Admin part, Check part, Session check part) and for the kernel part for the Business support (database access, import logic, delete logic, …)
- **BusinessImpl**: this part is the implementation specific part for the Business
- **WebSupport**: this part is the kernel part for the OpenLSD Servlet support
- **Common**: this part is about some global classes need by every package
- **Multiple**: this part is the implementation of some functions in Multiple Legacies mode support and of the specific Op Handler to maintain the LSD mirroring.

OpenLSD is then subdivided in multiple jars

- **OpenLSD-Server.jar**: Server and ServerApp parts
- **OpenLSD-Common.jar**: Common part
- **OpenLSD-Web.jar**: WebSupport part
- **OpenLSD-Client.jar**: ClientSupport, ClientApp
- **OpenLSD-Impl.jar**: BusinessImpl part
- **OpenLSD-Multiple.jar**: Multiple part (same as Client but in Multiple Legacy Mode plus the OpHandler)

Here is a presentation of the jars needed according to the functions:

Function	Server.jar	Client.jar	Multiple.jar	Web.jar	Impl.jar	Common.jar
OpenLSD Server	X					X
OpenLSD Client (get, put, admin)		X	X		X	X
OpenLSD Heavy Client (check consistency, init storage, check binary similarity, import based on check similarity)	X	X	X		X	X
OpenLSD Web in servlet (with the war from OpenLSDWebImpl)		X	X	X	X	X

Jars can be constructed from "jardesc" on the top of the OpenLSD Eclipse project. They should be modified to reflect the localization of the directory that will contain the jar files. To do that, just edit each file by modifying the following part:

```
<jar path="D:/ALLJARS/OpenLSD-Server.jar"/>
```

Some other directories are present in OpenLSD

- **Config**: files as example for the configuration of OpenLSD or the database (both Windows or Unix versions)
- **Doc**: API documentation
- **Scripts**: file as example to run some specific function as start OpenLSD Server, shutdown, import files, Check consistency, Check sessions, Delete files (both Windows or Unix versions and Windows example for PostGreSQL almost identical to Oracle version)

OpenLSDWebImpl contains the following examples

- Admin functions (local and distant)
- Client simple functions (get with or without MINA pool of connection, get and put with or without compression, delete support)

OpenLSD contains the following example scripts

All of them exist in bat or sh versions. All of them are to be considered as example and not as production ready. You have to adapt them to your needs.

- **Admin functions:**
  - **LSDInitOraFromConfigFiles** to initialize values in the database according to the configuration files in XML,
  - **LSDServerOra** to start LSDServer,
  - **LSDAdminOraShutdown** to stop LSDServer,
  - **LSDAdminOraInitStorage** to create some new Storages in each Legacy.

  The main class **openlsd.appli.admin.LSDAdmin** has more options. You can create new scripts as needed. Take a look at the code to know what options are available.

- **Check functions:** multiple options are possible. The following are only some examples. Check the source code to see more options.
    - **LSDCheckSession** to check the current number of sessions from LSDServer point of vue
    - **LSDCheckInDBThreadedOra** and its **DualLimit** version to check the consistency between the LSDServer document and the associated LSD database. The main idea here is to first load simple definition of each file (technical index and MD5 value), then to compare them with they internal database representations, and finally to check the Storage consistency (size and internal values). There are several options that are supported, such as check without any correction applied, or check with corrections applied on LSDServer storage only, on database only or both.
    - **LSDCheckDbConsistency** to check the consistency within the LSD database only. It checks the size, the next available index, the deleted index.
    - **LSDCheckSimilar** to check if one new document (not already imported) is already present in one specific Legacy, based on size, MD5 and then binary comparison. It only runs on the OpenLSD Server that hosts the Legacy and with non crypted Legacy.

- **Document handling functions:** multiple options are possible. The following are only some examples. Check the source code to see more options.
    - **LSDGetCopy** to export copy of files to the outpath directory locally from a source definition file with an out file including the business index from source file and the final path to the copied files. The getter should be on the same server than the LSDServer since the paths are local to the LSD Server.
    - **LSDGetPaths** to get real paths of files into the LSD Server storage locally from a source definition file with an out file including the real path of the original files. The getter is on the same server than the LSDServer. This function is useful if someone wants to make a tar of some files as stored in the LSD Server.

