Installation et utilisation de la BD eXist

Type du document	Procédure d'installation
Auteur(s)	Eric Bouladier
Date de création	26/02/20007
Domaine de diffusion	tous
Validé par	Equipe

Versions	***Date***	***Auteur(s)***	***Modifications***
V1.00	26/02/2007	Eric Bouladier	Création
V1.01	28/02/2007	Eric Bouladier	Problème validation fichier par schéma (bug du parseur utilisé par eXist).

Table des matières

Téléchargement

Le site utilisé est : http://exist.sourceforge.net/index.html

A la date de rédaction de ce document, deux versions de la base de données sont disponibles : les versions 1.0 et 1.1. Elles se distinguent principalement par leur moteur d'indexation. La version 1,1 est plus souple d'utilisation. C'est cette dernière version que nous choisissons bien qu'elle soit déclarée comme moins finalisée. Comme l'API est indépendante de la version (à quelque chose près), il serait aisé de passer à la version 1.1 en cas de difficulté.

L base de données eXist propose un déploiement en base autonome ou installée dans un moteur de servlet tel que Tomcat ou Jetty. Cette dernière option n'est pas retenue puisque le serveur de présentation (faisant tourner Tomcat) n'abrite pas la base de données.

La version téléchargée est donc : eXist-1.1.1-newcore.jar

Pré requis

La version 1.4.2 au minimum du JDK Java doit être installée sur le serveur faisant tourber eXist (attention, il s'agit bien du JDK : une JRE n'est pas suffisante). Il est préférable que la variable d'environnement JAVA_HOME soit présente.

Installation

Sous Windows, il suffit de double-cliquer sur le fichier téléchargé pour obtenir un enchaînement d'écrans ne présentant aucune difficulté particulière de compréhension.

Configuration de la base de données

La base de données lancée comme serveur autonome utilise le conteneur de Servlet en mode dégradé sur le port 8080. Ce port doit être modifié s'il est déjà utilisé par un autre logiciel (Tomcat par exemple). Pour changer le port, modifier le fichier jetty.xml du répertoire C:\Program Files\eXist\tools\jetty\etc.

La variable d'environnement EXIST_HOME peut être définie (pas forcément indispensable) pour contenir le chemin du répertoire d'installation (C:\Program Files\eXist dans notre exemple).

Les fichiers conf.xml et server.xml de C:\Program Files\eXist contiennent les paramètres de configuration. Dans conf.xml, passer le paramètre validation= «auto» à «no». Cela empèche eXist d'effectuer une vérification du fichier xml lors de l'ajout du fichier dans la base de données. C'est nécessaire car un bug dans le parseur utilisé par eXist rend inutilisable cette option avec un schéma xml. Dans server.xml, modifier si nécessaire le profile et le mot de passe de l'administrateur.

Lancement en mode serveur autonome

Pour lancer eXist en mode serveur autonome, il faut utiliser le fichier server.bat du répertoire C:\Program Files\eXist\bin.

L'arrêt de la base de données peut s'effectuer par un Ctrl C dans la fenêtre DOS résultant de l'exécution de server.bat.

La fenêtre DOS se présente comme ceci à l'issue du lancement.

