LogicalDOC™
Document Management System

Developer Manual

Revision 1.4

Revision	1.4
Doc Type	Development Practices
Date	April 24, 2009
Written by	Marco Meschieri (Logical Objects)
Verified by	
Approved by	

LogicalDOC Developer Manual 1.4

License

Disclaimer

LogicalDOC Developer Manual 1.4

Change Log

Rev.	Date	Changes
1.0	15/09/2008	Document creation
1.1	04/10/2008	Modifications on chapter §1
1.2	22/01/2009	Changed project layout (see §2.6)
1.3	23/04/2009	Updated spelling & grammar
1.4	24/04/2009	Added links to the TOC

Distribution

Person	Company	Position	Contact

Contents

1 Prerequisites

In order to setup a LogicalDOC CE (Community Edition) development environment some tools are required, and must be installed on to the computer system.

1.1 JDK

All LogicalDOC artifact are developed in Java requiring JDK 1.6 to be installed on the computer system. To verify the version of Java installed on the computer system, open a shell, and type: ***java -version***. If Java has been correctly installed on the computer system a line should appear that reads: **java version "1.6.0_07"**. Make sure that the environment variable JAVA_HOME points to the root of the JDK.

1.2 Ant

Ant 1.7 or greater is required on the computer system. To verify the version of Ant installed on the computer system, open a shell, and type: ***ant -version***. If ant is correctly installed on the computer system a line should appear that reads: **Apache Ant version 1.7.0 compiled on December 13 2006**. Make that the environment variable ANT_HOME points to the root of the Ant distribution.

For more information about Ant reference: http://ant.apache.org/

1.3 Maven

Maven 2.0.9 or greater is required on the computer system. To verify the version of Maven installed, open a shell, and type: ***mvn –version***. If maven is correctly installed on the computer system a line should appear that reads: **Maven version: 2.0.9**. Make sure that the environment variable M2_HOME points to the root of the Maven distribution. Some automated tests require a lot of memory, so create an environment variable: **MAVEN_OPTS:**
MAVEN_OPTS=-Xmx1024m -XX:PermSize=128m -XX:MaxPermSize=128m

For more information about Maven reference: http://maven.apache.org/

1.4 Eclipse

The LogicalDOC development team supports only the Eclipse IDE 3.3 or greater.

Eclipse IDE can be download from: http://www.eclipse.org/downloads/

1.5 SVN client

A SVN Client is required to be installed on the computer system to track versioning. To verify the version information about the SVN Client, open a shell, and type: ***svn --version***
If SVN is installed on the computer system a description should appear that reads similar to:

```
svn, version 1.6.0 (dev build)
compiled Sep 15 2008, 00:11:13
Copyright (C) 2000-2008 CollabNet.
```

For Windows the recommended SVN Client is TortoiseSVN, which can be downloaded from: http://nightlybuilds.tortoisesvn.net/latest/win32/

For more information about TortoiseSVN reference: http://tortoisesvn.net

1.6 Apache Tomcat

LogicalDOC is certified to work with Apache Tomcat 6.0.x. Download a recent distribution of the application server. LogicalDOC needs at least 512MB at runtime, edit the catalina.bat and/or catalina.sh. Edit the setup variable ***JAVA_OPTS=****-Xmx512m*

For more information about Apache Tomcat reference: http://tomcat.apache.org

1.7 Open Office

To write documentation about LogicalDOC install OpenOffice 2.4 or greater. All LogicalDOC documentation is written using OpenOffice. A set of template models to be used for document creation are provided in the SVN repository.

For more information about OpenOffice reference: http://www.openoffice.org/

2 Workspace Setup

The first task is to create an Eclipse workspace for logicaldoc, the workspace is a folder containing all development resources. Open a shell in the home directory of the computer system, and type the following command:

```
mvn -Declipse.workspace=workspace-logicaldoc eclipse:configure-workspace
```

This command will create the Eclipse workspace in the 'workspace-logicaldoc' directory, and this workspace comes up with the M2_REPO class path variable already configured.

