http://webdam.inria.fr

Web Data Management

Introduction to the eXist XML native database

Serge Abiteboul
INRIA Saclay & ENS Cachan

Ioana Manolescu
INRIA Saclay & Paris-Sud University

Philippe Rigaux
CNAM Paris & INRIA Saclay

Marie-Christine Rousset
Grenoble University

Pierre Senellart
Télécom ParisTech

Contents

This chapter proposes some exercises and projects to manipulate and query XML documents in a practical context. The software used in these exercises is EXIST, an open-source native XML database which provides an easy-to-use and powerful environment for learning and applying XML languages. We begin with a brief description on how to install EXIST and execute some simple operations. EXIST provides a graphical interface which is pretty easy to use, so we limit our explanations below to the vital information which can be useful to save some time to the absolute beginner.

1 Pre-requisites

In the following, we assume that you plan to install EXIST in your Windows or Linux environment. You need a *Java Development Kit* for running the EXIST java application (version 1.5 at least). If you do not have a JDK already installed, get it from the Sun site (try searching "download JDK 1.5" with Google to obtain an appropriate URL) and follow the instructions to set up your Java environment.

Be sure that you can execute Java applications. This requires the definition of a `JAVA_HOME` environment variable, pointing to the JDK directory. The `PATH` variable must also contain an entry to the directory that contain the Java executable, `$JAVA_HOME/bin`.

1. Under Windows: load the configuration panel window; run the *System* application; choose *Advanced*, then *Environment variables*. Create a new variable `JAVA_HOME` with the appropriate location, and add the `$JAVA_HOME/bin` path to the `PATH` variable.

2. Under Linux: same as before, the exact command depending on your shell language. For instance, with *bash*, put the following in the *.bashrc* file:

```
export JAVA_HOME=your_path_to_java
export PATH=$PATH:$JAVA_HOME/bin
```

3. Under MacOS X, the Java environment should be natively configured.

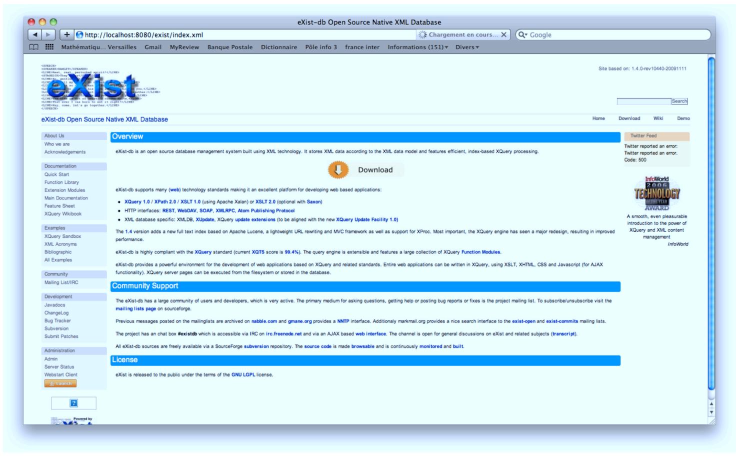
Check that you can run the *java* program. If yes you are ready to install EXIST.

2 Installing EXIST

EXIST (software, documentation, and many other things) can be found at `http://www.exist-db.org/`.

The software consists of a Java archive which can be downloaded from the home page. Assume its name is *exist.jar* (actually the archive name will contain the version as well). Run the installation package with the following command:

java -jar exist.jar



Twitter reported an error: Twitter reported an error. Code: 500

A smooth, even pleasurable introduction to the power of XQuery and XML content management
InfoWorld

eXist-db is an open source database management system built using XML technology. It stores XML data according to the XML data model and features efficient, index-based XQuery processing.

eXist-db supports many (web) technology standards making it an excellent platform for developing web based applications:

- XQuery 1.0 / XPath 2.0 / XSLT 1.0 (using Apache Xalan) or XSLT 2.0 (optional with Saxon)
- HTTP interfaces: REST, WebDAV, SOAP, XMLRPC, Atom Publishing Protocol
- XML database specific: XMLDB, XUpdate, XQuery update extensions (to be aligned with the new XQuery Update Facility 1.0)

