BaseX - Full Documentation

BaseX Team

Table of Contents

1. Introduction

BaseX is a native, open-source XML database and efficient XPath/XQuery Full Text processor. It supports very large XML instances and offers a visual, interactive frontend. BaseX is written in Java and freely available for download. It is developed by the Database and Information Systems Group at the University of Konstanz.

1.1. Prerequisites

Operating System

BaseX was tested on the following platforms:

- Windows 2000, XP, Vista, 7
- Linux: Debian/Ubuntu, SuSE, Redhat
- Max OS X (10.4+)
- OpenBSD (4.3)

Every other platform with a Java VM should be able to run BaseX.

Software

To run any version of BaseX, you need version 1.5 or later of the Java runtime environment (JRE) or development kit (JDK).

The latest version of Java can be downloaded at www.java.com or java.sun.com.

Hardware

BaseX works with very limited resources (400 MHz, 64 MB RAM). To work with larger databases, however, you might benefit from more RAM. If you encounter main memory limits, there are several things you can try:

- Increase Java's virtual memory
- Switch off database indexes
- Use BaseX's internal XML parser

1.2. Installation

BaseX works without any installation. The following files can be downloaded from the homepage:

- *BaseX.jar*: simple JAR file
- *BaseX.exe*: a JAR file, which is wrapped into a Windows executable
- *BaseX-XQJ.jar*: XQuery for Java Database API
- *BaseX-XMLDB.jar*: XML:DB Database API
- *BaseX-Complete.zip*: JAR files, sources, documentation and files to build the project

The JAR file and the Windows executable can be launched with a simple double-click. The "Complete" package contains the following files and directories (selection):

Project Directory

```
doc                 Project documentation
```

```
etc             Various files for project handling
src             Project sources
.project        Eclipse project file
makefile        Linux Makefile
BaseX.jar       Public JAR file
license.txt     Licensing Information
readme.txt      Project Information
build.xml       Ant Build file
input.xml       XML sample document
```

In the `etc` folder, you find Linux scripts and Windows batch files which can moved into your PATH environment to speedup execution.

1.3. Quick Start

By double-clicking the JAR file, the graphical interface is launched. If you start BaseX on the command line, or use the scripts mentioned above, you have some more options:

Console version

```
java -cp BaseX.jar org.basex.BaseX
```

Running the server and a client instance

```
java -cp BaseX.jar org.basex.BaseXServer
java -cp BaseX.jar org.basex.BaseXClient
```

GUI version (granting more memory)

```
java -Xmx512m -jar BaseX.jar
```

1.4. Starting with BaseX GUI

Just download the BaseX.jar or BaseX.exe to start the GUI Version of BaseX. If you download the source, you have to start the BaseXWin.java class.

1.5. Starting with BaseX Console

Just enter java -cp BaseX.jar org.base.BaseX on the command line. The Java option -Xmx... reserve more memory, and the -h flag of BaseX lists all available flags.

1.6. Using BaseX as Server-Client

Start the BaseXServer.java class and the Server is running. Now you can connect with BaseXClient.java.

- To start BaseX in server mode, type in: java -cp BaseX.jar org.basex.BaseXServer
- To run a BaseX client, please type in: java -cp BaseX.jar org.basex.BaseXClient -s server.org

1.7. Starting BaseX as a service

2. Working with the BaseX Console

2.1. Launching BaseX in console mode

Enter java -cp BaseX.jar org.basex.BaseX -h to get all available flags.

```
BaseX 5.6; DBIS, University of Konstanz
Usage: BaseX [options] [query]
  [query]    specify query file
  -c         chop whitespaces
  -d         debug mode
  -e         skip entity parsing
  -o [file]  specify output file
  -q <cmd>   send BaseX commands
  -v/V       show (all) process info
  -x         print result as xml
  -z         skip query output
```

2.2. What can I do in the console mode?

Type in "help" to get a list of all BaseX commands. Several commands can be separated by semicolons. To evaluate commands without entering the console mode, you can use the -q option on the command line. See the following example.

```
java -Xmx512m -cp BaseX.jar org.basex.BaseX
  -vq "create xml input.xml; xquery /"

Database 'input' created in 155.17 ms.

Parsing   : 110.35 ms
Compiling : 0.81 ms
Evaluating: 0.05 ms
Printing  : 2.06 ms
Total Time: 113.35 ms
Results   : 1 Item
Printed   : 360 Bytes
```