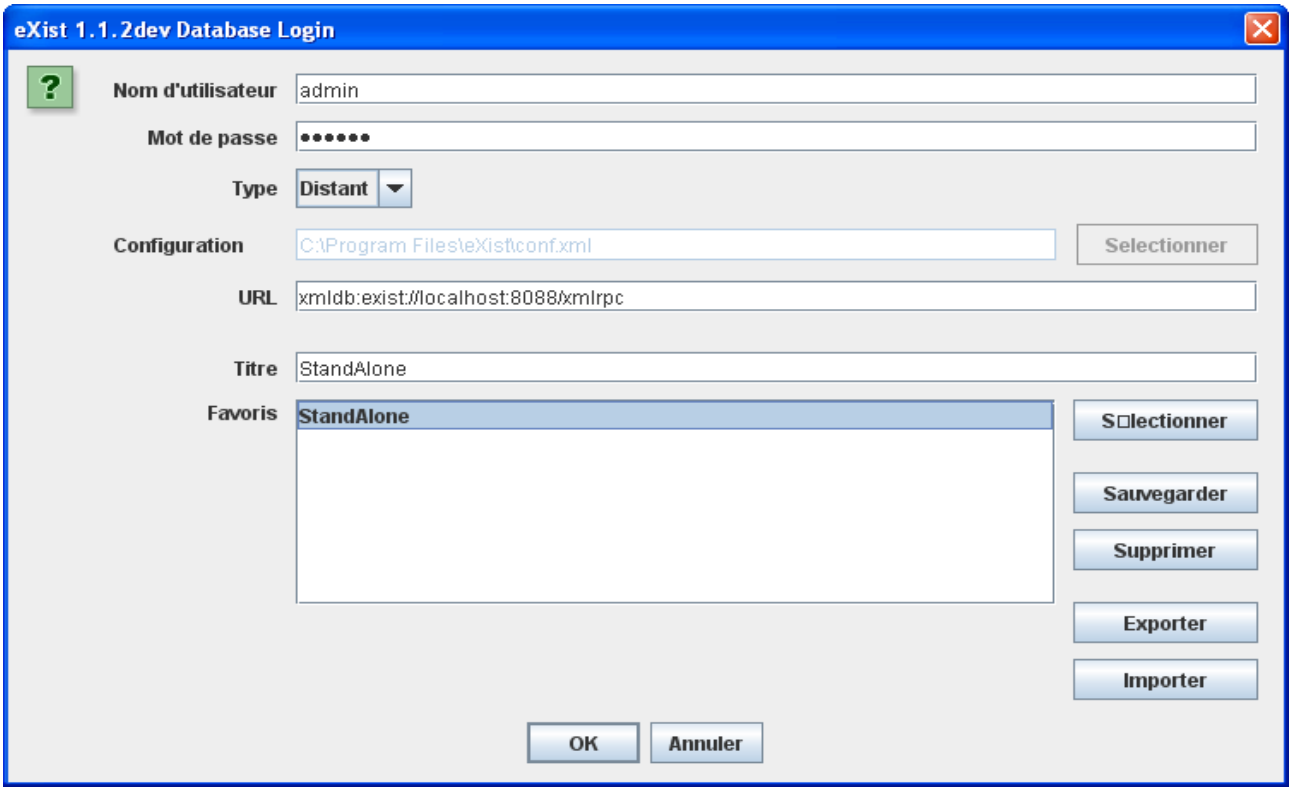
```
server.bat
eXist version 1.0, Copyright (C) 2005 The eXist Project
eXist comes with ABSOLUTELY NO WARRANTY.
This is free software, and you are welcome to redistribute it
under certain conditions; for details read the license file.

Reading server configuration from: C:\Program Files\eXist\server.xml
Loading configuration from C:\Program Files\eXist\conf.xml
Found 1 catalog entries.
Loading catalog: C:\Program Files\eXist\webapp\WEB-INF\catalog.xml
26 f├®vr. 2007 15:42:38,015 [main] INFO  (HttpServer.java [start]:654) - Startin
g Jetty/5.0.0
26 f├®vr. 2007 15:42:38,062 [main] INFO  (HttpContext.java [start]:1604) - Start
ed HttpContext[/]
26 f├®vr. 2007 15:42:38,078 [main] INFO  (SocketListener.java [start]:204) - Sta
rted SocketListener on 0.0.0.0:8088
26 f├®vr. 2007 15:42:38,078 [main] INFO  (HttpServer.java [start]:690) - Started
 org.mortbay.http.HttpServer@419d05

Server launched ...
Installed services:
---------------------------------------------
webdav: localhost:8088/webdav/*
xmlrpc: localhost:8088/xmlrpc/*
rest:   localhost:8088/*
```

