



LogicalDOC Architecture

Revision 4.6

Revision 4.6

Doc Type Design

Date 10/14/2009

Written by Marco Meschieri

Verified by Matteo Caruso

Approved by

LogicalDOC LogicalDOC Architecture 4.6

© 2009 Logical Objects snc, via Bonasi 2/A – 41012 Carpi Italy. All rights reserved.
<http://www.logicalobjects.com>

This document is subject to change without notice.

License

This work is licensed under a GNU Free Documentation License 1.2.
<http://www.gnu.org/licenses/fdl-1.2.txt>

Disclaimer

Documentation is provided 'AS IS' and all express or implied conditions, representations and warranties, including any implied warranty of merchantability, fitness for a particular purpose or on-infringement, are disclaimed, except to the extent that such disclaimers are held to be legally invalid.

LogicalDOC LogicalDOC Architecture 4.6

Change Log

Rev.	Date	Changes
4.6	10/14/2009	Updated data model
4.5	05/28/2009	Updated data model and Storer paragraph
4.2	04/24/2009	Added TOC links and footer links
4.1	04/21/2009	Document revision
4.0	10/27/2008	Document creation

Distribution

Person	Company	Position	Contact

Contents

1 APPLICATION LAYERS	5
1.1 <u>DATA LAYER</u>	5
1.2 <u>BUSINESS PROCESS LAYER</u>	7
1.3 <u>BUSINESS ENTITY LAYER</u>	7
1.4 <u>WEB INTERFACE</u>	7
1.5 <u>WEB SERVICES</u>	7
2 SECURITY MODEL	8
3 MODULAR DESIGN	8
3.1 <u>PLUG-IN SYSTEM</u>	8
4 TECHNOLOGIES	10
4.1 <u>SPRING</u>	10
4.2 <u>HIBERNATE</u>	10
4.3 <u>JAVA SERVER FACES</u>	10
5 APPENDIX	10
5.1 <u>DATA MODEL</u>	10

1 Application layers

LogicalDOC is composed of various application layers.

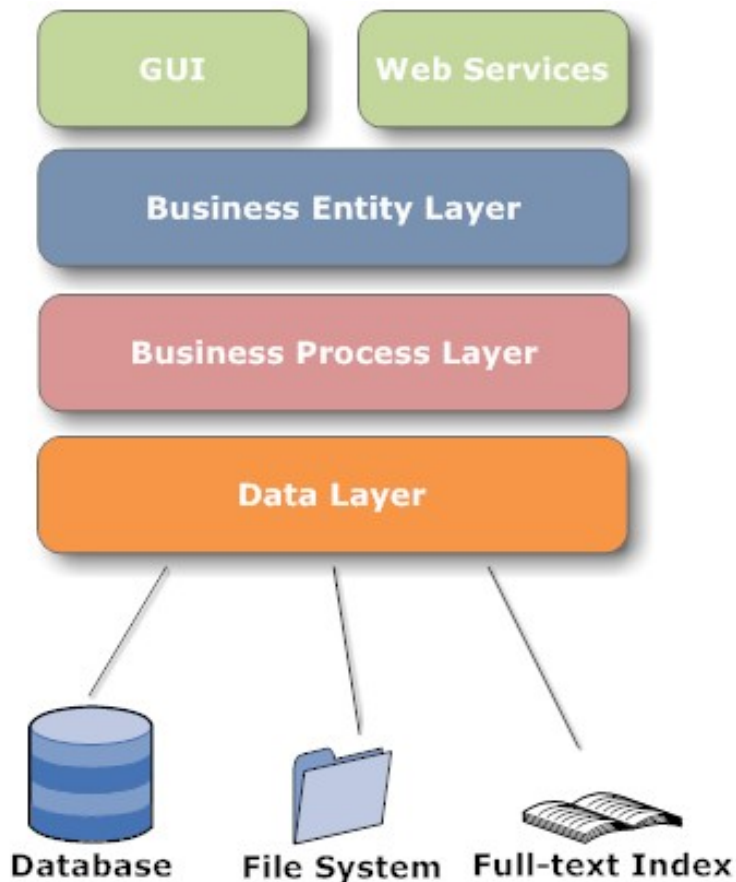


Image 1: LogicalDOC application layers

Layers communicate with each other through the Spring application contexts.