2.3. Description of the Console Features

You can type in help in the console to get a list of all BaseX commands. Several commands can be separated by semicolons. To evaluate commands without entering the console mode, you can use the -q option on the command line

Database Commands

- Create (create [DB|FS|INDEX] [...]): Creates database from XML or filesystem, or creates index

Example: create DB factbook.xml factbook

- Open (open [database]): Opens the specified [database].
- Info (info [DB|INDEX|TABLE]?): Shows information on the currently opened database.
- Close (close): Closes the current database.
- List (list): Lists all available databases.
- Drop (drop [DB|INDEX] [...]): Drops a database or an index.
- Export (export [file]): Exports the current context set to an XML [file].
- Optimize (optimize): Optimizes the current database structures.

Query Commands

- XQuery (xquery [query]): Evaluates an XQuery and prints the result.

  Example: xquery for $i in (1,2,3) return $i
- Find (find [query]): Evaluate a simple keyword [query] and print its results. This command is used in the simple search mode in the GUI.
- CS (cs [query]): Evaluates the specified XPath [query] and set the result as new context set.

Update Commands

- Copy (copy [pos] [source] [target]): Copy database nodes. Evaluates the [source] query and copies the resulting nodes as child nodes into the [target] query. [pos] specifies the child position; if 0 is specified, the nodes are inserted as last child. The queries should be enclosed by brackets.
- Delete (delete ["target"]): Delete database nodes resulting from the specified [target] query. The query should be enclosed by brackets.

  Example: delete //*
- Insert (insert [fragment|element|attribute|text|comment|pi] [...]): Insert database nodes. Insert a fragment or a specific node at the specified child [pos] of the specified [target] query.

  Example: insert element test 0 //*
- Update (update [element|attribute|text|comment|pi] [...]): Update database nodes satisfying the specified [target] query.

General Commands

- Help (help [command]): Get help on BaseX commands. If [command] is specified, information on the specific command is printed; otherwise, all commands are listed. If 'all' is specified, hidden commands are included.
- Set (set [option] [value?]): Sets global options. The currently set values can be shown with the info command.
- Exit (exit/quit): Leave the console mode of BaseX.

3. Working with the BaseX GUI

3.1. Description of all the Views in the GUI

- Text View: Displays query results and other textual output.
- Map View: This visualization represents all data in a TreeMap. All nodes of the XML document are represented as rectangles, filling the complete area. You can choose in the Menu Options/Map Layout the different Algorithms: Split, Strip, Squarified and Slice and Dice Layout.
- Folder View: This visualization displays all XML nodes in a usual tree view.
- Table View: This visualization displays all XML nodes in a table with rows and columns.
- Scatterplot View: This visualizations displays all XML nodes in a scatterplot.

3.2. Description of the GUI Features

What does the Filter button exactly do?

After pressing this button, the visualizations display the previously highlighted XML nodes, omitting all the other nodes of the document. If realtime filtering is enabled, the Filter button will be disabled, and all results are automatically filtered.

What is Realtime Filtering?

If realtime filtering is enabled, all visualizations directly show the query results while entering the query. If this feature is disabled, the query results are highlighted in the visualizations and can be ecplicitly filtered using the 'Filter' button.

How can I search without XQuery?

The Search field triggers a simple search query in the XML document. The following syntax is supported:

```
Query   Description
foo  Find tags and texts containing foo
=foo  Find exact matching text nodes
@foo  Find attributes and attribute values
@=foo  Find exact attribute values
```

The search field in the query panel and the search field right beyond the buttons, do they differ?

Not really, both offer the same functionality, but in the query panel there is a dropdown menu where it's possible to choose tags or attributes of the XML document.

How can I configure the look and feel of BaseX?

All panels are freely adjust- and dragable. Changing of colors, fonts and the layout is possible via the 'Layout' entry in the 'Options' menu.

What is the Main-Memory Mode?

The Main-Memory Mode speeds up querying but disables updates. The table data is kept in memory and the text of a document is still accesseed from disk.

How can I import my file system? What does that mean?