2.1 Checkout a working copy of LogicalDOC CE

Open a shell in the workspace directory previously created and type the following commands:

```
svn --depth files co https://logicaldoc.svn.sourceforge.net/svnroot/logicaldoc/community
community
cd community
checkout.bat
cd..
```

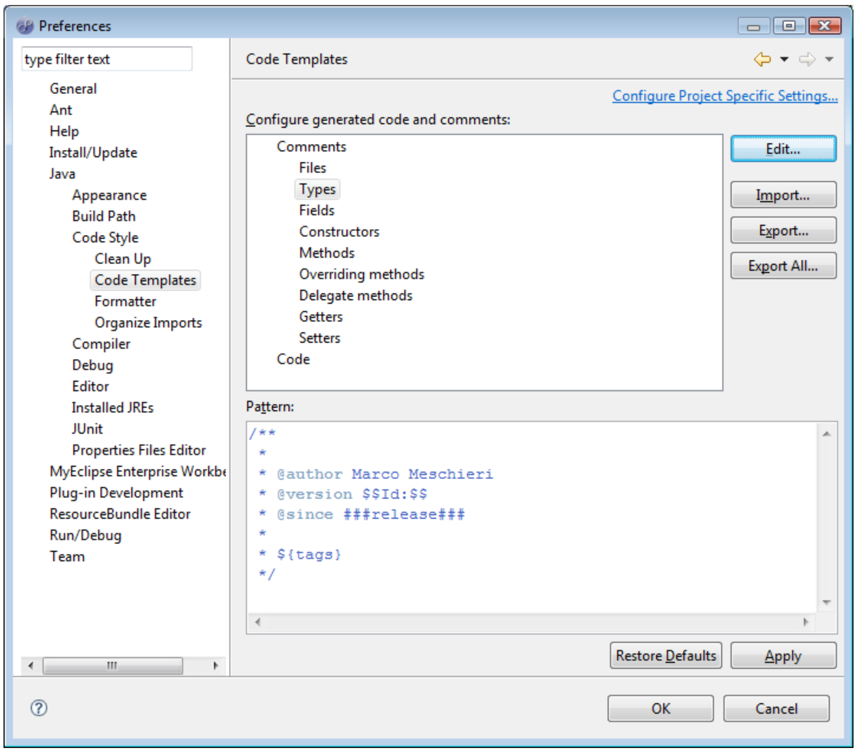
```
svn --depth files co https://logicaldoc.svn.sourceforge.net/svnroot/logicaldoc/build build
cd build
checkout.bat
cd..
```

```
svn --depth files co https://logicaldoc.svn.sourceforge.net/svnroot/logicaldoc/docs docs
cd docs
checkout.bat
cd..
```

Note: On Linux use checkout.sh instead of checkout.bat

2.2 Import code templates

Launch Eclipse and import logicaldoc code templates.

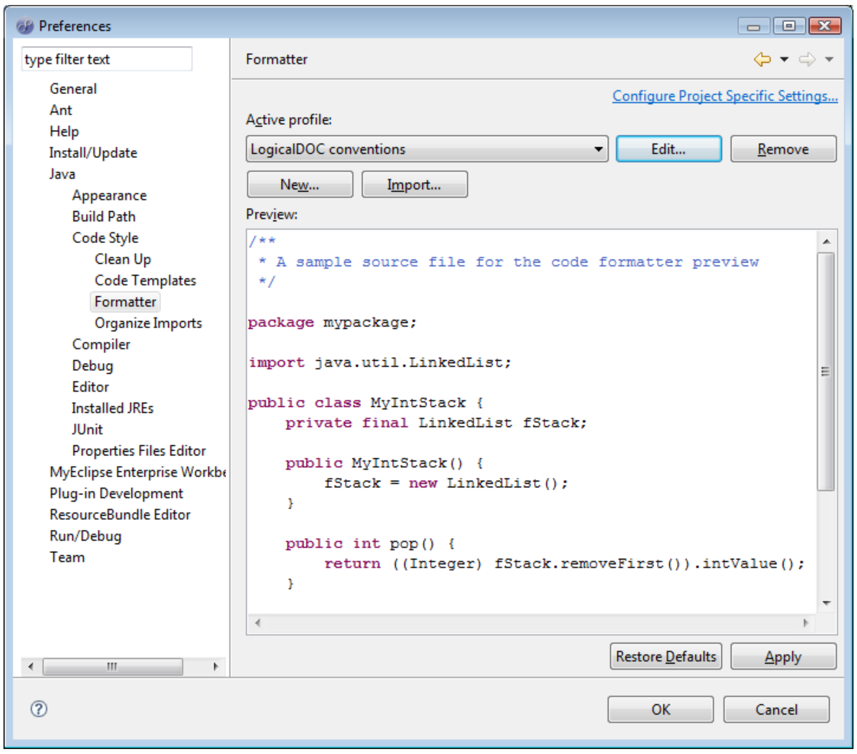


```
/**
 *
 * @author Marco Meschieri
 * @version $$Id:$$
 * @since ###release###
 *
 * ${tags}
 */
```

Figure 1: Import of code templates

Open the Preferences window, select Java→Code Style→Code Template. Then click the Edit button, and import the file workspace-logicaldoc/docs/devel/logicaldoc-codetemplates.xml.