1.1 Data layer

The Data Layer is responsible for resource management. In LogicalDOC DMS, information is stored into three main stores: Relational DB, File System, Full-text Index.

- **Relational DB:** contains all persistent records needed by the program such as users, groups, documents, etc.
- **File System:** all document files are maintained in a disk area organized in a particular layout which is described later
- **Full-text Index:** contains the text extracted from all documents allowing fast searches by content

This layer aims to abstract data handling by handling all physical details regarding storage technologies.

DAOs

DAOs (Data Access Object) are objects used to make CRUD (Create, Read, Update, and Delete) operations on the database in LogicalDOC. For each business entity a proper DAO already exists. For example the business entity *Document* has its *DocumentDAO*.

The purpose of a DAO is to make a given business entity persistent, and to provide finder methods that retrieve persisted instances. DAOs methods are transactional, and this behavior is obtained using Spring's Aspect Oriented features.

Storer

The Storer is the component responsible for document file archiving. It organizes files in a way that optimizes disk space and access time. For each document in LogicalDOC the file system is created considering its id: it is divided in groups of 3 digits and for each group is created a sub-folder. Inside the last one is created a sub-folder named 'doc' containing all document's files. In particular, each version has its own file named as version code (1.0, 1.1 and so on).

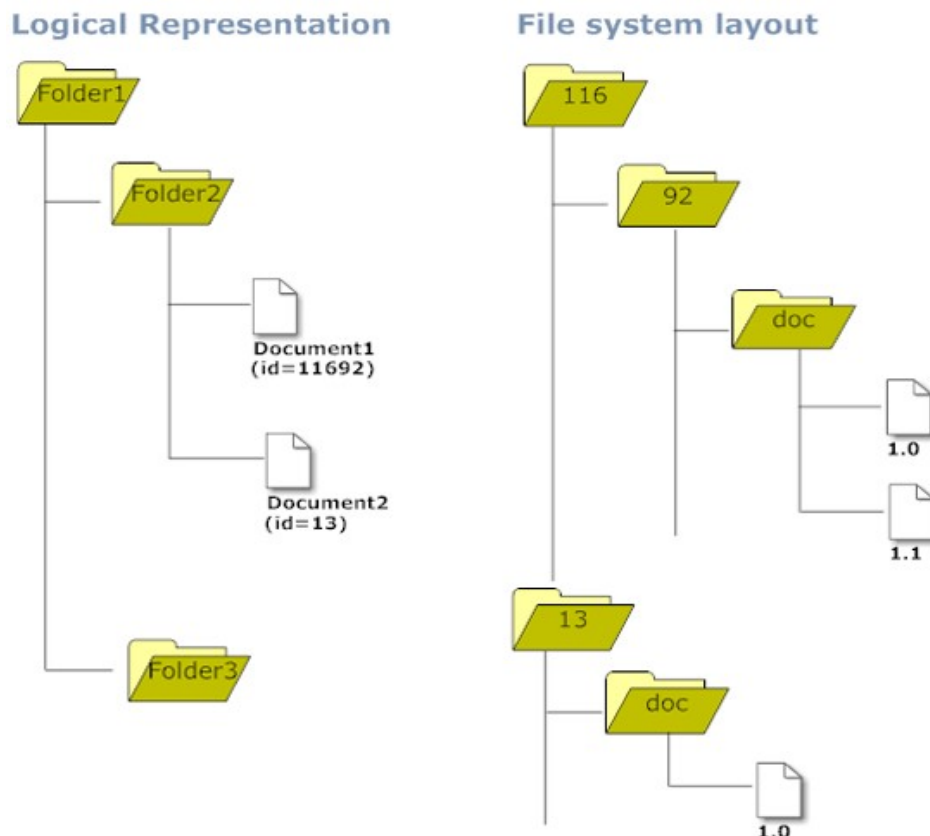


Image 2: File system layout

Search Engine

The search engine module is basically a *facade* on Lucene, and is responsible for the full-text index update as well as full-text searches. For each supported language a dedicated index is maintained, the user can

search on all languages or on a single language only. In the first case all indexes are accessed, and in the second case only the index of the specified language is accessed.