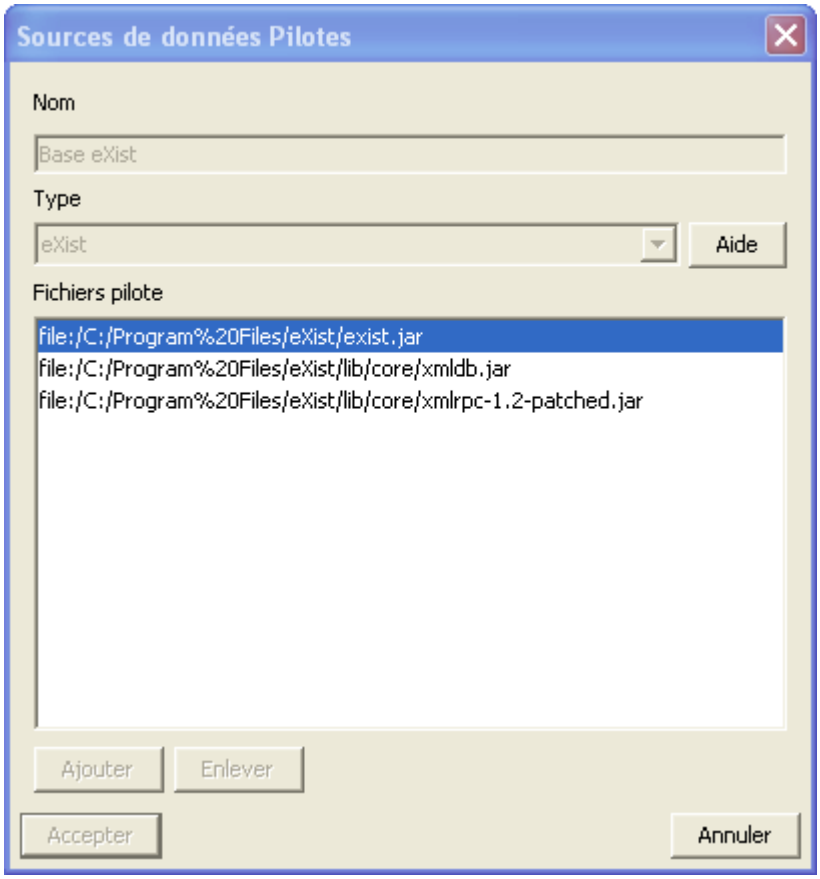
Paramétrage du client eXist

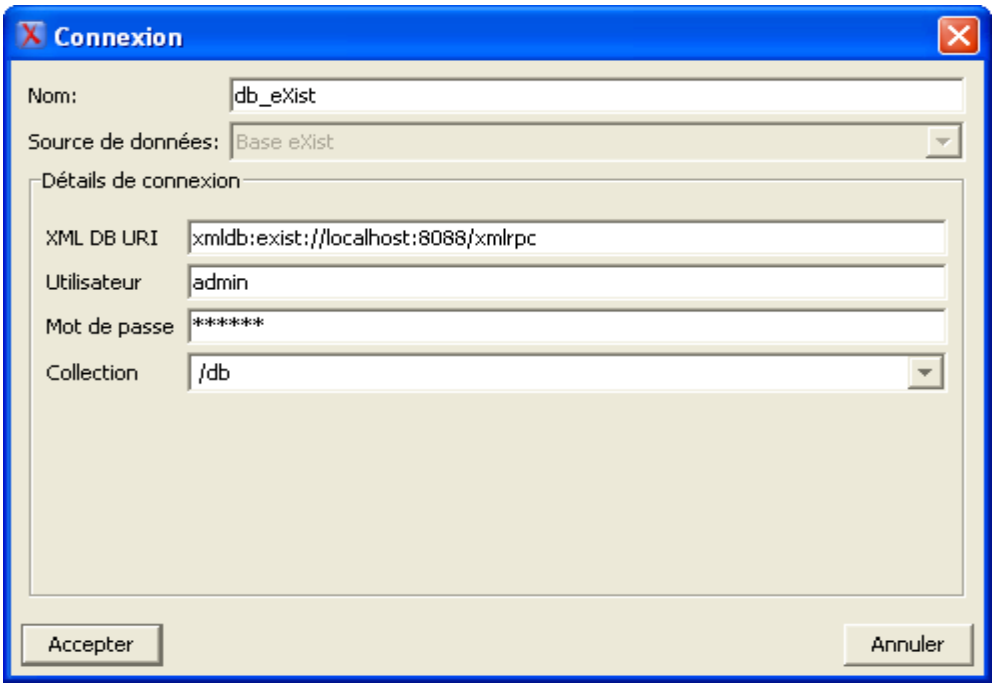
Le paramétrage de la connexion dans le client eXist est le suivant (noter l'URL qui n'est pas celle proposée par défaut) :