The 1.4 version adds a new full text index based on Apache Lucene, a lightweight URL rewriting and MVC framework as well as support for XProc. Most important, the XQuery engine has seen a major redesign, resulting in improved performance.

eXist-db is highly compliant with the XQuery standard (current XQTS score is 99.4%). The query engine is extensible and features a large collection of XQuery Function Modules.

eXist-db provides a powerful environment for the development of web applications based on XQuery and related standards. Entire web applications can be written in XQuery, using XSLT, XHTML, CSS and Javascript (for AJAX functionality). XQuery server pages can be executed from the filesystem or stored in the database.

The eXist-db has a large community of users and developers, which is very active. The primary medium for asking questions, getting help or posting bug reports or fixes is the project mailing list. To subscribe/unsubscribe visit the mailing lists page on sourceforge.

Previous messages posted on the mailinglists are archived on nabble.com and gmane.org provides a NNTP interface. Additionaly markmail.org provides a nice search interface to the exist-open and exist-commits mailing lists.

The project has an chat box **#existdb** which is accessible via IRC on irc.freenode.net and via an AJAX based web interface. The channel is open for general discussions on eXist and related subjects (transcript).

All eXist-db sources are freely available via a SourceForge subversion repository. The source code is made browsable and is continuously monitored and built.

eXist is released to the public under the terms of the GNU LGPL license.

Figure 1: The home page of a local EXIST server.

Just follow the instructions, which asks in particular for the EXIST installation directory (referred to a 'EXIST home' in the following). Once the installation is completed, you can start the EXIST server as follows:

Linux, Mac OS X and other Unix systems. Move to the EXIST home directory, and type either `bin/startup.sh` or `bin/startup.bat`. If something goes wrong, look at the *README* file.

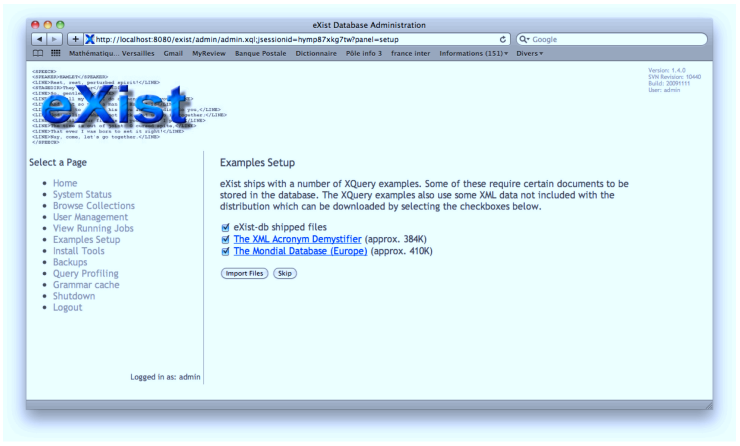
Windows. The installation procedure creates an EXIST menu and shorcuts in the desktop. Simply use them to start/stop the server.

When the server is up and running, it waits for HTTP requests on the port 8080[1]. So, using any Web browser, you can access to the eXist interface (Figure 1) at the URL `http://localhost:8080/exist/`.

From this interface, you can carry out administration tasks, get the documentation, and run a few predefined client applications. Look at the *QuickStart* part of the documentation for further information on the configuration of eXist.

3 Getting started with eXist

eXist comes with a predefined set of XML samples which can be loaded in the database. To start using these example applications, you must log in to the eXist administration page. Enter the username "admin" and leave the password field empty[2]. Next, choose *Examples Setup* from the menu on the left. Click on the "Import Data" button to start the setup and begin downloading example data from the Internet (Figure 2).



Examples Setup

eXist ships with a number of XQuery examples. Some of these require certain documents to be stored in the database. The XQuery examples also use some XML data not included with the distribution which can be downloaded by selecting the checkboxes below.