Normally the index is not optimized to speed-up write operations, so a scheduled task takes care of the index optimization. The index optimization minimizes disk consumption and search response time.

1.2 Business Process Layer

The most important processes are identified in this layer which are business entities. These processes are accomplished by managers. A manager is a component that operates on one or more business entities, and accomplishes complex business operations on the entities. For example, the *DocumentManager* implements the check-in process that involves document creation, content indexing, file storage, and many more processes.

Managers use the underlying Data Layer to save/retrieve information to/from the data stores, without knowing anything about implementation issues.

This way managers are completely decoupled from the data store internals.

This layer aims to make a reliable and helpful API (Application Programming Interface) to the upper layers.

1.3 Business Entity Layer

In this level layer there is the static model of the application domain. Each business entity is modeled as a Java Bean that represents a primary class concept. Business entities are for example *Document*, *User*, *Menu*, etc.

An Entity does not contain any business logic since it is implemented by managers, which make business entities very simple objects. Business Entities and Managers create the API that allows the GUI (Graphical User Interface) components to interact with the DMS engine.

1.4 Web Interface

The web interface is the web-application that allows the end-user to interact with LogicalDOC. The GUI of LogicalDOC is completely web-based, and it is implemented using JSF and AJAX technologies.

1.5 Web Services

LogicalDOC can be used as middleware, and can be integrated with other systems through the use of the built-in Web Service. The Web Service module is part of the LogicalDOC core distribution, and it is compliant with W3C specifications SOAP and MTOM.

2 Security model

Following the general product philosophy, the implemented security model is very simple and involves only three entities: *User*, *Group* and *Menu*.

Users can be member of one or more Groups, and read/write permissions can be granted to a Group on a Menu. The Menu entity doesn't just model application menus, it also models other concepts such as the folders where documents are stored. All security policies are expressed on Menus only, so any entity that needs security policies needs to have an associated Menu.

Note: In LogicalDOC security policies can be defined only on folders and not on a single file. This design choice is due to performance considerations.

3 Modular design

LogicalDOC is not a monolithic application, in fact its design is fully modular as the LogicalDOC core is a plug-in. New features can be added by plug-ins.

3.1 Plug-in system

One of the key features of LogicalDOC is its plug-in system. Plug-in is the only means by which the product can be extended. If you want to add features or customize existing ones, you must provide a plug-in. The standard LogicalDOC distribution comprises various plug-ins such as the logicaldoc-core and the logicaldoc-webservice. Below is a list of the most important plug-ins for the standard distribution:

Plug-in	Description
logicaldoc-core	core API
logicaldoc-webservice	WebService implementation
logicaldoc-webdav	WebDAV support

Optional plug-ins include:

Plug-in	Description
logicaldoc-email	downloads documents from mail server
logicaldoc-language-pt	support for portuguese language
logicaldoc-language-nl	support for dutch language

Extension points

Each plug-in can define one or more extension points that can be extended by other plug-ins. For example, the extension point *Language* defined by the *logicaldoc-core* plug-in is extended by the *logicaldoc-lang-pt* plug-in in order to add support for the Portuguese language.

Contributions to the platform

A plug-in can add several functionalities to the platform. This is only a brief list of the most important functionalities:

- Extension of existing extension points from other plug-ins
- Definition of new extension points
- Business entities
- Managers
- i18n translations
- Static resources
- Libraries and technologies

Plug-in archive

A plug-in is packaged in a compressed zip archive called the plug-in archive. The structure of a plug-in archive is very simple.