Paramétrage de l'outil oXygen xml Editor

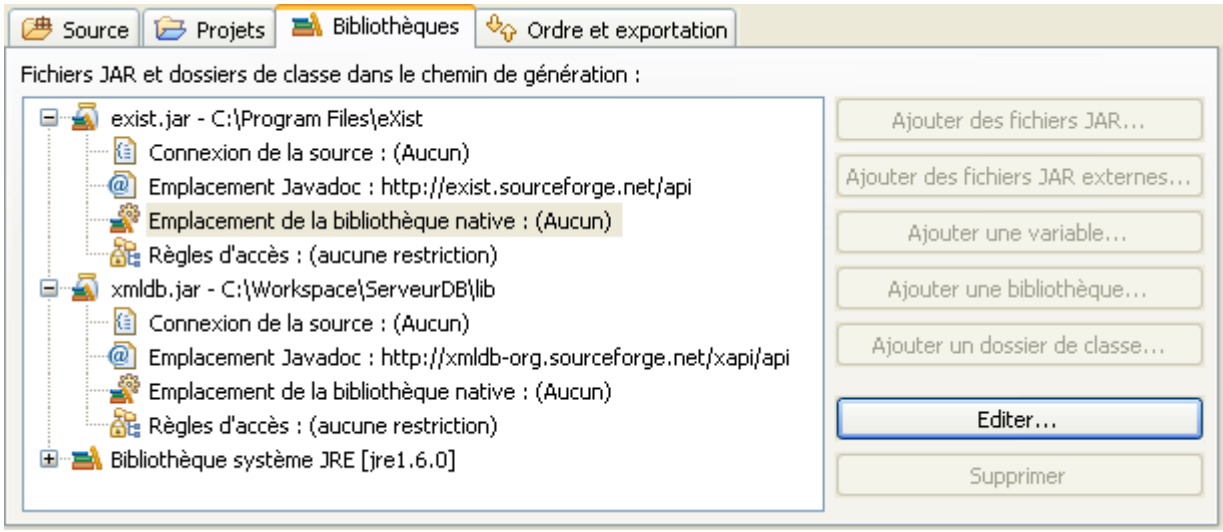
Dans l'outil Oxygene XML Editor, il faut créer une source de données et une connexion telles que montrées ci-dessous:



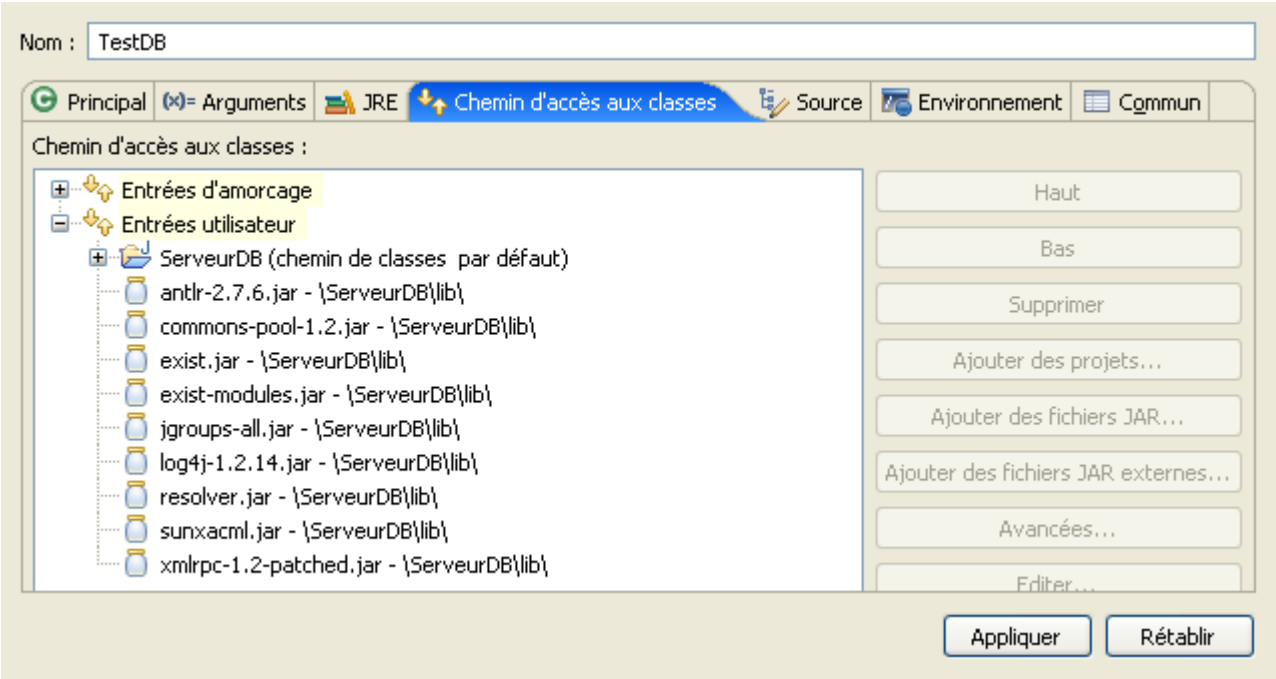


Configuration d'un projet Eclipse attaquant la BD eXist

Les fichiers jar suivants doivent être incorporés dans le chemin de génération. L'emplacement de la javadoc est indiqué : profitez en.



Les fichiers jar suivants doivent être incorporés dans le chemin d'accès aux classes (exécution).