- **Import functions:** multiple options are possible. The following are only some examples. Check the source code to see more options.
    - **LSDImportNetOra** to import files from a directory through the network interface using a limited number of database connection from a source definition file. The importer is not on the same server than the LSDServer.
    - **LSDImportOra** to import files one by one from a source definition file using a limited number of database connection from a source definition file. The importer must be physically on the same server than LSDServer.
    - **LSDImportOra-BLOCK** to import files by group of files from a source definition file using a limited number of database connection from a source definition file. The importer must be physically on the same server than LSDServer.
    - **LSDAutoImportOra-fromDir** or **LSDAutoImportOra-fromDir-Block** to import files one by one from a directory or by a group of files from a directory and as a permanent importer (so the name auto import). These auto importers have to be physically on the same server than LSDServer. To stop them, you have to create a specific file as a shutdown order.
    - **LSDServerImportXXXCheckSimilarOra** (where XXX can be empty or BLOCK) to import files as LSDImportOra or LSDImportOra-BLOCK but using the CheckSimilar function before the import, so as to check if the document already exists on a binary comparison check. This function only runs on the OpenLSD Server that hosts the Legacy and with non crypted Legacy.

- **Delete function:**
  - **LSDDelete** to delete files from database and LSDServer storage. This function is obviously sensible.

- **Multiple support functions:**
  - **LSDInitOraFromConfigFilesML** to initialize values in the database according to the configuration files in XML when one Legacy is already defined (add other legacies only).
  - **LSDInitOpFromDB** initializes the Op table from all documents that exist in the database in order to be able to resynchronize one OpenLSD Server with another one. It is intend either when one wants to add a mirror function on an existing OpenLSD solution without mirror, either to resynchronize files in the mirror after a crash. It takes as argument the Legacy and the IpPort from which the files will be copied.
    Other versions exist like **xForStorage** (for one storage only) or **xPL** (using a PL/SQL procedure to go faster). Also an option concurrent enables to run it while OpenLSD Server is also running on both import and delete but at the price of slower run due to synchronization issue.
  - **LSDCheckInDBThreadedOraML** and its **DualLimit** version to check the consistency between one LSDServer from Multiple Legacies and the associated LSD database. The main idea here is to first load simple definition of each file (technical index and MD5 value), then to compare them with they internal database representations, and finally to check the Storage consistency (size and internal values). There are several options that are supported, such as check without any correction applied, or check with corrections applied on LSDServer storage only, on database only or both.
  - **LSDImportNetOraML** to import files from a directory through the network interface using a limited number of database connection from a source definition file. The importer is not on the same server than the LSDServer. ***This version implies the use of the Op Handler.***
  - **LSDImportOraML** to import files one by one from a source definition file using a limited number of database connection from a source definition file. The importer must be physically on the same server than LSDServer. ***This version implies the use of the Op Handler.***
  - **LSDImportOra-BLOCKML** to import files by group of files from a source definition file using a limited number of database connection from a source definition file. The importer must be physically on the same server than LSDServer. ***This version implies the use of the Op Handler.***
  - **LSDServerImportXXXCheckSimilarOraML** (where XXX can be empty or BLOCK) to import files as LSDImportOra or LSDImportOra-BLOCK but using the CheckSimilar function before the import, so as to check if the document already exists on a binary comparison check. This function only runs on the OpenLSD Server that hosts the Legacy and with non crypted Legacy.
    This version implies the use of the Op Handler.
  - **LSDDeleteML** to delete files from database and LSDServer storage. This function is obviously sensible. ***This version implies the use of the Op Handler.*** This function is for now immediate (delete immediately after asked), it could be later on with a delay to handle errors of users.
  - **LSDAutoImportOra-fromDirML** or **LSDAutoImportOra-fromDir-BlockML** to import files one by one from a directory or by a group of files from a directory and as a permanent importer (so the name auto import). These auto importers have to be