- ✓ eXist-db shipped files
- ✓ The XML Acronym Demystifier (approx. 384K)
- ✓ The Mondial Database (Europe) (approx. 410K)

Figure 2: Loading XQuery examples in eXist.

The import creates so-called "collections". A collection can be used to store a set of documents sharing the same schema, but it can be itself organized recursively in sub-collections for more flexible document sets organizations. Loading the example creates several collections, including:

1. *library*: a bibliographic RDF document, *biblio.rdf*;
2. *mondial*: administrative information on several countries;
3. *shakespeare*: a few plays from William Shakespeare.

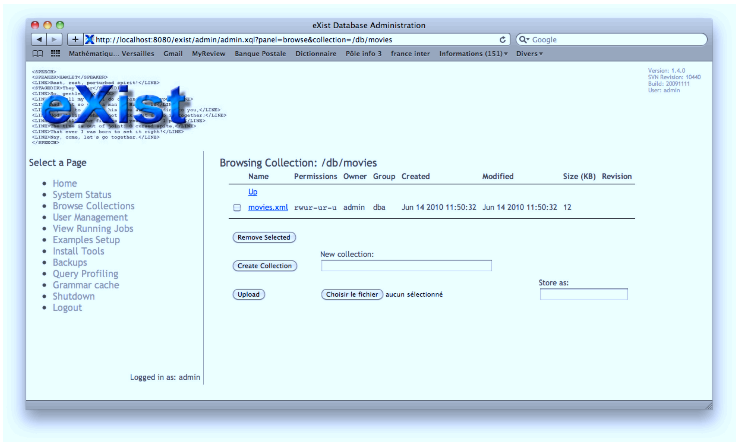
[1]This is the default port. You can change in the *jetty.xml* file in the sub-directory *tools/jetty* of eXist.

[2]The installation tool sometimes requires a password to set up eXist, in which case access to the administration page is protected by this password.

You can access these collections from the "Browse collections" option of the admin menu. You can also create a new collection with the form at the bottom of the page. Do the following:

1. create a *movies* collection;
2. add to the *movies* collection the document *movies.xml* which can be downloaded from our site.

EXIST stores now *movies.xml* in its repository, and you can search, update or transform the document. Figure 3 shows the Web interface that you should obtain.



Browsing Collection: /db/movies

Name	Permissions	Owner	Group	Created	Modified	Size (KB)	Revision
Up							
movies.xml	rwur-ur-u	admin	dba	Jun 14 2010 11:50:32	Jun 14 2010 11:50:32	12	

Figure 3: The *movies* collection, containing a *movies.xml* sample document.

Now, get back the home page, and choose the "XQuery Sandbow" option. It provides a simple interface that allows to enter XQuery (or XPath) expressions, and displays the result. Check the following XPath/XQuery query:

```
/movies
```

This shows the content of the *movies* elements found in *all* the collections stored under the `/db` root element of EXIST (which plays the role of a global root for all the documents stored in the repository). Figure 4 shows how to run XPath queries with the sanbox: the interface shows the result sequence in the bottom window.

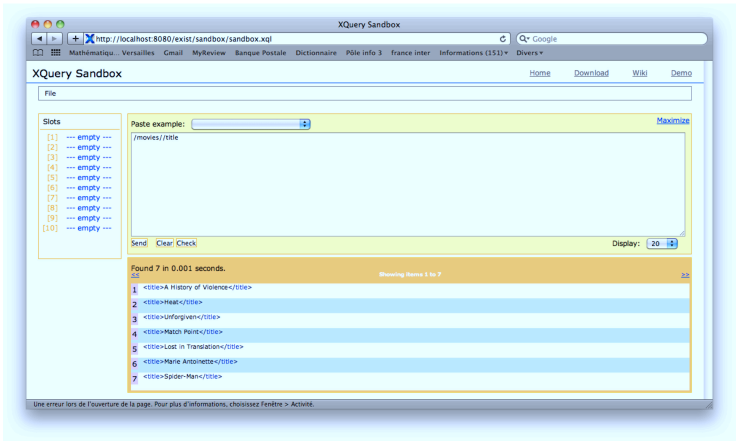
You can restrict the search to a specific documents (or set of documents) with the *document()* function. Here is an example:

```
document('/db/movies/movies.xml')/movies/movie[year=2005]
```

The *collection()* function allows to refer to a collection, as in:

```
collection('movies')/movies/movie[year=2005]
```

You are now ready to play with XPath and XQuery, using the sandbox. The next section proposes exercises.