Exemple en Java

```
package serv_eXist;

import java.io.File;

import javax.xml.transform.OutputKeys;

import org.exist.xmldb.XQueryService;
import org.w3c.dom.Document;
import org.xmldb.api.DatabaseManager;
import org.xmldb.api.base.Collection;
import org.xmldb.api.base.CompiledExpression;
import org.xmldb.api.base.Database;
import org.xmldb.api.base.Resource;
import org.xmldb.api.base.ResourceIterator;
import org.xmldb.api.base.ResourceSet;
import org.xmldb.api.base.XMLDBException;
import org.xmldb.api.modules.CollectionManagementService;
import org.xmldb.api.modules.XMLResource;

/**
 * Classe gérant l'accès aux données stockées dans la base de données XML eXist.<br/>
 * La classe est utilise le pattern Singleton.
 *
 * @author Eric BOULADIER
 * @version 1.0
 */
public class GestColl {

	public final static String driver = "org.exist.xmldb.DatabaseImpl";

	public final static String URI = "xmldb:exist://localhost:8088/xmlrpc";

	public final static String rootColl = "/db";

	public final static String collTypDoc = "typesdoc";

	public final static String pathCollTypDoc = rootColl + "/" + collTypDoc;

	public final static String collDoc = "biblio";

	public final static String pathCollDoc = rootColl + "/" + collDoc;

	public final static String dbadmin_user = "admin";

	public final static String dbadmin_pwd = "admin";

	/**
	 * L'unique instance statique
	 */
	private static GestColl gestColl;

	/**
	 * Le constructeur est privé pour interdir la création d'instances de
	 * GestColl par new. Il faut utiliser la méthode getInstance().
	 *
	 * @see #getInstance()
	 *
	 * @throws ClassNotFoundException
	 * @throws InstantiationException
	 * @throws IllegalAccessException
	 * @throws XMLDBException
	 */
	private GestColl() throws ClassNotFoundException, InstantiationException,
		IllegalAccessException, XMLDBException {
		super();
		Class cl = Class.forName(driver);
		Database database = (Database) cl.newInstance();
		DatabaseManager.registerDatabase(database);
```

```
        // initilisation de la base de données eXist (création des collections
        // de base)
        initDB();
}

/**
 * Retourne l'unique instance de GestColl
 *
 * @return gestColl, objet permetant de manipuler la base de données eXist.
 */
public static GestColl getInstance() {
        if (null == gestColl) { // Premier appel
                try {
                        gestColl = new GestColl();
                } catch (ClassNotFoundException e) {
                        // TODO Bloc catch auto-généré
                        e.printStackTrace();
                } catch (InstantiationException e) {
                        // TODO Bloc catch auto-généré
                        e.printStackTrace();
                } catch (IllegalAccessException e) {
                        // TODO Bloc catch auto-généré
                        e.printStackTrace();
                } catch (XMLDBException e) {
                        // TODO Bloc catch auto-généré
                        e.printStackTrace();
                }
        }
        return gestColl;
}

/**
 * Empeche la duplication du singleton par clonage.
 *
 * @see java.lang.Object#clone()
 */
public Object clone() throws CloneNotSupportedException {
        throw new CloneNotSupportedException();
}

/**
 * Ajoute un nouveau type de document. La procédure crée deux nouvelles
 * collections portant le nom 'nomTypDoc' dans la base de données : une sous
 * la racine biblio et l'autre sous la racine typdoc. Si le chemin
 * 'pathShemaTypDoc' est renseigné, le shema XML est enregistré dans la base
 * de données sous la racine typdoc/'nomTypDoc'.
 *
 * @param nomTypDoc :
 *            nom du type de document (exemple : these, annale)
 * @param pathShemaTypDoc :
 *            Chemin avec le nom du fichier schema XML
 */
public void addTypdoc(String nomTypDoc, String pathShemaTypDoc) {
        try {
                addCollection(nomTypDoc, pathCollDoc);
                addCollection(nomTypDoc, pathCollTypDoc);
                if (pathShemaTypDoc != null) {
                        addResource(pathCollTypDoc + "/" + nomTypDoc, "shemaXML",
                                        pathShemaTypDoc);
                }
        } catch (XMLDBException e) {
                // TODO Bloc catch auto-généré
                e.printStackTrace();
        }
}

/**
 * Ajoute le fichier de métadonnées d'un document d'un type déjà défini (par
 * addTypDoc). Si le document existe déjà, il est supprimé et remplacé.
 *
 * @param nomTypDoc :
 *            nom du type de document (exemple : these, annale)
```

```
 * @param idDoc :
 *            identifiant unique du document.
 * @param pathFile :
 *            Chemin avec le nom du fichier XML contenant les métadonnées.
 */
public void addMetaData(String nomTypDoc, String idDoc, String pathFile) {
        try {
                addResource(pathCollDoc + "/" + nomTypDoc, idDoc, pathFile);
        } catch (XMLDBException e) {
                // TODO Bloc catch auto-généré
                e.printStackTrace();
        }
}

/**