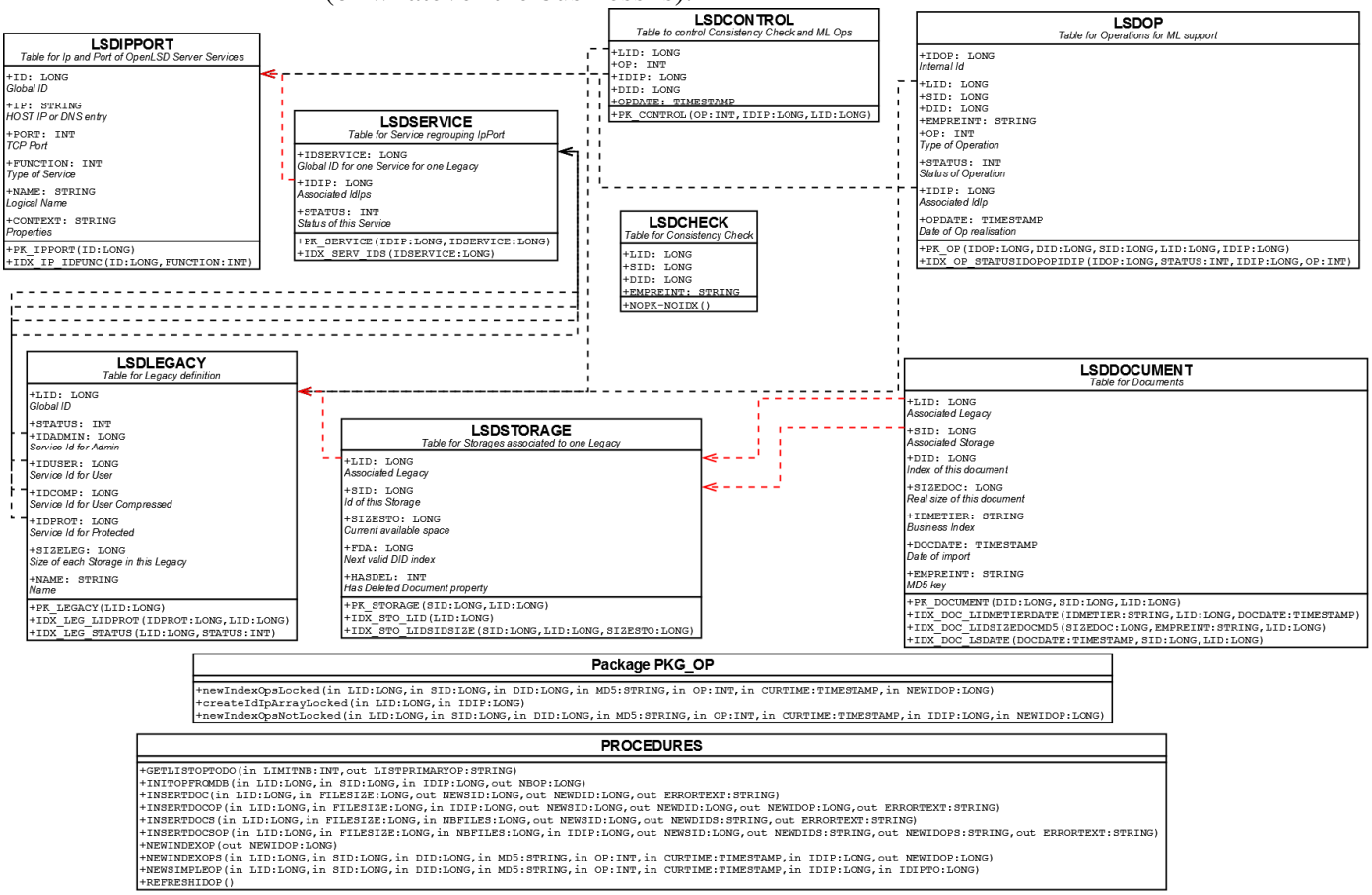
physically on the same server than LSDServer. To stop them, you have to create a specific file as a shutdown order. ***This version implies the use of the Op Handler.***

- **LSDOpHandlerML** to launch the Op Handler which is responsible to maintain the consistency between all Legacies that are in relation (replication). It uploads or deletes documents in other Legacies according to the action taken and registered using ML versions.