XQuery Sandbox
http://localhost:8080/exist/sandbox/sandbox.xql
Google
Mathématiqu... Versailles Gmail MyReview Banque Postale Dictionnaire Pôle info 3 france inter Informations (151) Divers
XQuery Sandbox
Home Download Wiki Demo
File
Slots
Paste example:
Maximize
Send Clear Check
Display: 20
<< Showing items 1 to 7 >>

```
[1] --- empty ---
[2] --- empty ---
[3] --- empty ---
[4] --- empty ---
[5] --- empty ---
[6] --- empty ---
[7] --- empty ---
[8] --- empty ---
[9] --- empty ---
[10] --- empty ---
```

```
/movies//title
```

```
Found 7 in 0.001 seconds.
1 <title>A History of Violence</title>
2 <title>Heat</title>
3 <title>Unforgiven</title>
4 <title>Match Point</title>
5 <title>Lost in Translation</title>
6 <title>Marie Antoinette</title>
7 <title>Spider-Man</title>
```

Une erreur lors de l'ouverture de la page. Pour plus d'informations, choisissez Fenêtre > Activité.

Figure 4: Running the `/movies//title` XPath query in the sandbox.

4 Running XPath and XQuery queries with the sandbox

4.1 XPath

Get the *movies.xml* and *movies_refs.xml* documents from the book's Web site, and insert them into EXIST. Look at the document structure: in the first one, each movie is representated as one element, including the director's and actors' names. In the second one the document consists of two lists, one for movies, one for actors, the former referencing the latter.

Express the following queries in XPath 1.0, on both documents (note: *movies.xml* does not require joining the two lists, which makes expressions easier).

1. All `title` elements.
2. All movie titles (i.e., the textual value of `title` elements).
3. Titles of the movies published after 2000.
4. Summary of "Spider-Man".
5. Who is the director of *Heat*?
6. Title of the movies featuring Kirsten Dunst.
7. Which movies have a summary?
8. Which movies do *not* have a summary?
9. Titles of the movies published more than 5 years ago.
10. What was the role of Clint Eastwood in *Unforgiven*?
11. What is the *last* movie of the document?

12. Title of the film that immediatly precedes *Marie Antoinette* in the document?
13. Get the movies whose title contains a "V" (hint: use the function *contains*()).
14. Get the movies whose cast consists of exactly three actors (hint: use the function *count*()).

4.2 XQuery

Exercise 4.1 *Get the movies_alone.xml and artists_alone.xml documents from the book's Web site. They contain respectively the list of movies with references to artists, and the list of artists. It might be simpler to first express complex queries over the movies.xml document which contains all the data in one file, before reformulating them over these two documents.*

For convenience, you may define a variable for each of these document with a ***let*** *at the beginning of each query:*

```
let $ms:=doc("movies/movies_alone.xml"),
    $as:=doc("movies/artists_alone.xml")
```

Express the following XQuery queries:

1. *List the movies published after 2002, including their title and year.*
2. *Create a flat list of all the title-role pairs, with each pair enclosed in a "result" element. Here is an example of the expected result structure:*

```
<results>
  <result>
    <title>Heat</title>
    <role>Lt. Vincent Hanna</role>
  </result>
  <result>
    <title>Heat</title>
    <role>Neil McCauley</role>
  </result>
</results>
```

3. *Give the title of movies where the director is also one of the actors.*
4. *Show the movies, grouped by genre. Hint: function distinct-values() removes the duplicates from a sequence. It returns* atomic values.
5. *For each distinct actor's id in movies_alone.xml, show the titles of the movies where this actor plays a role. The format of the result should be:*

```
<actor>16,
<title>Match Point</title>
<title>Lost in Translation</title>
</actor>
```

Variant: show only the actors which play a role in at least two movies (hint: function count() returns the number of nodes in a sequence).