 * Supprime un fichier de métadonnées d'un document d'un type donné.
 *
 * @param nomTypDoc :
 *            nom du type de document (exemple : these, annale)
 * @param idDoc :
 *            identifiant unique du document.
 * @param pathFile
 */
public void rmvMetaData(String nomTypDoc, String idDoc) {
        try {
                rmvResource(pathCollDoc + "/" + nomTypDoc, idDoc);
        } catch (XMLDBException e) {
                // TODO Bloc catch auto-généré
                e.printStackTrace();
        }
}

/**
 * Renvoie le fichier de métadonnées sous forme d'un document DOM
 * (org.w3c.dom.Document).
 *
 * @param nomTypDoc :
 *            nom du type de document (exemple : these, annale)
 * @param idDoc :
 *            identifiant unique du document.
 * @return : fichier de métadonnées sous forme d'un document DOM.
 */
public Document rtvMetaDataAsDom(String nomTypDoc, String idDoc) {
        try {
                return rtvResourceAsDom(pathCollDoc + "/" + nomTypDoc, idDoc);
        } catch (XMLDBException e) {
                // TODO Bloc catch auto-généré
                e.printStackTrace();
                return null;
        }
}

/**
 * Renvoie le fichier de métadonnées dans un Sax content handler.
 *
 * @param nomTypDoc :
 *            nom du type de document (exemple : these, annale)
 * @param idDoc :
 *            identifiant unique du document.
 * @param handler :
 *            Sax Content handler devant recevoir le fichier de métadonnées.
 */
public void rtvMetaDataAsSax(String nomTypDoc, String idDoc,
                org.xml.sax.ContentHandler handler) {
        try {
                rtvResourceAsSax(pathCollDoc + "/" + nomTypDoc, idDoc, handler);
        } catch (XMLDBException e) {
                // TODO Bloc catch auto-généré
                e.printStackTrace();
        }
}
```

```
	/**
	 * Renvoie le nombre de documents présents dans la base de données pour un
	 * type donné.
	 *
	 * @param nomTypDoc :
	 *            nom du type de document dont on souhaite compter les documents
	 *            présents.
	 * @return nombre de documents de type nomTypDoc.
	 */
	public int nombreDocumentsParType(String nomTypDoc) {
		/* Récupération de la collection si elle existe */
		int nbDoc = 0;
		try {
			Collection col;
			col = DatabaseManager.getCollection(URI + pathCollDoc + "/"
					+ nomTypDoc, dbadmin_user, dbadmin_pwd);
			if (col != null) {
				XQueryService service = (XQueryService) col.getService(
						"XQueryService", "1.0");
				service.setProperty("indent", "yes");
				service.declareVariable("collection", pathCollDoc + "/"
						+ nomTypDoc);
				String query = "count(collection($collection))";
				CompiledExpression compiled = service.compile(query);
				ResourceSet result = service.execute(compiled);
				ResourceIterator i = result.getIterator();
				while (i.hasMoreResources()) {
					Resource r = i.nextResource();
					nbDoc += Integer.parseInt(r.getContent().toString());
				}
			}
		} catch (XMLDBException e) {
			// TODO Bloc catch auto-généré
			e.printStackTrace();
		}
		return nbDoc;
	}

	public void listDocumentsParType(String nomTypDoc) {
		/* Récupération de la collection si elle existe */
		Collection col;
		try {
			col = DatabaseManager.getCollection(URI + pathCollDoc + "/"
					+ nomTypDoc, dbadmin_user, dbadmin_pwd);
			if (col != null) {
				XQueryService service = (XQueryService) col.getService(
						"XQueryService", "1.0");
				service.setProperty("indent", "yes");
				service
						.declareVariable("collection", collDoc + "/"
								+ nomTypDoc);
				String query = "<res>{for $i in collection($collection) return
<doc>{$i/*}</doc>}</res>";
				CompiledExpression compiled = service.compile(query);
				ResourceSet result = service.execute(compiled);
				ResourceIterator i = result.getIterator();
				while (i.hasMoreResources()) {
					Resource r = i.nextResource();
					// XMLResource res = (XMLResource) i.nextResource();
					System.out.println((String) r.getContent());
				}
			}
		} catch (XMLDBException e) {
			// TODO Bloc catch auto-généré
			e.printStackTrace();
		}
	}

	private void addCollection(String nomCollection, String pathCollection)
			throws XMLDBException {
		/* Récupération de la collection si elle existe */
		String collection = pathCollection + "/" + nomCollection;
```

```
        Collection col = DatabaseManager.getCollection(URI + collection,
                        dbadmin_user, dbadmin_pwd);
        if (col == null) {
                /* si la collection n'éxiste pas, elle est crée */
                Collection root = DatabaseManager.getCollection(URI
                                + pathCollection);
                CollectionManagementService mgtService = (CollectionManagementService) root
                                .getService("CollectionManagementService", "1.0");
                col = mgtService.createCollection(collection);
        }
}

/**
 * Ajoute ou remplace une ressource dans une collection. Si la ressource
 * existe dèjà dans la collection elle est remplacée (dixit internet : à
 * vérifier)
 *
 * @param pathCollection :