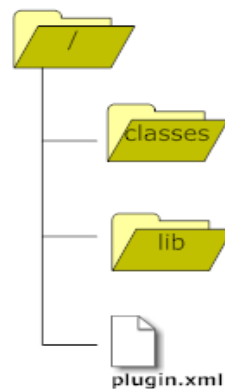


Image 3: Plug-in archive

- `classes/`: contains all the plug-in classes as well as resources that are business entities, managers, etc.
- `lib/`: contains the needed dependencies (.jar files)
- `plugin.xml`: This is an XML file of the plug-in descriptors that define extension point definitions

4 Technologies

LogicalDOC leverages the best of the breed Java technologies. In this chapter we will introduce the most important technologies and describe how they are used in the LogicalDOC platform.

4.1 Spring

All DAOs and Managers among the many others fall into a Spring Application Context. Each plug-in can define its own application context simply by placing a resource named *context.xml* in the class path containing all bean definitions. At the startup, all class path resources named *context-*.xml* are collected in order to setup the big application context in which all DAOs and Managers defined in each plug-in can reference the DAOs and Managers defined in the other plug-ins. In other words, all beans defined in a specific plug-in are immediately usable by all other plug-ins.

4.2 Hibernate

LogicalDOC DAOs are implemented using the Hibernate ORM (Object Relational Mapping) framework. Hibernate abstracts the Data Layer from JDBC issues and alleviates development efforts. Each plug-in can contribute new persisted entities simply by placing the **.hbm.xml* file in the class path mapping. At the startup, all class path resources named **.hbm.xml* are considered mapping files which are then passed to the Hibernate configuration. Each mapping file is used by Hibernate to persist a Business Entity, e.g. the mapping *Document.hbm.xml* contains mapping informations for the entity *Document*. As a general rule for the naming of a mapping file for an entity, use *<EntityName>.hbm.xml*.

4.3 Java Server Faces

The user interface of LogicalDOC is based on JSF(Java Server Faces), utilizing the JSF implementation called IceFaces. The main configuration file for the managed beans is the *WEB-INF/faces-config.xml*, but each plug-in can contribute a new managed bean simply by declaring them as a simple bean inside the Spring application context.

5 Appendix

5.1 Data model

Physical Data Model

Model: LogicalIDOC

Package:

Diagram: Data Model

Author: Matteo - Logical Objects

Version: 4.6

Id_template		
Id_id	bigint	<pk>
Id_lastmodified	timestamp	
Id_deleted	int	
Id_name	varchar(255)	
Id_description	varchar(2000)	

Id_template_ext		
Id_templateid	bigint	<pk.fk>
Id_mandatory	int	
Id_type	int	
Id_stringvalue	varchar(4000)	
Id_intvalue	int	
Id_doublevalue	float	
Id_datevalue	timestamp	
Id_name	varchar(255)	<pk>

Id_systemmessage		
Id_id	bigint	<pk>
Id_lastmodified	timestamp	
Id_deleted	int	
Id_author	varchar(255)	
Id_recipient	varchar(255)	
Id_messagetext	varchar(2000)	
Id_subject	varchar(255)	
Id_sentdate	timestamp	
Id_datescope	int	
Id_prio	int	
Id_confirmation	int	
Id_red	int	

Id_dthread		
Id_id	bigint	<pk>
Id_lastmodified	timestamp	
Id_deleted	int	
Id_docid	bigint	
Id_creation	timestamp	
Id_creatorid	bigint	
Id_creatormame	varchar(255)	
Id_lastpost	timestamp	
Id_subject	varchar(255)	
Id_replies	int	
Id_views	int	

Id_dcomment		
Id_threadid	bigint	<pk.fk>
Id_replyto	int	
Id_replypath	varchar(255)	
Id_usend	bigint	
Id_username	varchar(255)	
Id_date	timestamp	
Id_subject	varchar(255)	
Id_body	varchar(4000)	
Id_deleted	int	
Id_id	int	<pk>