2.3 Import coding conventions



Active profile: LogicalDOC conventions

```
/**
 * A sample source file for the code formatter preview
 */

package mypackage;

import java.util.LinkedList;

public class MyIntStack {
    private final LinkedList fStack;

    public MyIntStack() {
        fStack = new LinkedList();
    }

    public int pop() {
        return ((Integer) fStack.removeFirst()).intValue();
    }
```

Figure 2: Import of coding conventions

From the Preferences window import the coding conventions, select *Java→Code Style→Formatter.* Click on the ***Edit*** button, import the file workspace-logicaldoc/docs/devel/logicaldoc-conventions.xml

2.4 Setup SVN connection

In order for SVN to operate with eclipse the SubEclipse plug-in must be installed. This can be done through adding the update site: http://subclipse.tigris.org/update_1.4.x

Instructions on how to install the SubEclipse plug-in can be referenced here: http://subclipse.tigris.org/install.html

Once the SubEclipse plug-in is installed, open the **SVN Repository Exploring** perspective, then from the **SVN Repositories** view right click , select *New→Repository Location.* In the dialog box type the URL: https://logicaldoc.svn.sourceforge.net/svnroot/logicaldoc
Then confirm by clicking the ***Finish*** button. This is the remote repository all resources will be committed against.

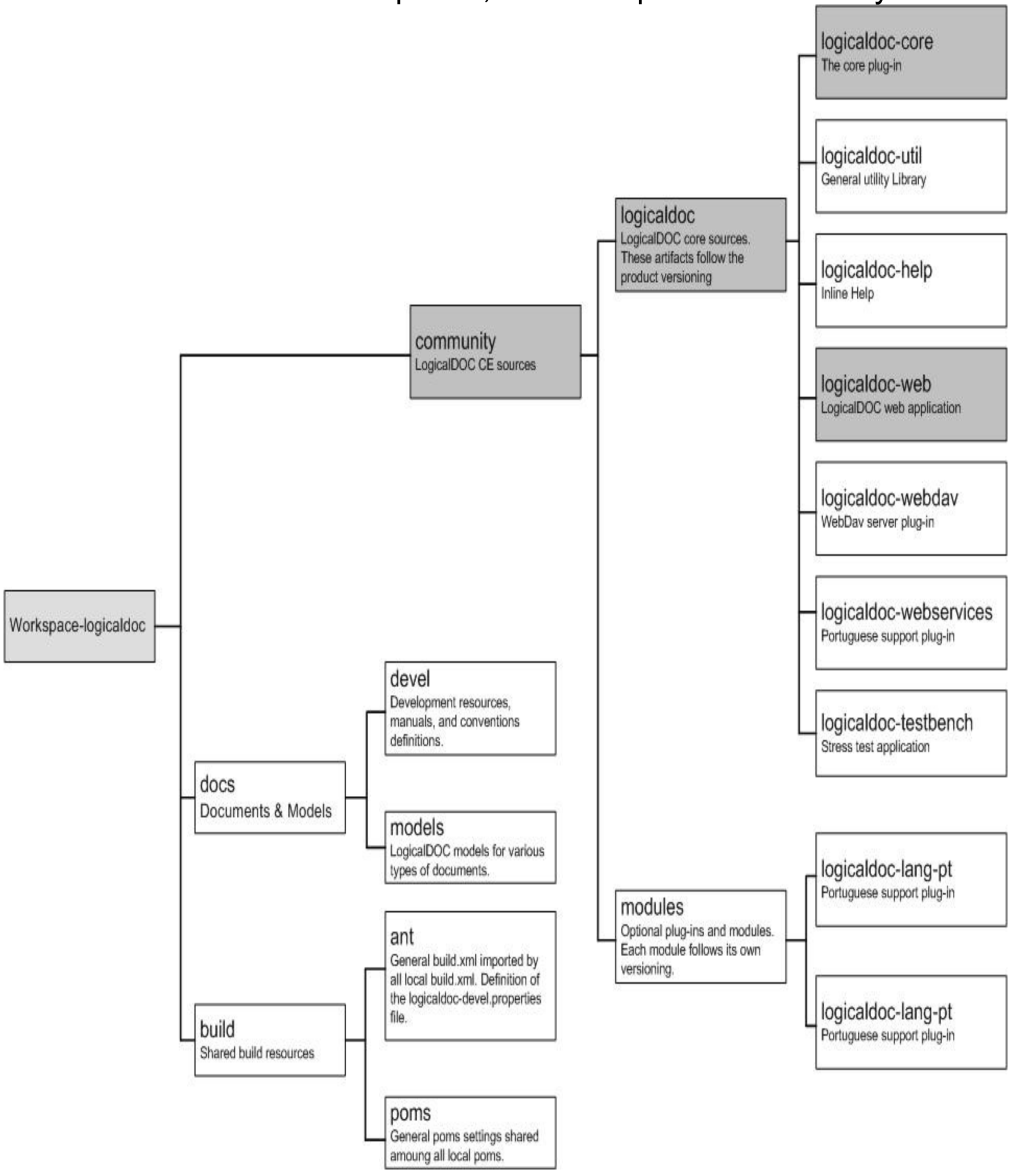
2.5 Configure the logicaldoc-devel.properties