How to install OpenLSD

The main steps to install OpenLSD for the current ATLAS example application are the following:

- Create a database schema from **conflsd-oracle.txt** or **conflsd-postgre.txt** (later on will be available MySQL version) using a database connection. This file is in the Config directory. One should consider for huge volume to use a partitioned table for LSDDOCUMENT table. Different options can be implemented, for instance to partition on LID/SID couple or on IDMETIER (or whatever the business is).



- Install all jars in a directory you want that will be included in the classpath: all dependent jar are in ALLJARS.zip. It contains also specific version of some jars like MINA and Fast_MD5. If you use a JDK5, you have to recompile jar since they are compiled using JDK6.

- Install and adapt config files in a directory you want. Config files examples are in the directory Config in OpenLSD project. Mainly, there are 4 kinds of configuration files:
  1. **openlsd.xml** for the LSD Server configuration (ports, functions, but no database) and legacy configuration files inclusion
  2. **legacy<n>.xml** for Legacy storage configuration (mounting point, size, crypto, …)
  3. **openlsddb.xml** for LSD Clients configuration (database access for oracle or postgre)
  4. **slf4j-client.xml**, **slf4j-server.xml** and **slf4j-web.xml** for Log support

- Install and adapt scripts from the directory Scripts in OpenLSD project according to directories

- Then you must initialize some values inside the Database. To do that, one can do by hand or one can use the java script named LSDServerOraInitStorage (bat or sh). Once edited, this script will take three arguments: the openlsddb.xml file, the openlsd.xml file and the server IP address or DNS name.

- If you are the System administrator, you will have to defined filesystems and mount points such that Storage are correctly set up. To do that, first you have to interact with the database and the LSD Server using the LSDAdminOraInitStorage (bat or sh). By default, it creates 10 new storages for each Legacy (by default 2).

- Some check can be done by starting LSDAdminOraShutdown (bat or sh): it should be able to access to the database but not of course to OpenLSD Server. Check the log (LSDClient.log).

- Second check the start of LSDServer using the script LSDServerOra (bat or sh): it should start normally. Check the log (LSDServer.log).

- Test again LSDAdminOraShutdown: it should now be able to stop the LSDServer. Check the logs (LSDClient.log and LSDServer.log).

- Test finally import and delete with any LSD Import scripts and LSD Delete scripts.

You're done for the main part (Client-Server).

You can also compile C programs using a GCC compiler on any platform or whatever C compatible compiler. One is a library to be load from Java in Fast_MD5 package (MD5.so or dll), the others are in OpenLSDC project as two simple executables (checkFileFromPath and openlsddu).

For the Config files, you will find the tomcat example and some hints on the meaning of values.

How to install Tomcat LSD instance with Oracle

This example is using Tomcat 5.5.
You have to install common libraries in tomcat/common/lib:

- Commons apache libraries: (needed by Put functions)
  - commons-fileupload-1.2.jar
  - commons-io-1.3.2.jar
  - commons-codec-1.3.jar
  - commons-lang-2.3.jar
- XML support:
  - dom4j-1.6.1.jar
  - jaxen-1.1.1.jar
- MINA and relevant libraries:
  - Mina-2-M2 (as today): Mina-2-Pre-M2.jar and Mina-2-Pre-M2-ASW-Common.jar
  - jzlib-1.0.7.jar (compression support)
  - slf4j-api-1. 5.0.jar (log support)
  - logback-access-0.9.8.jar
  - logback-classic-0.9.8.jar
  - logback-core-0.9.8.jar
- Factory resource for JDBC from Tomcat: (or any other way to get jdbc database connection)
  - commons-el.jar
  - naming-factory.jar
  - naming-factory-dbcp.jar
  - naming-resources.jar
- Oracle support:
  - ojdbc14.jar or ojdbc5.jar or ojdbc6.jar
  - nls_charset12.jar or orai18n.jar
- MySQL support:
  - mysql-connector-java-5.0.4-bin.jar
- OpenLSD support:
  - fast_md5.jar and the relative so or dll library in C if wanted
  - OpenLSD-Client.jar
  - OpenLSD-Common.jar
  - OpenLSD-Web.jar
  - OpenLSD-Impl.jar