Id_menugroup		
Id_menuid	bigint	<pk.fk1>
Id_groupid	bigint	<pk.fk2>
Id_write	int	<pk>
Id_addchild	int	<pk>
Id_managesecurity	int	<pk>
Id_manageimmutability	int	<pk>
Id_delete	int	<pk>
Id_rename	int	<pk>
Id_bulkimport	int	<pk>
Id_bulkexport	int	<pk>
Id_sign	int	<pk>
Id_archive	int	<pk>
Id_workflow	int	<pk>

Id_menu		
Id_id	bigint	<pk>
Id_lastmodified	timestamp	
Id_deleted	int	
Id_text	varchar(255)	
Id_parentid	bigint	<fk>
Id_sort	int	
Id_icon	varchar(255)	
Id_path	varchar(255)	
Id_pathextended	varchar(4000)	
Id_type	int	
Id_ref	varchar(255)	
Id_size	bigint	

Id_link		
Id_id	bigint	<pk>
Id_lastmodified	timestamp	
Id_deleted	int	
Id_type	varchar(255)	
Id_docid1	bigint	<fk2>
Id_docid2	bigint	<fk1>

Id_generic		
Id_id	bigint	<pk>
Id_lastmodified	timestamp	
Id_deleted	int	
Id_type	varchar(255)	
Id_subtype	varchar(255)	
Id_string1	varchar(4000)	
Id_string2	varchar(4000)	
Id_integer1	int	
Id_integer2	int	
Id_double1	float	
Id_double2	float	
Id_date1	timestamp	
Id_date2	timestamp	

Id_generic_ext		
Id_genid	bigint	<pk.fk>
Id_mandatory	int	
Id_type	int	
Id_stringvalue	varchar(4000)	
Id_intvalue	int	
Id_doublevalue	float	
Id_datevalue	timestamp	
Id_name	varchar(255)	<pk>

Id_group		
Id_id	bigint	<pk>
Id_lastmodified	timestamp	
Id_deleted	int	
Id_name	varchar(255)	
Id_description	varchar(255)	
Id_type	int	

Id_document		
Id_id	bigint	<pk>
Id_lastmodified	timestamp	
Id_deleted	int	
Id_immutable	int	
Id_customid	varchar(4000)	
Id_title	varchar(255)	
Id_version	varchar(10)	
Id_fileversion	varchar(10)	
Id_date	timestamp	
Id_creation	timestamp	
Id_publisher	varchar(255)	
Id_publisherid	bigint	
Id_creator	varchar(255)	
Id_creatorid	bigint	
Id_status	int	
Id_type	varchar(255)	
Id_lockuserid	bigint	
Id_source	varchar(255)	
Id_sourceauthor	varchar(255)	
Id_sourceauthorid	bigint	
Id_sourcecreate	timestamp	
Id_sourceid	varchar(4000)	
Id_sourcetype	varchar(255)	
Id_object	varchar(4000)	
Id_coverage	varchar(255)	
Id_language	varchar(10)	
Id_filename	varchar(255)	
Id_filesize	bigint	
Id_indexed	int	
Id_signed	int	
Id_digest	varchar(255)	
Id_recipient	varchar(4000)	<fk2>
Id_folderid	bigint	<fk1>
Id_templateid	bigint	<fk1>
Id_exportstatus	int	
Id_exportid	bigint	
Id_exportname	varchar(255)	
Id_exportversion	varchar(10)	

Id_document_ext		
Id_docid	bigint	<pk.fk>
Id_mandatory	int	
Id_type	int	
Id_stringvalue	varchar(4000)	
Id_intvalue	int	
Id_doublevalue	float	
Id_datevalue	timestamp	
Id_name	varchar(255)	<pk>