After the checkout, copy the file workspace-logicaldoc/build/ant/logicaldoc-devel.properties into the home directory. This file contains various settings used during builds. Two properties are required for the setup:

- *logicaldoc.devroot* must point to the logicaldoc workspace director
- *logicaldoc.webroot* must point to the development webapp installation

2.6 Workspace structure

Once the checkout is completed, the workspace structure layout is as follows:



Workspace-logicaldoc
community
LogicalDOC CE sources
logicaldoc
LogicalDOC core sources. These artifacts follow the product versioning
logicaldoc-core
The core plug-in
logicaldoc-util
General utility Library
logicaldoc-help
Inline Help
logicaldoc-web
LogicalDOC web application
logicaldoc-webdav
WebDav server plug-in
logicaldoc-webservices
Portuguese support plug-in
logicaldoc-testbench
Stress test application
modules
Optional plug-ins and modules. Each module follows its own versioning.
logicaldoc-lang-pt
Portuguese support plug-in
logicaldoc-lang-pt
Portuguese support plug-in
docs
Documents & Models
devel
Development resources, manuals, and conventions definitions.
models
LogicalDOC models for various types of documents.
build
Shared build resources
ant
General build.xml imported by all local build.xml. Definition of the logicaldoc-devel.properties file.
poms
General poms settings shared amoung all local poms.

Figure 3: LogicalDOC CE workspace layout

2.7 Import projects

To import LogicalDOC modules into the eclipse workspace projects, open a shell into *workspace-logicaldoc/community/logicaldoc,* type **mvn eclipse:eclipse**. This will create all needed eclipse project files.

Now start eclipse and import all projects *File→Import→Existing Projects Into Workspace*

2.8 Compile LogicalDOC

To compile LogicalDOC open a shell to the workspace, type **mvn -Dmaven.test.skip=true install**. The WAR file will be created in the directory /logicaldoc-web/target. Make sure this directory has the POM.xml file and the logicaldoc-devel.properties file in the directory.

3 Best practices

3.1 Coding conventions

The following guidelines should be followed for contributing to LogicalDOC

- All classes and interfaces must be created in a package directly or indirectly contained in com.logicaldoc.
- All classes and interfaces must be formatted using logicaldoc conventions defined in the file *logicaldoc-conventions.xml*.
- All classes and interfaces must have a javadoc comment with the description of the class or interface. The author full name, the program version in which the resource has been created and the current revision number (import the file *logicaldoc-codetemplates.xml*) must be present in the comment.
- All public methods (with exclusion of banal getters/setters) must be commented with a javadoc. It is also recommended to comment even private or protected methods.
- The source code, must be clear and properly commented, so insert comments inside the code.
- Comments or documentation can be written in any language other than English.

3.2 Modularization

The LogicalDOC sources are subdivided into modules for ease of development, each module is named logicaldoc-***name***. For example the core module is called ***logicaldoc-core***.

Each module metadata is stored in it's local pom.xml. A module can produce one or more artifacts. The name of the artifact is a composition of the module name and version: ***moduleName***-***version***.***type***. An example of an artifact name: ***logicaldoc-core-3.6.0.jar***, and ***logicaldoc-web-3.6.0.war***.

3.3 Versioning

All LogicalDOC artifact are versioned using three numbers following the pattern: ***majorVersion.minorVersion.microVersion***. Increases in the major version indicate heavy changes, such as database alterations or the introduction of very important new features. Increases in the minor version indicate relevant changes, such as new features or modifications of existing features. The micro version is used for bug fixes. The micro version number can be omitted, and in this case it is considered a value of zero. The major and minor version must always be specified even if the value is zero. Examples of valid versions: ***3.6.0***, ***3.6***

All modules inside the community/logicaldoc/ directory share the same version number since they follow the product versioning. All other modules, even those in the community/modules/ are versioned in an independent way.