Some of the config files are placed inside the project WAR. You can copy the modified config files inside the web source project so as they will be included into the final WAR.
For PostGreSQL, you can follow the website from PostGreSQL itself.

How to modify configuration files

context.xml file from tomcat/conf directory:

Here you will have to define the database access through the servlet in a DataSourceFactory. You can change the value and also the localization of this definition if you know what you do.

```
<!-- The contents of this file will be loaded for each web application -->
<Context>

    <!-- Default set of monitored resources -->
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <Resource
      name="jdbc/poollsdora"
      auth="Container"
      type="javax.sql.DataSource"
      factory="org.apache.tomcat.dbcp.dbcp.BasicDataSourceFactory"
      maxActive="20"
      maxIdle="2"
      username="lsd"
      maxWait="5000"
      driverClassName="oracle.jdbc.OracleDriver"
      password="lsd"
      url="jdbc:oracle:thin:@localhost:1521:lsd"/>
    <!--Uncomment this to disable session persistence across Tomcat restarts -->
    <!--
    <Manager pathname="" />
    -->

</Context>
```

Here are the possible values:

- `name`: the name as seen by the servlet for the DataSource
- `maxActive`, `maxIdle`, `mawWait`: specify the maximum number of active database connection, of unactive but in the spool connection and the time before the connection is over
- `username` and `password`: specify the user and password for the database access
- `url`: specify the url in jdbc format to access to the database (here Oracle example)

openlsd.cfg

This file defines the LSD Server configuration (ports, functions, but no database), legacy configuration files inclusions and SLF4J configuration file inclusion.

```
<openlsd>
      <server>
            <timeout>60</timeout>
            <blocksize>16277</blocksize>
            <fastmd5path>D:/ALLJARS/lib/arch/win32_x86/MD5.dll</fastmd5path>
            <getfilepath>D:/LSD/getFileFromPath3.exe</getfilepath>
            <runlsdpath>D:/Source/OpenLSD/Scripts-
Ora/LSDServerOra.bat</runlsdpath>
            <runlsdcheckpath>D:/Source/OpenLSD/Scripts-
Ora/LSDCheckInDbThreadedOra.bat</runlsdcheckpath>
            <dfpath>df -kP</dfpath>
            <dupath>D:/LSD/openlsddu.exe</dupath>
            <slf4jpath>D:/LSDORA/slf4j-server.xml</slf4jpath>
            <httpport>8085</httpport>
      </server>
      <admin>
            <port>8082</port>
            <compressionserver>false</compressionserver>
            <compressionclient>false</compressionclient>
            <password>abcdefghijkl</password>
            <funcadmin>true</funcadmin>
            <funcdel>false</funcdel>
            <funcget>true</funcget>
            <funcgetna>true</funcgetna>
            <funcinfo>true</funcinfo>
            <funcmove>true</funcmove>
            <funcput>true</funcput>
            <funcputna>true</funcputna>
            <funcputfile>true</funcputfile>
      </admin>
      <client>
            <port>8084</port>
            <compressionserver>false</compressionserver>
            <compressionclient>false</compressionclient>
            <funcadmin>false</funcadmin>
            <funcdel>false</funcdel>
            <funcget>true</funcget>
            <funcgetna>true</funcgetna>
            <funcinfo>false</funcinfo>
            <funcmove>false</funcmove>
            <funcput>true</funcput>
            <funcputna>true</funcputna>
            <funcputfile>true</funcputfile>
      </client>
      <clientcomp>
            <port>8081</port>
            <compressionserver>true</compressionserver>
            <compressionclient>true</compressionclient>
            <funcadmin>false</funcadmin>
            <funcdel>false</funcdel>
            <funcget>true</funcget>
            <funcgetna>true</funcgetna>
            <funcinfo>false</funcinfo>
            <funcmove>false</funcmove>
            <funcput>true</funcput>
            <funcputna>true</funcputna>
            <funcputfile>true</funcputfile>
```

```
	</clientcomp>
	<protected>
		<port>8083</port>
		<compressionserver>false</compressionserver>
		<compressionclient>false</compressionclient>
		<password>abcdefghijkl</password>
		<funcadmin>false</funcadmin>
		<funcdel>true</funcdel>
		<funcget>false</funcget>
		<funcgetna>false</funcgetna>
		<funcinfo>true</funcinfo>
		<funcmove>true</funcmove>
		<funcput>true</funcput>
		<funcputna>true</funcputna>
		<funcputfile>true</funcputfile>
	</protected>
	<legacy>
		<file>D:/LSDORA/legacy1.xml</file>
		<file>D:/LSDORA/legacy2.xml</file>
	</legacy>
</openlsd>
```