To import a file system, go to 'File/Import Filesystem...'. You will be prompted for the next steps. A file hierarchy traversal is performed and the directory structure of the filesystem is mapped into an XML representation. Additionally some metadata is extracted from some known file types. What you get is an 'XML view' of your current file system. You can query the filesystem just as any other database instance. Manipulation (update, deletion) of the XML database instance representing a file hierarchy has no effect on the real filesystem.

How can I import my data aka how do I create a new database instance aka how to shred an XML file?

There are 2 ways:

- If your input data is an XML file, you can use the 'New' command. BaseX will create a new database with the name of your XML file. (GUI only)
- You can use the create command. (GUI and console)

4. General Information

4.1. Location of BaseX Files

The databases are stored in a BaseXData directory in your home directory. The path can be changed in the GUI via the Options/Preferences menu. The two configuration files .basex/.basewin are stored in the same directory. This path can't be changed by the user.

4.2. How do you measure the performance of the queries?

The measurements include parsing, compilation, evaluation and printing time of a query. There are different ways to retrieve the performance info:

- Console: use "-v" flag as command line argument
- Console Mode: enter set info or set info all in the console mode
- GUI Mode: display performance results in the QueryInfo view

4.3. What indexing techniques are available and what do they do?

Indexes can speedup queries by magnitudes. Currently, three indexes exist:

- Text Index: All text nodes are indexed to speedup XPath predicates.
- Attribute Value Index: All attributes are indexed to speedup XPath predicates.

- Full-Text Index: A full-text index is created to speedup content based queries in XPath.

Note that the indexes only speedup XPath and simple user queries. The query processor will optimize queries completely automatic whenever possible. Expect XQuery to make use of the indexes in the next official release.

4.4. How can I create a Database for a couple of XML files?

There are 2 ways to do this:

- GUI Mode: Open the GUI, choose File -> New -> Browse -> (choose directory)
- Console Mode: java -cp BaseX.jar org.basex.BaseX -q "set chop; create db /path/to/collection dbname"

Next, you can open a collection and run an XQuery as follows: for $doc in collection("dbname") return ....

4.5. Is there a way to save large results of a query in a file?

The best way to save large results is to work on the command line and use the "-o" flag, as shown below:

```
sample.xq: for $node in doc('dbname')//etc return $node/something
BaseX-Command:
java -cp BaseX.jar org.basex.BaseX -o result.xml sample.xq
```

5. Description of BaseX-Features

5.1. XQuery Implementation

All XPath optimizations are integrated in XQuery and the XPath implementation was obsolete and so it was replaced by XQuery. As XPath is a subset of XQuery, you won't actually notice many differences, but XQuery directly allows to work with several databases/documents, so you won't have to manually change the Context references. A query like...

```
for $n in 1 to 10
for $doc in doc( concat('document', $n) )
return $doc//any/path/etc
```

could then be used to access several databases in one query. You can just type all your XPath queries in the XQuery textfield.

5.2. XQuery Full-Text Implementation

The full-text features can be used in XPath as well as in XQuery. Again, queries will be evaluated much faster in XPath, and XQuery covers more features of the language specification. The following example queries are based on the W3C Use Cases:

```
Return all title tags which contain the word 'Usability'
```

```
//title[text() ftcontains 'Usability']

Return all authors, containing 'Marigold'
for    $i in //author score $s
where  $i/text() ftcontains 'Marigold'
return <hit score="{ $s }">{ $i/text() }<
```

5.3. Description of the Indexes

Text indexes allow a speedup of order of magnitudes for text-based queries. Here are some examples for queries which are rewritten for index access:

`Text-Based Queries:`

- //node()[text() = 'Usability']
- //div[p = 'Usability' or p = 'Testing']
- path/to/relevant[text() = 'Usability Testing']/and/so/on

`Attribute Index:`

- //node()[@align = 'right']
- descendant::elem[@id = '1']
- range/query[@id >= 5]

`Full-Text Index:`

- //node[text() ftcontains 'Usability']
- //node[text() ftcontains 'Usebiliti' with fuzzy]
- //book[chapter ftcontains ('web' ftor 'WWW' without stemming) ftand 'diversity' case sensitive with stemming distance at most 5 words]

The full-text index is optimized to support all full-text features of the XQuery Full-Text recommendation. BaseX extends the specification by offering a fuzzy match option. Fuzzy search is based on the Levenshtein algorithm; the longer query terms are, the more errors will be tolerated. Default "Case Sensitivity", "Stemming" and "Diacritics" options will be considered in the index creation. Consequently, all queries will be sped up which use the default index options.

