Frequently Asked Questions

Table of contents

Questions

1. What is Xindice?

Xindice is an open source Native XML Database. It stores and indexes compressed XML documents in order to provide that data to a client application with very little server-side processing overhead. It also provides functionality that is unique to XML data, which can't easily be reproduced by relational databases.

2. What is Xindice not?

Xindice is not a persistent DOM implementation. It was not designed to store and manage single monster sized documents, where one document is treated as a set of mini documents. It was specifically designed for managing many small to medium sized documents.

3. Why not use a relational database?

XML documents are organized as tree structures. Relational databases organize data in a tabular, or grid-like fashion, and use relational linking in order to expose hierarchical constraints on the data. Unfortunately, while it's generally very easy to map relational data into XML, trying to map XML, which can be very complex and freeform, into relational data can be incredibly difficult and lossy.

4. Why not use an object database?

Object database are tightly coupled with the applications that they serve, and are not generally considered to be client/server databases. Typically, the objects that they manage can only be deserialized and used by the programming language that created them. While object graphs like DOM trees can be persisted in object databases, the number of objects that are produced for a a large collection of XML documents can become a serious performance and resource bottleneck.

5. Why not use a full-text indexing engine?

Full text indexing engines are very fast and appropriate for performing broad queries across a set of documents. The drawbacks to using them for XML applications is that creating an inverted index causes you to lose document context, requiring very intensive post-processing to isolate the document nodes that are being managed. Xindice provides efficient indexing of element and attribute values in addition to the ability to return and manage aggregated document fragments.

6. What query languages do you support?

Xindice supports XPath and XUpdate. XPath is the W3C's XML Pathing language, and is a very powerful way of selecting sets of nodes from documents in a collection. XUpdate is the XML:DB Initiative's updating language, and is very flexible in its ability to update sets of documents that meet specific XPath criteria.

7. Is your XPath implementation fully compliant?

Our XPath implementation is the Apache Software Foundation's Xalan library. Xindice provides indexing mechanisms and query optimization facilities to augment the performance of collection level XPath queries, but Xalan ultimately has the final say in any query.

8. Which JDK should I use?

We recommend Sun's SDK for Java 1.4 or 1.3. Xindice will not work with the SDK 1.2 or earlier. When using 1.3, be aware that there are known issues with Sun's SDK for Java 1.3.1 on several Linux distributions, but 1.3.0 or 1.4.2 should work without any problems.

9. The XML:DB API is missing something, what should I do?

The XML:DB API is being designed be the XML:DB Initiative, and so its interfaces and classes are predetermined by that group. If you have any questions or would like to help further define the standard API for XML databases, please visit http://xmldb-org.sourceforge.net.

10. My 5 megabyte file is crashing the command line, help?

See FAQ #2. Xindice wasn't designed for monster documents, rather, it was designed for collections of small to medium sized documents. The best thing to do in this case would be to look at your 5 megabyte file, and determine whether or not it's a good candidate for being sliced into a set of small documents. If so, you'll want to extract the separate documents and add them to a Xindice collection individually. A good example of this, would be a massive document of this form:

```
<cars>
   <car>
      <make>Boyoda</make>
      <model>Cordova</model>
      <year>1989</year>
   </car>
   <car>
```

```
        <make>Frod</make>
        <model>Tortorus</model>
        <year>1990</year>
     </car>
     ...
  </cars>
```

In this case, it makes more sense to split the file into several `<car/>` documents instead of a single `<cars/>` document.

11. XML parser conflict

```
org/w3c/dom/DOMException
  at org.Xindice.xml.dom.NodeImpl.<clinit>(NodeImpl.java:86)
  at java.lang.Class.forName0(Native Method)
  at java.lang.Class.forName(Class.java:120)
  at org.apache.xerces.parsers.DOMParser.setDocumentClassNa
  ...
```

This is also a common error one will encounter, and usually is related to having more than one version of the Xerces XML Parser on your CLASSPATH. Before running Xindice or any client programs, be sure that the Xerces Jar file that is included in the Xindice distribution is the first Xerces Jar file on your CLASSPATH.

12. Where is my database?

Since Xindice 1.1 the database is running under an application server (e.g. Tomcat). The database configuration is located at [your webapps directory]/xindice/WEB-INF/system.xml (assuming of course that your war was unpacked by the container).

Xindice configuration, by default, uses "./db" as the database location, which is assumed as a path relative to the webapp's /WEB-INF location. This means that usually the database will be located at [your webapps directory]/xindice/WEB-INF/db.

This behaviour can be overridden in two ways: either by editing /WEB-INF/system.xml and setting an absolute path as the "dbroot" attribute of the "root-collection" tag, or by setting the Java property "xindice.db.home" to a path that will be the parent for the "./db" directory. As an example, setting the property on the command line with a value of "/var/xindice" (-Dxindice.db.home=/var/xindice) will mean that the database will be found at /var/xindice/db.