Here are the possible values:

- Server part
  - `timeout`: specify the timeout time of an idle connection in second in OpenLSD
  - `blocksize`: specify the default block size for OpenLSD (size of minimal message), should be a multiple of 8K minus 104 bytes. Usually set as 16K minus 104 bytes so 16280.
  - `fastmd5path`: specify the path to the filename for the fastMD5 C library for the fastMD5 jar library
  - `getfilepath`: specify the path to the filename for the getfilepath C command
  - `runlsdpath`: specify the full command to start OpenLSD from Tomcat
  - `runlsdcheckpath`: specify the full command to check consistency in OpenLSD and the database
  - `dfpath`: path and args for 'df' command (in Java6, it can be ignored if the module in LSDConstants is uncommented and allowed)
  - `dupath`: path for opendu executable command
  - `slf4jpath`: specify where is the configuration file for slf4j support
  - `httpport`: specify the port on which the HTTP Service will listen
- Clients part: admin, client, clientcomp (compressed client) and protected part
  - `port`: specify on which port this client is listening
  - `compressionserver` and `compressionclient`: true to enable compression, false to disable compression (in general it should be disabled)
  - severall functions: for each function, specify if this server will enable or not the support of this function
    - `admin` (stop, start)
    - `delete`
    - `get` and `get with NoAck`
    - `info`
    - `move`
    - `put` and `put with NoAck` and `putfile` (local put)
- Legacy part
  - `file`: specify where is the corresponding legacy configuration file

The mapping of the function should be as follow, but one can change this according to specific needs. We specify the necessity of this support.

Client	Port	Compression	Admin	Del	Get	GetNa	Info	Move	Put	PutNa	PutFile
ADMIN	**X**	**-**	**X**	**-**	**-**	**-**	**X**	**X**	**-**	**-**	**-**
Necessity	Y		Y				Y	Y			
CLIENT	**X**	**-**	**-**	**-**	**X**	**X**	**-**	**-**	**X**	**X**	**X**
Necessity	N				Y	Y			Y	Y	Y
COMP	**X**	**X**	**-**	**-**	**X**	**X**	**-**	**-**	**X**	**X**	**X**
Necessity	N	Y			Y	Y			Y	Y	Y
PROTECTED	**X**	**-**	**-**	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
Necessity	Y			Y	Y	Y	Y	Y	Y	Y	Y

legacy<n>.xml

This file defines the Legacy storage configuration (mounting point, size, crypto, …) that are included in the openlsd.xml file.

```
<legacy>
      <lid>-9223372036854775807</lid>
      <name>MON LEGACY1</name>
      <base>d:/LSDORA/Legacy1/files</base>
      <outbase>d:/LSDORA/Legacy1/out</outbase>
      <storagesize>100000000</storagesize>
      <crypted>false</crypted>
      <key></key>
      <status>true</status>
</legacy>
```