5.4. What about the different indexes BaseX offers: which data structures are applied?

- Text Index: B-Tree: Both the text and attribute index are based on a B-Tree and support fast exact and range queries.
- Full-Text Index (Fuzzy Version): The standard full-text Index is implemented as sorted array structure. It is optimized for simple and fuzzy searches.
- Full-Text Index (Trie Version): A second full-text Index is implemented as a compressed trie. Its needs slightly more memory than the standard full-text index, but it supports more features, such as full wildcard search.

5.5. Do you support XML Updates?

BaseX provides internal commands to perform updates on XML (see also BaseX Commands). Moreover, the GUI provides a convenient way to perform updates on the data.

5.6. What can I do if I get Exceptions on huge XML files as input?

You can try to use the internal XML parser of BaseX instead of Java's own XML parser. To do this there are 2 ways: Moreover, the GUI provides a convenient way to perform updates on the data.

- Console Mode: basex -q "set intparse; ... create db ..."
- GUI Mode: File -> New -> Parsing -> Use internal XML parser

5.7. Can I protect a DB in BaseX with a password?

No, there is no password protection inside the BaseX architecture - but (at least in Linux) you can restrict the BaseX database directory to a specific user or group. Moreover, the GUI provides a convenient way to perform updates on the data.

6. Developer's Guide

6.1. Programming with BaseX API

This section decribes how to implement own applications using the BaseX API. So you can easily develop and expand BaseX with your own ideas and to your own needs. Here you can find a few instructions to start.

1. Creating a Database

To execute these examples you need to create a Main-Class and define the following:

```
// Creates a standard output stream
ConsoleOutput out = new ConsoleOutput(System.out);
// Creates a new database context, referencing the database.
Context context = new Context();
```

After defining the ConsoleOutput and the Context you can start with one of the following examples. The first example shows how to create a database from a xml file.

Figure 1. Creating a Database from a file

```
// Sets an option: activates command info output
new Set("Info", "on").execute(context, out);

// Chops whitespaces between text nodes
new Set("Chop", "on").execute(context, out);

// Creates a database from the specified file.
new CreateDB("input.xml", "DB1").execute(context, out);
```

The second example shows how to create a database from an input string.

Figure 2. Creating a Database from an input string

```
// XML string.
String xml = "<xml>This is a test</xml>";

// Creates a database for the specified input.
new CreateDB(xml, "DB2").execute(context, out);
```

The third example shows how to open an existing database after you have created it.

Figure 3. Opens an existing Database

```
// Opens an existing database
new Open("DB1").execute(context, out);

// Dumps information on the specified database context
new InfoDB().execute(context, out);
```

After all of the examples you have to close the Context and the ConsoleOutput

```
// Closes the database
context.close();
// Closes the output stream
out.close();
```

2. Executing XQuery

To execute these examples you need to create a Main-Class and define the following:

```
/** Sample query. */
private static final String QUERY = "<xml>Test</xml>/text()";
// Creates a standard output stream
ConsoleOutput out = new ConsoleOutput(System.out);
```

After defining the ConsoleOutput and the Query you can start with one of the following examples. The first example shows how to execute a query and create a result instances.

Figure 4. Creating a result instance

```
// Creates a result serializer
XMLSerializer serializer = new XMLSerializer(out);

// Creates a query instance
QueryProcessor processor = new QueryProcessor(QUERY);

// Executes the query.
Result result = processor.query();

// Serializes the result
result.serialize(serializer);

// Closes the query processor
processor.close();
```

The second example shows how to iterate through all results.

Figure 5. Iterating through all results

```
// Creates a query instance
processor = new QueryProcessor(QUERY);

// Returns a query iterator
Iter iterator = processor.iter();

// Uses an iterator to serialize the result
for(Item item : iterator) item.serialize(serializer);

// Closes the query processor
processor.close();
```

The third example shows how to execute a Query with the BaseX Command.