6. *Give the title of each movie, along with the name of its director. Note: this is a join!*
7. *Give the title of each movie, and a nested element* `<actors>` *giving the list of actors with their role.*
8. *For each movie that has at least two actors, list the title and first two actors, and an empty "et-al" element if the movie has additional actors. For instance:*

```
<result>
<title>Unforgiven</title>
<actor>Clint Eastwood as William 'Bill' Munny</actor>
<actor>Gene Hackman as Little Bill Daggett</actor>
<et-al/>
</result>
```

9. *List the titles and years of all movies directed by Clint Eastwood after 1990, in alphabetic order.*

4.3 Complement: XPath and XQuery operators and functions

XPath proposes many operators and functions to manipulate data values. These operators are mostly used in predicates, but they are also important as part of the XQuery language which inherits all of the XPath language. We briefly describe the most important ones here, all from XPath 1.0. XPath 2.0 widely extends the library of available functions.

Operators.

- Standard arithmetic operators: `+`, `-`, `*`, `div`, `mod`.

  Warning! `div` is used instead of the usual `/`.

  Warning! "−" is also a valid character inside XML names; this means an expression `a-b` tests for an element with name `a-b` and does not compute the substraction between the values contained in elements `a` and `b`. This substration can be expressed as `a - b`.

- Boolean operators: `or`, `and`, as in `@a and c=3`.

- Equality operators: `=`, `!=` that can be used for strings, Booleans or numbers.

  Warning! `//a!=3` means: there is an `a` element in the document whose string value is different from 3.

- Comparison operators: <, <=, >=, > as in `$a<2 and $a>0`.

  Warning! They can only be used to compare numbers, not strings.

  Warning! If an XPath expression is embedded in an XML document (this is the case in XSLT), `&lt;` must be used in place of <.

- Set-theoretic union of node sets: `|` as in `node()|@*`.

Note that `$a` is a *reference* to the variable `a`. Variables cannot be defined in XPath 1.0, they can only be referred to.

Node functions.

The following functions apply to node sets.

- `count($s)` and `sum($s)` return, respectively, the *number of items* and the *sum of values* in the node set `$s`.
- `local-name($s)`, `namespace-uri($s)`, and `name($s)` respectively return the *name without namespace prefix*, *namespace URI*, and *name with namespace prefix*, of the node argument; if `$s` is omitted, it is taken to be the context item.

String functions.

The following functions apply to character strings.

- `concat(`$\$s_1, \ldots, \$s_n$`)` *concatenates* the strings $\$s_1, \ldots, \$s_n$.
- `starts-with($a,$b)` returns true if the string `$a` *starts with* the string `$b`.
- `contains($a,$b)` returns true if the string `$a` *contains* the string `$b`.
- `substring-before($a,$b)` returns the *substring* of `$a` that precedes the first occurrence of `$b` in `$a`.
- `substring-after($a,$b)` returns the *substring* of `$a` that follows the first occurrence of `$b` in `$a`.
- `substring($a,$n,$l)` returns the *substring* of `$a` of length `$l` starting at the `$n`-th position of `$a`. (One starts counting from 1). If `$l` is omitted, the *substring* of `$a` of length `$l` starting at the `$n`-th position of `$a` is returned.
- `string-length($a)` returns the *length* of the string `$a`.
- `normalize-space($a)` *removes* all leading and trailing *white-space* from `$a`, and *collapses* each white-space sequence to a single white space.
- `translate($a,$b,$c)` returns the string `$a`, where all occurrences of a character from `$b` has been *replaced* by the character at the same place in $c.

Boolean and number functions.