Here are the possible values:

- `lid`: the numeric unique id of this legacy
- `name`: the logical name of this legacy
- `base`: the path of the base of all storages of this legacy
- `outbase`: the path of the base for the outbase storage of this legacy (export, temporary, …)
- `storagesize`: the size of each storage of this legacy (should be 10% lower than real filesystem size)
- `crypted`: set if this legacy is crypted (cannot be changed once set)
- `key`: the associated key of this legacy (cannot be changed once set)
- `status`: the associated status, True meaning fully opened, False meaning close to any write or delete operations except move operations, read operations are always allowed

openlsddb.cfg

This file defines the LSD Clients configuration (database access and log4j configuration file inclusion).

```
<openlsddb>
      <driver>oracle.jdbc.OracleDriver</driver>
      <server>jdbc:oracle:thin:@localhost:1521:lsd</server>
      <user>lsd</user>
      <password>lsd</password>
      <timeoutcon>3000</timeoutcon>
      <timeout>60</timeout>
      <blocksize>16277</blocksize>
      <slf4jpath>D:/LSDORA/slf4j-client.xml</slf4jpath>
      <business>openlsd.business.LSDBusinessImpl</business>
</openlsddb>
```

Here are the possible values:

- `driver`: which class for the JDBC driver
- `server`: the connect string
- `user`: the user for the database
- `password`: the associated password
- `timeoutcon`: timeout in millisecond when connection is to be done
- `timeout`: timeout of an idle connection in second
- `blocksize`: specify the default block size for OpenLSD (size of minimal message), should be a multiple of 8K minus 104 bytes. Usually set as 16K minus 104 bytes so 16280.
- `slf4jpath`: specify where is the configuration file for slf4j support
- `business`: specify which class implements the LSDBusiness, this class must be in the classpath

For PostGreSQL, only two values need to be changed:

```
      <driver>org.postgresql.Driver</driver>
      <server>jdbc:postgresql://localhost:5432/lsd</server>
```

slf4j.xml

Each type of client has its own slf4j.xml configuration file. So there is one for the Server, one for the Client and one for the Web support.

Server version: rotations each day on file

```
<configuration>
  <appender name="FILE"
    class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>D:/LSDORA/log/LSDServer2.log</file>
      <Append>true</Append>
      <BufferedIO>false</BufferedIO>
    <ImmediateFlush>true</ImmediateFlush>
    <rollingPolicy
      class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <FileNamePattern>
        D:/LSDORA/log/LSDServer2-%d{yyyy-MM-dd}.log
      </FileNamePattern>
    </rollingPolicy>
    <layout class="ch.qos.logback.classic.PatternLayout">
      <Pattern>
        %date %level [%thread] %logger [%file %method : %line] %msg%n
      </Pattern>
    </layout>
  </appender>
  <root>
    <level value="warn" />
    <appender-ref ref="FILE" />
  </root>
</configuration>
```

Client version: rotations each day on file and output

```
<configuration>
  <appender name="FILE"
    class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>D:/LSDORA/log/LSDClient2.log</file>
      <Append>true</Append>
      <BufferedIO>false</BufferedIO>
    <ImmediateFlush>true</ImmediateFlush>
    <rollingPolicy
      class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <FileNamePattern>
        D:/LSDORA/log/LSDClient2-%d{yyyy-MM-dd}.log
      </FileNamePattern>
    </rollingPolicy>
    <layout class="ch.qos.logback.classic.PatternLayout">
      <Pattern>
        %date %level [%thread] %logger [%file : %line] %msg%n
      </Pattern>
    </layout>
  </appender>
  <appender name="STDOUT"
    class="ch.qos.logback.core.ConsoleAppender">
    <ImmediateFlush>true</ImmediateFlush>
    <layout class="ch.qos.logback.classic.PatternLayout">
      <Pattern>%date %level [%thread] %logger [%file %method : %line]
%msg%n</Pattern>
    </layout>
  </appender>
  <root>
    <level value="debug" />
```

```
    <appender-ref ref="FILE" />
      <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

Web version: rotations each day on file