Figure 6. Using the BaseX command

```
// Creates a database context
Context context = new Context();

// Creates and executes a query
new XQuery(QUERY).execute(context, out);
```

After all of the examples you have to close the Context and the ConsoleOutput

```
// Closes the database
context.close();
// Closes the output stream
out.close();
```

3. Updating a Database

After you successfully created a Database you can update and modify the data on the following ways:

Insert a document:

```
// Inserts a document into the database;
// argument can be a file name or XML
// Position: 0 (ignored for documents)
// Target  : insert on root level
new Insert("fragment", "<doc>second</doc>", "0", "/").execute(context, out)
```

Delete:

```
// Deletes all attributes in the database.
new Delete("//@*").execute(context, out);
```

Insert a node:

```
// Inserts an element fragment into the database
// Position: 1 = as first child
```

```
// Target  : insert after all /doc elements...
new Insert("fragment", "<sub/>", "1", "/doc").execute(context, out);
```

You can do it the same way with new Update() and new Copy()

6.2. Programming with XML:DB API

This section decribes how to implement own applications using the XML:DB API. These classes are predefined and help to use every native XML Database with the same application implemented with the XML:DB API.

1. Executing XPath

To execute this example you need to create a Main-Class and define the following:

```
/** Database Driver. */
static final String DRIVER = "org.basex.api.xmldb.BXDatabase";
/** Name of the referenced database. */
static final String DBNAME = "xmldb:basex://localhost:1984/input";
/** Sample query. */
static final String QUERY = "//li";
```

After defining the DRIVER, the DBNAME and the QUERY you can start with the following example.

Figure 7. Executing XPath

```
  Collection col = null;

  try {
    Class c = Class.forName(DRIVER);

    Database database = (Database) c.newInstance();
    // Registers the Database.
    DatabaseManager.registerDatabase(database);
    // Receives the Database.
    col = DatabaseManager.getCollection(DBNAME);
    // Receives the XPathQueryService.
    XPathQueryService service =
      (XPathQueryService) col.getService("XPathQueryService", "1.0");
    // Executes the query and receives all results.
    ResourceSet resultSet = service.query(QUERY);
    // Iterator for ResultSets.
    ResourceIterator results = resultSet.getIterator();
    while(results.hasMoreResources()) {
      // Receives the next results.
      Resource res = results.nextResource();
      // Writing the result to the console.
      System.out.println((String) res.getContent());
    }
  } catch(XMLDBException e) {
    System.err.println("XML:DB Exception occured " + e.errorCode);
  } finally {
    if(col != null) col.close();
  }
}
```

2. Inserting a XML Document

To execute this example you need to create a Main-Class and define the following:

```
/** Database Driver. */
static final String DRIVER = "org.basex.api.xmldb.BXDatabase";
/** Name of the referenced database. */
static final String DBNAME = "xmldb:basex://localhost:1984/input";
```

After defining the DRIVER and the DBNAME you can start with the following example.

Figure 8. Inserting a XML Document

```
Collection col = null;

try {
  Class c = Class.forName(DRIVER);

  Database database = (Database) c.newInstance();
  // Registers the Database.
  DatabaseManager.registerDatabase(database);
  // Receives the Database.
  col = DatabaseManager.getCollection(DBNAME);
  // ID for the new Document.
  String id = "NewDocument";
  // Content of the new Document.
  String document = "This is the new Document!";
  // Creates a  new XMLResource with the ID.
  XMLResource resource = (XMLResource) col.createResource(id,
    XMLResource.RESOURCE_TYPE);
  // Sets the content of the XMLResource as the Document.
  resource.setContent(document);
  //Stores the Resource into the Database.
  col.storeResource(resource);
} catch (XMLDBException e) {
  System.err.println("XML:DB Exception occured " + e.errorCode);
} finally {
  if(col != null) col.close();
}
```

6.3. Programming with XQuery API

This section describes how to implement own applications using the XQuery API.

Executing XQuery

To execute this example you need to create a Main-Class and type the following:

Figure 9. Executing XQuery

```
// Gets the XQDataSource for the specified Driver.
XQDataSource source = (XQDataSource) Class.forName(DRIVER).newInstance();
// Creates an XQConnection
XQConnection conn = source.getConnection();
// Prepares the Expressionwith the Document and the Query.
XQPreparedExpression expr = conn.prepareExpression("doc('input')//li");
// Executes the XQuery query.
XQResultSequence result = expr.executeQuery();
// Gets all results of the execution.
while(result.next()) {
  // Prints the results to the console.
  System.out.println(result.getItemAsString(null));
}
```