- `not($b)` returns the *logical negation* of the Boolean `$b`.
- `floor($n)`, `ceiling($n)`, and `round($n)` round the number `$n` to, respectively, the *next lowest*, *next greatest*, and *closest* integer.

5 Programming with eXist

We give below some examples of the programming interfaces (API) provided by eXist. They can be used to access an XML repository from an application program. Two interfaces are shown: the first one, `XML:DB`, is a JDBC-like component that can be introduced in a Java application as a communication layer with eXist; the second one are the web services of eXist.

Along with XPath and XQuery, these APIs constitute a complete programming environment which is the support of projects proposed in the final section of the Chapter.

5.1 Using the `XML:DB` API with eXist

Documents stored in eXist can be accessed from a Java application. eXist provides an implementation of the `XML:DB` API specification, which is to XML databases what JDBC is to relational databases.

We give below a few examples of Java programs that connect to eXist, access documents and perform XQuery queries. You can get these files from our site. You need a Java programming environment (*Java Development Kit*) with the `JAVA_HOME` environment variable properly set, and `JAVA_HOME/bin` directory added to the `PATH` environment variable.

In order to use the eXist API, the following archives must be put in your `CLASSPATH`.

1. *exist.jar*, found in `EXIST_HOME` or `EXIST_HOME/lib`;
2. *xmldb.jar*, *xmlrpc-1.2-patched.jar*, and *log4j-1.2.14.jar*, found in `EXIST_HOME/lib/core`.

These archives are sufficient to compile and run our examples. For more complex Java programs, some other archives might be necessary (for instance the Xerces parser or the Xalan XSLT processor). They can usually be found in `EXIST_HOME/lib`

You can find many explanations and examples on Java programming with eXist at *http://exist.sourceforge.net/devguide.html*.

5.2 Accessing eXist with Web Services

If you do not want to use Java, or if the architecture of your application makes the embedding of XQuery calls in your code unsuitable, you can use the Web Service layer of eXist to send queries and get result. This is actually an excellent example of the advantage of a well-defined query language as an interface to a remote data source, and serves as a (small) introduction to the world of distributed computing with web services.

eXist runs a server on a machine, and the server opens several communication port to serve requests that might come from distant locations. Several protocols are available, but the simple one is based on REST (Representational state transfer), a service layer implementation that completely relies on HTTP.

Recall that a web service allows a Client to send a function call to a server without having to deal with data conversion and complicated network communication issues. In the case of REST-style services, the function call is encoded as a URL, including the function parameters, and transmitted to the server with an HTTP GET or POST request. The REST servers sends back the result in HTTP. REST services are quite easy to deal with: the client application just

```java
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.modules.XMLResource;

public class ExistAccess {
        protected static String DRIVER = "org.exist.xmldb.DatabaseImpl";
        protected static String URI = "xmldb:exist://localhost:8080/exist/xmlrpc";
        protected static String collectionPath = "/db/movies/";
        protected static String resourceName = "Heat.xml";

        public static void main(String[] args) throws Exception {

                // initialize database driver
                Class cl = Class.forName(DRIVER);
                Database database = (Database) cl.newInstance();
                DatabaseManager.registerDatabase(database);

                // get the collection
                Collection col = DatabaseManager.getCollection(URI + collectionPath);

                // get the content of a document
                System.out.println("Get the content of " + resourceName);
                XMLResource res = (XMLResource) col.getResource(resourceName);
                if (res == null) {
                        System.out.println("document not found!");
                } else {
                        System.out.println(res.getContent());
                }
        }
}
```