Id_group_ext		
Id_groupid	bigint	<pk.fk>
Id_mandatory	int	
Id_type	int	
Id_stringvalue	varchar(4000)	
Id_intvalue	int	
Id_doublevalue	float	
Id_datevalue	timestamp	
Id_name	varchar(255)	<pk>

Id_usergroup		
Id_groupid	bigint	<pk.fk1>
Id_userid	bigint	<pk.fk2>

Id_ticket		
Id_id	bigint	<pk>
Id_lastmodified	timestamp	
Id_deleted	int	
Id_ticketid	varchar(255)	<fk1>
Id_docid	bigint	<fk2>
Id_userid	bigint	<fk2>

Id_userdoc		
Id_id	bigint	<pk>
Id_lastmodified	timestamp	
Id_immutable	int	
Id_deleted	int	
Id_docid	bigint	<fk2>
Id_userid	bigint	<fk1>
Id_date	timestamp	

Id_history		
Id_id	bigint	<pk>
Id_lastmodified	timestamp	
Id_deleted	int	
Id_docid	bigint	<fk1>
Id_userid	bigint	<fk2>
Id_date	timestamp	
Id_username	varchar(255)	
Id_event	varchar(255)	
Id_comment	varchar(4000)	
Id_version	varchar(10)	
Id_title	varchar(255)	
Id_path	varchar(4000)	

Id_tag		
Id_docid	bigint	<fk>
Id_tag	varchar(255)	

Id_version_ext		
Id_versionid	bigint	<pk.fk>
Id_mandatory	int	
Id_type	int	
Id_stringvalue	varchar(4000)	
Id_intvalue	int	
Id_doublevalue	float	
Id_datevalue	timestamp	
Id_name	varchar(255)	<pk>

Id_user		
Id_id	bigint	<pk>
Id_lastmodified	timestamp	
Id_deleted	int	
Id_enabled	int	
Id_username	varchar(255)	
Id_password	varchar(255)	
Id_name	varchar(255)	
Id_firstname	varchar(255)	
Id_street	varchar(255)	
Id_postalcode	varchar(255)	
Id_city	varchar(255)	
Id_country	varchar(255)	
Id_state	varchar(255)	
Id_language	varchar(10)	
Id_email	varchar(255)	
Id_telephone	varchar(255)	
Id_telephone2	varchar(255)	
Id_type	int	
Id_passwordchanged	timestamp	
Id_passwordexpires	int	

Id_version		
Id_id	bigint	<pk>
Id_lastmodified	timestamp	
Id_deleted	int	
Id_immutable	int	
Id_customid	varchar(4000)	
Id_title	varchar(255)	
Id_version	varchar(10)	
Id_fileversion	varchar(10)	
Id_date	timestamp	
Id_creation	timestamp	
Id_publisher	varchar(255)	
Id_publisherid	bigint	
Id_creator	varchar(255)	
Id_creatorid	bigint	
Id_status	int	
Id_type	varchar(255)	
Id_lockuserid	bigint	
Id_source	varchar(255)	
Id_sourceauthor	varchar(255)	
Id_sourceauthorid	bigint	
Id_sourcecreate	timestamp	
Id_sourceid	varchar(4000)	
Id_sourcetype	varchar(255)	
Id_object	varchar(4000)	
Id_coverage	varchar(255)	
Id_language	varchar(10)	
Id_filename	varchar(255)	
Id_filesize	bigint	
Id_indexed	int	
Id_signed	int	
Id_digest	varchar(255)	
Id_recipient	varchar(4000)	<fk1>
Id_folderid	bigint	<fk2>
Id_foldename	varchar(4000)	
Id_templateid	bigint	
Id_templateiname	varchar(4000)	
Id_tgs	varchar(4000)	
Id_username	varchar(255)	
Id_userid	bigint	<fk1>
Id_versiondate	timestamp	
Id_comment	varchar(4000)	
Id_event	varchar(255)	
Id_documentid	bigint	<fk2>
Id_exportstatus	int	
Id_exportid	bigint	
Id_exportname	varchar(255)	
Id_exportversion	varchar(10)	