 *            chemin de la collection
 * @param idResource :
 *            identifiant de la ressource
 * @param pathFile :
 *            chemin du fichier XML (avec le nom du fichier)
 * @throws XMLDBException
 */
private void addResource(String pathCollection, String idResource,
                String pathFile) throws XMLDBException {
        /* Récupération de la collection si elle existe */
        Collection col = DatabaseManager.getCollection(URI + pathCollection,
                        dbadmin_user, dbadmin_pwd);
        if (col != null) {
                // create new XMLResource
                XMLResource document = (XMLResource) col.createResource(idResource,
                                "XMLResource");
                File f = new File(pathFile);
                if (!f.canRead()) {
                        System.out.println("cannot read file " + pathFile);
                        return;
                }
                document.setContent(f);
                System.out.print("enregistrement document " + document.getId()
                                + "...");
                col.storeResource(document);
                System.out.println("ok.");
        }
}

private void rmvResource(String pathCollection, String idResource)
                throws XMLDBException {
        /* Récupération de la collection si elle existe */
        Collection col = DatabaseManager.getCollection(URI + pathCollection,
                        dbadmin_user, dbadmin_pwd);
        if (col != null) {
                // create new XMLResource
                XMLResource document = (XMLResource) col.createResource(idResource,
                                "XMLResource");
                System.out
                                .print("suppression document " + document.getId() + "...");
                col.removeResource(document);
                System.out.println("ok.");
        }
}

private Document rtvResourceAsDom(String pathCollection, String idResource)
                throws XMLDBException {
        /* Récupération de la collection si elle existe */
        Collection col = DatabaseManager.getCollection(URI + pathCollection,
                        dbadmin_user, dbadmin_pwd);
        Document doc = null;
        if (col != null) {
                col.setProperty(OutputKeys.INDENT, "no");
                XMLResource res = (XMLResource) col.getResource(idResource);
```

```
                if (res == null)
                        System.out.println("document non trouvé!");
                else
                        System.out.println(res.getContent());
                doc = (Document) res.getContentAsDOM();
        }
        return doc;
}

private void rtvResourceAsSax(String pathCollection, String idResource,
                org.xml.sax.ContentHandler handler) throws XMLDBException {
        /* Récupération de la collection si elle existe */
        Collection col = DatabaseManager.getCollection(URI + pathCollection,
                        dbadmin_user, dbadmin_pwd);
        if (col != null) {
                col.setProperty(OutputKeys.INDENT, "no");
                XMLResource res = (XMLResource) col.getResource(idResource);
                if (res == null)
                        System.out.println("document non trouvé!");
                else
                        System.out.println(res.getContent());
                res.getContentAsSAX(handler);
        }
}

private void initDB() throws XMLDBException {
        // création des 2 collections de base (biblio et typesdoc) sous /db
        addCollection(collDoc, rootColl);
        addCollection(collTypDoc, rootColl);

        // ajouts des collections correspondant aux types de document these et
        // annale.
        addCollection("these", pathCollDoc);
        addCollection("these", pathCollTypDoc);

        addCollection("annale", pathCollDoc);
        addCollection("annale", pathCollTypDoc);

}

}
```

```
package serv_eXist;

import org.w3c.dom.Document;
import org.xmldb.api.base.XMLDBException;

public class TestDB {

        /**
         * @param args
         * @throws XMLDBException
         * @throws IllegalAccessException
         * @throws InstantiationException
         * @throws ClassNotFoundException
         */
        public static void main(String[] args) throws ClassNotFoundException,
InstantiationException, IllegalAccessException, XMLDBException {
                String nomTypDoc = args[0];
                GestColl gestColl = GestColl.getInstance();
                gestColl.addTypdoc(nomTypDoc, null);

                String idDoc = args[1];
                String pathFile = args[2];
                gestColl.addMetaData(nomTypDoc, idDoc, pathFile);

                Document doc = gestColl.rtvMetaDataAsDom(nomTypDoc, idDoc);
                System.out.println(doc.getTextContent());

                gestColl.listDocumentsParType(nomTypDoc);

                System.out.println(gestColl.nombreDocumentsParType(nomTypDoc));

                gestColl.rmvMetaData(nomTypDoc, idDoc);

        }
}
```