Figure 5: First example: retrieving a document

```java
import org.exist.xmldb.XQueryService;
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.modules.XMLResource;

public class ExistQuery {
        protected static String DRIVER = "org.exist.xmldb.DatabaseImpl";
        protected static String URI = "xmldb:exist://localhost:8080/exist/xmlrpc";
        protected static String collectionPath = "/db/movies";
        protected static String resourceName = "movies.xml";

        public static void main(String[] args) throws Exception {

                // initialize database driver
                Class cl = Class.forName(DRIVER);
                Database database = (Database) cl.newInstance();
                DatabaseManager.registerDatabase(database);

                // get the collection
                Collection col = DatabaseManager.getCollection(URI + collectionPath);

                // query a document

                String xQuery = "for $x in doc('" + resourceName + "')//title "
                                + "return data($x)";
                System.out.println("Execute xQuery = " + xQuery);

                // Instantiate a XQuery service
                XQueryService service = (XQueryService) col.getService("XQueryService",
                                "1.0");
                service.setProperty("indent", "yes");

                // Execute the query, print the result
                ResourceSet result = service.query(xQuery);
                ResourceIterator i = result.getIterator();
                while (i.hasMoreResources()) {
                        Resource r = i.nextResource();
                        System.out.println((String) r.getContent());
                }
        }
}
```

Figure 6: Second example: query a collection

needs an HTTP client library, available in all programming languages. Moreover, the service can easily be tested with a browser by sending "manual" URL.

Here is a first example: the following GET query retrieves the *movies.xml* document from your local EXIST repository:

```
http://localhost:8080/exist/rest/db/movies/movies.xml
```

Note that this is a "pseudo-url": there is nothing on your local disk that resembles directory path */exist/rest/db/movies/movies.xml*. Actually,

1. the REST service is located at */exist/rest/*; thanks to a URL rewriting mechanism, any GET request that begins with this address is redirected to the service;
2. the remaining part of the URL, *db/movies/movies.xml*, is a parameter sent to the service; it must denote either a collection or a document.

In summary, a REST query such as the above one is tantamount to a *document()* or *collection()* call sent to EXIST. The service accepts other parameters, and in particular the `_query` parameter whose value may be an XPath or XQuery expression. Try the following URL:

```
http://localhost:8080/exist/rest/db/movies?_query=//title
```

You should get the result of the XPath query `collection('movies')//title`.

Remark 5.1 *REST services are called via GET or POST requests. In the case of GET, all the values must be URL-encoded. This is automatically done by a browser (or any HTTP client) when the URL is a link in an HTML page, but be careful when you directly enter the URL.*

Here is a selected list of parameters accepted by the EXIST REST service (for details, please refer to the *Developper's guide* on the EXIST web site).

- `_xsl=XSL Stylesheet`.
  Applies an XSLT stylesheet to the requested resource. If the `_xsl` parameter contains an external URI, the corresponding external resource is retrieved. Otherwise, the path is treated as relative to the database root collection and the stylesheet is loaded from the database.
- `_query=XPath/XQuery Expression`.
  Executes a query specified by the request.
- `_encoding=Character Encoding Type.`
  Sets the character encoding for the resultant XML. The default value is UTF-8.
- `_howmany=Number of Items.`
  Specifies the number of items to return from the resultant sequence. The default value is 10.
- `_start=Starting Position in Sequence.`
  Specifies the index position of the first item in the result sequence to be returned. The default value is 1.

- `_wrap=yes | no.`
  Specifies whether the returned query results are to be wrapped into a surrounding <exist:result> element. The default value is yes.
- `_source=yes | no` Specifies whether the query should display its source code instead of being executed. The default value is no.

6 Projects

The following projects are intended to let you experiment XML data access in the context of an application that needs to manipulate semi-structured data. In all cases, you must devise an architecture that fulfills the proposed requirements, the only constraint being that the data source model *must* be XML. Of course you are invited to base your architecture on an appropriate combination of the XML tools and languages presented in this book and during the classes: XPath, XSLT (see the resources available on the companion Web site of this book), XQuery, and Java APIs or REST services.

6.1 Getting started

The minimal project is a very simple Web application that allows to *search* some information in an XML document, and displays this information in a user-friendly way. You can take the *movies.xml* document, or any other XML resource that can be used for the same purpose.

The application must be accessible from a Web Browser (e.g., Firefox) or from a smartphone browser (e.g., a mobile phone: take one of the many mobile phone simulators on the Web). Here are the requirements:

1. there must be a form that proposes to the user a list of search criteria: (fragment of) the title, list of genres, director and actors names, years, and key-words that can be matched against the summary of the movie;
2. when the user submits the form, the application retrieves the relevant movies from the XML repository, and displays the list of these movies in XHTML;
3. in the previous list, each movie title should be a link that allows to display the full description of the movie.