```
<configuration>
  <appender name="FILE"
    class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>D:/LSDORA/log/LSDWeb.log</file>
      <Append>true</Append>
      <BufferedIO>false</BufferedIO>
    <ImmediateFlush>true</ImmediateFlush>
    <rollingPolicy
      class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <FileNamePattern>
        D:/LSDORA/log/LSDWeb-%d{yyyy-MM-dd}.log
      </FileNamePattern>
    </rollingPolicy>
    <layout class="ch.qos.logback.classic.PatternLayout">
      <Pattern>
        %date %level [%thread] %logger [%file : %line] %msg%n
      </Pattern>
    </layout>
  </appender>
  <root>
    <level value="warn" />
    <appender-ref ref="FILE" />
  </root>
</configuration>
```

Adapt any value as wanted in openlsd.common.LSDConstants

Some values can be changed to allow different behaviour.
Logging properties:

```
/**
 *  Mode debug (very verbose)
 */
public static final boolean DEBUG = false;
/**
 *  Mode logging (partially verbose)
 */
public static final boolean LOGGING = false;
```

External C functions:

```
/**
 * Use C functions : only for some specific functions in Server
 */
public static final boolean useCFunctions = true;
```

Size of internal buffers:

```
/**
 * Default size for buffers (NIO)
 */
public static final int BUFFERSIZEDEFAULT = 65536; // 64K
```

Limit Size for switching between Ack and No Acked method for Web retrieve:

```
/**
 *  Limit size where NoAck method will be used (in Web retrieve functions)
 */
public static final int WEBRETRIEVENOACKLIMIT = 1024000;
```

Time limit before retry and limit number of retry:

```
/**
 *  Time elapse for retry in ms
 */
public static final long RETRYINMS = 10;
/**
 * Number of retry before error
 */
public static final int RETRYNB = 3;
```

Size of blocks in the filesystems:

```
/**
 * Size of one block in the filesystem: could be later in the Legacy
 * definition.
 * <br>JFS2 supports multiple file system block sizes of 512, 1024, 2048,
 * and 4096, with 4 KBytes as standard.
 * <br>EXT3 supports multiple file system block sizes of 1024, 2048 and
 * 4096, with 4 KBytes as standard.
 * <br>NTFS supports sizes of clusters from 512 bytes up to 64 KBytes,
 * with 4 KBytes as standard.
 */
public static final long SIZEBLOCKFS = 4096;
/**
 * Size of one block in the filesystem (double version): could be later in
 * the Legacy definition.
 */
public static final double DSIZEBLOCKFS = 4096.0;
```

If you use JDK6 or greater, you can enable this value and uncomment the following code:

```
/**
 * Is this running under JAVA6 (some optimisation)
 */
public static final boolean ISJAVA6 = false;
/**
 * Get the Global Used and Free space in KB ONLY FROM JDK6
 * @param storage
 * @return the global and used and free space in KB in long[] or null
 */
private static double[] getGlobalUsedFreeSpaceJava6(File storage) {
      double []size = new double[3];
      /* JDK6 ONLY
      size[0] = storage.getTotalSpace();
      size[1] = size[0] - storage.getUsableSpace();
      size[2] = storage.getFreeSpace();
      */
      return size;
}
```

How to extend to a new database support

To extend to another database support (that what I did for PostGreSQL for instance), here are the main points:

1) Create a static value that identify the database resource in LSDConstants.java:

```
/**
 * Type of DB: MySQL
 */
public static final int typeMySQL = 0;
/**
 * Type of DB: Oracle
 */
public static final int typeOracle = 1;
/**
 * Type of DB: PostGreSQL
 */
public static final int typePostGreSQL = 2;
```

2) Create a new directory and class in Client:openlsd.database.data.xxx as the one from mysql or oracle or postgresql (Client:openlsd.database.data.mysql.LSDMySQLSpecific.java)
3) Modify the class that masks the underlying database for specific sql codes (Client:openlsd.database.data.LSDSpecific.java)