This is a simple project. It can be achieved by a single person in limited time. In that case you are allowed to omit other markup languages (but doing it will result in a better appreciation!).

6.2 Shakespeare Opera Omnia

The project is based on the Shakespeare's collection of plays shipped with the eXist software. The application's purposes can be summarized as follows: browsing through a play in order to analyze its content, read some specific parts and maybe find related information.

Basically, it consists in writing a small Web application devoted to navigating in a play and extracting some useful information. Here are a few precise requirements:

1. show the part of a given character, for a given act and/or a given scene;
2. show the table of contents of the play, along with the organization in acts and scenes, and the characters present in each scene;
3. show a full summary of the play, including the author, list of characters, stages requirements, etc.

The web application should be presented in a consistent way, allowing users to switch from one summary to another. It should be possible to navigate from the table of contents to the full description of a scene, from the description of the scene to a character, and from anywhere to the full summary. This list is by no way restrictive. Any other useful extract you can think of will raise the final appreciation!

Finally, the web application should be available on a traditional browser, as well as on a smartphone.

The project must be conducted in two steps:

1. write a short description of the architecture and design of your XML application; check with some limited tests than you know how to put each tool at the right place where it communicates correctly with the rest of the application;
2. once the design has been validated, you can embark in the development.

6.3 MusicXML on line

This is an exploratory project, since there are no guarantees that all the ideas presented below can be implemented in a reasonable amount of time. The project is also focused on a specific area: music representation and manipulation. So, it should be chosen only by people with both musical inclination and appetite for not yet explored domains.

Music is traditionally distributed on the Web via audio files, described by a few meta-data (author, style, etc.). A quite different approach consists in distributing an accurate content description based on a digital score. The MusicXML DTD serves this purpose. It allows to represent the music notation of any piece of music (voice, instruments, orchestra, etc.). The goal of the projet is to specify a Web portal distributing music scores, and to investigate the functionalities of such a portal.

The following is a list of suggestions, but the project is really open to new directions. Other ideas are welcome, but please talk first to your advisor before embarking in overcomplicated tasks.

Data

It is not that difficult to find data (e.g., digital music scores). Look for instance at *http://icking-music-archive.org/*. A collection is also available from your advisor, but you are encouraged to search resources on the Web. Any kind of music could do, from full symphony to voice-piano/guitar reduction of folk songs. You can look for digital score produced by tools such as *Finale* or *Sibelius*. From these digital scores it is possible to export MusicXML files.

Core functions

The basic requirement is to be able to store XML music sheets in EXIST, and to display the music on demand. Displaying score can be achieved with the Lilypond software (*http://lilypond.org/*), along with a convertor (`musicxml2ly`) from MusicXML to Lilypond format. Putting all these tools together is probably the first thing you should do.

It would be useful to extract some important parts from the MusicXML document. For instance, you can extract the lyrics from a song, as well as the melody. A basic search form to extract lyrics and/or melody based on a simple pattern (i.e., e keyword or a musical fragment) would be welcome.

Advanced options

Here is now a list of the possible additional functionalities that could be envisaged. You are free to limit yourself to a state-of-the-art of the possible solutions, to implement (in a simple way) some of them, or both.

1. **Input**: how can we enter music (e.g., a song) in a database, and how can we query music (e.g., with a keybard simulator, by whistling, by putting your iPod in front of a microphone, etc.);
2. **Ouput**: OK, we can print a score; but what if we we want to *listen* music? Can we transform an XML document to a MIDI document? Yes, this is possible with Lilypond: you are encouraged to investigate further.

Finally, the web application should be available on a traditional browser, as well as on a smartphone.

This remains, of course, about XML and its multiple usages. You must devise a convenient architecture, using appropriately all the tools that, together, will enable the functionalities of your application.